Structure-Based Design of Combinatorial Libraries

by

Allan Geoffrey Skillman Jr.

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

PHARMACEUTICAL CHEMISTRY

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA SAN FRANCISCO

[redacted]

Date University Librarian

Degree Conferred: ....

To my grandfathers, G

who showed me that ever

To my

who taught me

To my mother, Julie Ma

who loved an

Dedication

To my grandfathers, Geoffrey Bruffey Skillman and Roland Gustav Scherer,

who showed me that even trout fishermen can become scientists and physicians.

To my father, Allan Geoffrey Skillman,

who taught me to be both a trout fisherman and a scientist.

To my mother, Julie Mary Skillman, and my wife, Jennifer Lyon Skillman,

who loved and supported me throughout this endeavor.

I have been very fort

work at play, and at home by

This thesis would not be poss

Thank you to those who insp

Allan Skillman, Eliza

Thank you to those who guide

Teri Klein, Jack Collin

Thank you to those who taught

Paul Wender, Steven B.

Jon Ellman, George Ke

Thank you to those who helped

Connie Oshiro, Donna H

Barbara Chapman, Tasir

Yax Sun, Todd Ewing, D

Jeremy Yang, Dolan Barg

Thank you to my collegues in the

Brian Shoichet, Dale Bog

Dan Gschwend, Keith Bur

Xiaoqin Zou, Seth Hopkin

Michelle Lamb, Takamitsu

Preface

I have been very fortunate in my scientific career to be surrounded at school, at work, at play, and at home by a an extraordinarily gifted and gracious set of individuals. This thesis would not be possible without the generous assistance of them all.

Thank you to those who inspired me;

Allan Skillman, Elizabeth Horsh, Hyman Rickover, and J. Andrew McCammon.

Thank you to those who guided me;

Teri Klein, Jack Collins, Fred Cohen, and Tack Kuntz.

Thank you to those who taught me;

Paul Wender, Steven Boxer, Arthur Kluge, Gilda Loew, Ken Dill, Peter Kollman, Jon Ellman, George Kenyon, and C. C. Wang.

Thank you to those who helped me;

Connie Oshiro, Donna Hendrix, Ronald Knegtel, Ellen Kick, Debby Camper, Barbara Chapman, Tasir Haque, Meg Stauber, Karl Maurer, Malin Young, Yax Sun, Todd Ewing, Diana Roe, Steve Hughes, Paul Boyer, John Somoza, Jeremy Yang, Dolan Eargle, and Shinichi Katakura.

Thank you to my collegues in the Kuntz group;

Brian Shoichet, Dale Bodian, Elaine Meng, Guy Bemis, Cindy Corwin, Dan Gschwend, Keith Burdick, Andy Good, Hans Briem, Luke Hoffman, Xiaoqin Zou, Seth Hopkins, Judy Hempel, Wei Wang, Juan Wang, Michelle Lamb, Takamitsu Kobayashi, Ken Bramald, Sam Toba, Jim Arnold, Alex Aronov, Kaisa

Thank you to those who he

Connie Oshiro, Don

Malin Young, Kent K

Mike Zuerlein and I.

Mark and Kelly Cole

Mike and Brenda Da

Freddy Chen, and Brad

Thank you to those who loved

Jennifer Lyon Skillman

Allan and Julie Skillm

Randy, Anne, Sophia, a

Daniel and Elizabeth K

and Cynthia Kok.

And thank you to the one who

Jenny, my loving wife

Many Grateful Thanks Indeed!

Portions of this dissertat

emic Publishers, Elsevier Scien

dual chapters for detailed permi

Alex Aronov, Kajsa Ljungberg, Andy Verras, Scott Pegg, and David Sullivan.

Thank you to those who befriended me;

Connie Oshiro, Donna Hendrix,

Malin Young, Kent Kirshenbaum,

Mike Zuerlein and Liz Penades,

Mark and Kelly Cole,

Mike and Brenda Davis,

Freddy Chen, and Bradford Chamberlain.

Thank you to those who loved and encouraged me;

Jennifer Lyon Skillman,

Allan and Julie Skillman,

Randy, Anne, Sophia, and Max Roberts,

Daniel and Elizabeth Kok,

and Cynthia Kok.

And thank you to the one who chose me (and then put up with me);

Jenny, my loving wife.

Many Grateful Thanks Indeed!

Portions of this dissertation are reprinted with kind permission from Kluwer Academic Publishers, Elsevier Science, and the American Chemical Society (please see individual chapters for detailed permissions and references).

Combinatorial chem

ligands at relatively low cost

based design principles oper

compounds. In chapters two

ware for structure-based desi

identify inhibitors of four enz

natorial libraries.

I have developed three

ial libraries. *UC_Select* is an

reagents for a virtual library us

gram that prepares virtual librar

CombiDock is a version of the

designing combinatorial librarie

synthetically accessable combin

ty, selectivity, and potency.

Chapter five describes id

mechanism of action. Chapter

novel mechanisms of action by

ter six describes our identificatio

HGXPRTase. This chapter emph

readily synthesized in a comb

Abstract

Combinatorial chemistry makes it possible to generate large families of potential ligands at relatively low cost. Merging combinatorial chemistry strategies with structure-based design principles opens the exploration of virtual libraries containing billions of compounds. In chapters two through four of this thesis, I address the development of software for structure-based design of combinatorial libraries. In chapters five through eight, I identify inhibitors of four enzymes and address critical questions regarding design of combinatorial libraries.

I have developed three tools for the efficient construction and design of combinatorial libraries. *UC_Select* is an internet-based tool that allows synthetic chemists to select reagents for a virtual library using common chemical nomenclature. *Diversify* is a program that prepares virtual libraries from virtual reagents using *in silico* chemical reactions. *CombiDock* is a version of the structure-based design program DOCK, optimized for designing combinatorial libraries. Together, these tools can be used to rapidly propose synthetically accessable combinatorial libraries that address optimization of bioavailability, selectivity, and potency.

Chapter five describes identification of potent HIV-1 RT inhibitors with a novel mechanism of action. Chapter five is remarkable for our discovery of inhibitors with a novel mechanisms of action by direct our design efforts to a selected portion of RT. Chapter six describes our identification of the first non substrate-analog inhibitors of *T. foetus* HGXPRTase. This chapter emphasizes selection of potential inhibitors whose analogs can be easily synthesized in a combinatorial fashion. We apply these library design tools to

human cathepsin D in chap

tion to demonstrating the st

ver diverse design of comb

selective, and medicinally se

demonstrates the utility of m

design process.

I have developed soft

ture-based design of synthetic

to investigate critical question

potent and selective inhibitors

human cathepsin D in chapter seven. Here we identify several potent inhibitors, in addition to demonstrating the superiority of structure-based design of combinatorial libraries over diverse design of combinatorial libraries. Finally, in chapter eight, we identify potent, selective, and medicinally suitable inhibitors of *P. falciparum* Plasmepsin II. This chapter demonstrates the utility of including surrogates for medicinal suitability in the library design process.

I have developed software tools which facilitate efficient construction and structure-based design of synthetically accessible combinatorial libraries. We used these tools to investigate critical questions about the design of combinatorial libraries and to identify potent and selective inhibitors of four medically apropos macromolecular targets.

Software Tool Developn

"UC Select:

Common Nomencl

Table of Contents

Chapter 1

"Introduction"

Chapter 2

"UC Select: Development and Implementation of A Common Nomenclature Method to Search Chemical Databases"

Chapter 3

"CombiDock"

Part 2: "Screening a Database of Combinatorial Libraries"

"Diversify: Compu

Chapter 4

"Diversify: Computer-Assisted Construction of Molecular Libraries"

Drug Design Applicatio

Chapter 5

"A Novel Mechanism for Inhibition of HIV-1 Reverse Transcriptase"

"Ratio

Blocking P

Chapter 6

“Rational Design of Novel Antimicrobials:

Blocking Purine Salvage in a Parasitic Protozoan”

"Structure-Bas

Yield Low N

Chapter 7

"Structure-Based Design and Combinatorial Chemistry Yield Low Nanomolar Inhibitors of Cathepsin D"

Chapter 8

"Single Digit Nanomolar, Low Molecular Weight, Non-Peptide Inhibitors of Malarial Aspartyl Protease Plasmepsin II"

"Con

Chapter 9

"Conclusions and Future Directions"

Appendix 1: Additional Publications

Appendix 2: UC_Select

Appendix 3: Diversify

Appendix 4: CombiDock

Chapter 2

Chapter 4

Chapter 5

Chapter 6

Chapter 7

List of Tables

Chapter 2

Chapter 4

Chapter 5

Chapter 6

Chapter 7

Chapter 8

Chapter 2

Chapter 3, Part 1

Chapter 3, Part2

List of Figures

Chapter 2

Chapter 3, Part 1

Chapter 3, Part2

Chapter 4

Chapter 5

Chapter 6

Chapter 7

Chapter 8

Chapter 5

Chapter 6

Chapter 7

Chapter 8

Chapter 1: Introduction

by

A. Geoffrey Skillman

Improving and ex
and science. From their in
been intellectually and me
century B.C., when Hippo
precieved physiological di
where diesases are unders
have continued to prescribe
While it must be appreciate
treatment are also major con
hold a fundamental role in th
cological regimens they con
humans long into the twenty

A wide variety of age
macological therapeutics. Th
namic function, as defined by
agonists or calcium channel b
gonize diseases (antihypertens
proteins (urokinase), peptides
elements (oxygen), and even n
ing diseases(2). These agents c
receptor agonists (albuterol) or
allosteric effectors including al
promoting or interrupting natur

Improving and extending human life has long been a cherished pursuit of society and science. From their inceptions, the disciplines of medicine and pharmacology have been intellectually and methodologically intertwined in this joint endeavor. From the 5th century B.C., when Hippocratic physicians began deciphering the connections between precieved physiological disorders, disease, and appropriate curatives, to the modern era, where diesases are understood, diagnosed, and treated at a molecular level, physicians have continued to prescribe pharmacologic treatmens to cure or palliate their patients(1). While it must be appreciated that surgical, behavioral, dietary, and mechanical modes of treatment are also major contributors to modern therapeutics, pharmacological methods hold a fundamental role in the treatment of most medical maladies. Drugs and the pharmacological regimens they comprise will continue to play an important role in the care of humans long into the twenty-first century.

A wide variety of agents acting by a myriad of mechanism can be considered pharmacological therapeutics. These agents can be classified according to their pharmacodynamic function, as defined by the target type and biological endpoint (beta-adrenergic agonists or calcium channel blockers) or according to the medical divisions used to categorize diseases (antihypertensives or neuroleptics). Nucleic acids (gene therapy), whole proteins (urokinase), peptides (insulin), small organic molecules (furosemide), molecular elements (oxygen), and even ions (potassium) are used as pharmacological agents in treating diseases(2). These agents can act in the body as enzyme inhibitors (methotrexate), receptor agonists (albuterol) or antagonists (propanolol), channel blockers (nifedipine), allosteric effectors including allosteric effectors of transcription factors (estrogen), or by promoting or interrupting natural complexes(epoetin alpha)(2). Considering the vast diversity of biological targets and pharmacological endpoints, the pharmacological targets currently utilized can be classified into relatively few categories including; enzymes (proteases in particular), transmembrane receptors, transcription factors, signal proteins (kinases and phosphorylases), and structural elements. Finally, each of these agents may be delivered by a variety of routes including: oral, intravenous, intranasal, intramuscular, intrathecal, subcutaneous, inhaled, per rectum, topical, and transdermal, with different pharmacokinetic properties and potencies in different formulations.

Initially drugs were almost solely natural products and were often delivered in impure or partially purified forms. As chemistry developed as a field, the purification and identification of small molecules, including natural products, became possible. Natural products remain a rich source of medicinally important molecules including aspirin, morphine, digoxin, cyclosporin, and lovastatin(3). Not only are natural products used as drugs, but beginning around the turn of the century, they also began to serve as templetes for develpment of synthetic and semi-synthetic analogs. Modern synthetic chemicsts continue to be educated and challenged through the synthesis of natural products(4). It is only in the last sixty years that medicinal chemistry has flourished to the point of identifying novel compounds with desired pharmacological properties and developing these "lead" compounds via the planned synthesis of related molecules(5). Further, it has only been the last twenty years that computational chemistry has been able to make a large impact on the design of small molecules with desired physical and biological properties(6, 7, 8, 9, 10, 11, 12).

The design of small molecules with specific properties can be framed as a constrained optimization problem. The optimized function is the binding energy (ΔG), the difference in free energy between being bound to the receptor and being in aqueous solvent. This optimization is constrained by the fundamental (and obvious) constraints that the small molecule solutions to the problem are bound first by the structural principles of chemistry and second by the complex biological systems in which they act. The biological constraints include the concepts that the ideal molecule will be orally available, soluble, non-toxic, non-teratogenic, not metabolized too rapidly or too slowly, not bind to serum proteins, able to cross biological membranes, and, most importantly, able to carry out the desired biological function.

Prediction of the biological function and potency of small molecules has long been a pursuit of computational chemistry (*vida infra*). While recent work defining molecular-structure surrogates for the ability to cross biological membranes has greatly enhanced our ability to design compounds with this property, we have only a rudimentary understanding of the molecular motifs which effect toxicity, metabolism, and oral availability(13). The balance of the biological constraints are fulfilled only on an empiric basis, thus introducing much of the risk to drug development.

The constraints that chemistry places on the solutions are that they must be molecules rather than collections of independent atoms. This constrains the topology, scale, flexibility, and physical properties of the compounds based upon the finite number of stable hybridization, valence, charge, and bonding states of atoms. Chemistry further constrains the molecular solutions to the subset of molecules which can be synthesized by known chemical reactions. Although the exact degree of this constraint is subject to small variations in knowledge, technology, technique, and resources, from larger perspective, it is a rigid constraint. Despite all of these constraints, it has been estimated that between

10^{50} and 10^{200} small molecules which fulfill all of these constraints are possible(14). Thus it has become useful to generate models of some of these small molecules and use these models to predict their physical and biological properties prior to the measurement of the property or indeed even before the synthesis of the compound.

Prediction of molecular properties is an exceedingly broad topic. We will focus first on prediction of properties of small organic molecules of pharmacological interest, and focus even further on prediction of the interaction of those molecules with macromolecules with known atomic-resolution structures. DOCK is one computer program for analyzing these interactions(15, 16, 17). In DOCK, the interactions of small molecules and macromolecules are estimated using a molecular mechanics force field with the assumptions that bonds are unbreakable and all interactions are pairwise. The potential also assumes all bond lengths and angles are fixed, and the torsion angles can be assumed to be fixed or allowed to move. The current potential has two enthalpic terms and no entropic term and takes no account of the thermodynamic cycle. The solvent electrostatic screening is modeled using a 4r distance dependent dielectric constant. The final assumption is that the charges are atom-centered point charges calculated using the method of Gasteiger(18).

$$\text{Dock Score} = \Sigma_{(i,j)}\ (A_iA_j/(r_{ij})^{12} - B_iB_j/(r_{ij})^{6} + 332.0\ q_iq_j/4(r_{ij})^{2})$$

(for details see Meng et.al.(17))

Despite these limitations, DOCK is still successful at screening databases of available compounds to identify low micromolar inhibitors(19, 20, 21, 22, 23, 24). This utility comes because DOCK is exceptional at identifying true negatives. DOCK can be used to quickly eliminate all the compounds which simply have no chance of binding in a target pocket. When screening the ACD, even if half of the small number of inhibitors are eliminated (sensitivity 50%) it is acceptable because DOCK generally eliminates more than

99% of the non-inhibitors (specificity >99%). This high specificity is essential, because the prevalence of inhibitors in the ACD database is very low. Even a two percent false positive rate would result in thousands of false positives, effectively swamping out the few inhibitors and yielding very low hit rates in the final selections. Additional screening to eliminate the false positives caused by systematic errors is extremely helpful. Compounds which have large hydrophobic groups extending into the solvent, too many buried hydrogen bond clashes, bias caused by excessive polarization (multiple halides or multiple nitro groups), and long chain aliphatics can be eliminated . Finally, clustering can be helpful so only a single compound from each structural class is assayed initially(25).

After a typical molecular docking exercise, the best scoring 100's to 1000's of compounds are examined, and between 20 and 100 compounds are purchased and assayed. Screening assays are usually run at concentrations between 10 and 1000 μM, resulting in "hit" rates (>50% inhibition in the single point assay) between 10 and 30 percent. In many cases, the best inhibitor identified by an experienced computational chemist has an IC_{50} or K_i between 1 and 10 μM. It is also common to follow-up the initial screening hits with similarity searches and further screening. In favorable cases, when the database contains many similar compounds, these searches can yield significant increases in binding potency (an order of magnitude or more). In addition, they are an outstanding source of ideas for synthetic optimization of the hits. Despite its approximations, when applied to the problem of database screening, DOCK is an outstanding example of the successful application of computational chemistry to make practical and prospective predictions about relevant biological interactions.

In the fall of 1994, the Kuntz group critically reviewed DOCK and the molecular docking field. The major

as a single low-energy (C

the target macromolecule

ligand-receptor enthalpy (

not enough molecules wer

itors. To expand this final

ing hit rates (1hit 5,000-1

binding energy for each c

inhibitors would be identif

illustrates the necessity to s

inhibitors.

Over the past five ye

Kuntz group and in the gene

addressed by two contrastin

ligand by a series of rigid low

constructed to fit into a partic

variety of incremental growth

developed a method to assess

the crystallographic structure

recently begun to address rec

ple methods are under devel

taneously improve scoring fun

tens or hundreds of compoun

docking field. The major limitations discerned at the time were: first, ligands were treated as a single low-energy (CONCORD(26)) conformation; second, only one conformation of the target macromolecule was considered (27); third, the scoring function only addressed ligand-receptor enthalpy(17), neglecting entropy, solvation, and desolvation; and finally, not enough molecules were being screened to consistently identify sub-micromolar inhibitors. To expand this final point, it was recognized that based on estimated random screening hit rates (1hit/5,000-10,000 compounds screened), *even with a perfect calculation of binding energy for each compound*, it was likely that only a handful of low micromolar inhibitors would be identified among the 100-200 thousand compounds in the ACD. This illustrates the necessity to screen additional molecules in order to identify more potent inhibitors.

Over the past five years, each of these problems has been addressed both within the Kuntz group and in the general scientific community. Ligand flexibility has been addressed by two contrasting philosophies. First, flexibase methods(28) represent each ligand by a series of rigid low energy conformations. Second, ligand conformations can be constructed to fit into a particular active site by either distance geometry methods(29) or a variety of incremental growth algorithms(30, 31, 32, 33). Knegtel, Kuntz, and Oshiro developed a method to assess areas of receptor structural variability from NMR or multiple crystallographic structure and dock to this receptor ensemble(34). Other groups have recently begun to address receptor side-chain and backbone flexibility(35). Further, multiple methods are under development to adapt molecular dynamics techniques to simultaneously improve scoring functions and incorporate receptor flexibility for screening of tens or hundreds of compounds(36, 37, 38). However, state of the art molecular docking continues to approximate

In 1994, the most

coulombic potential and

AMBER(17). Developme

over the past five years. M

functions developed to op

compounds(39, 40, 41). A

ge errors as low as 1 kcal

when extrapolated to more

sented in the training set. T

solvation terms into molecu

approximation(42), or by th

ods have recently seen dram

dreds of thousands of compo

predicting binding energies

In 1994, initial attem

taken the form of *de novo* d

variety of random (non rea

order to optimize their inte

in two critical ways. First,

thesized. Second, approx

synthesized compound w

the field ripe for the inte

continues to approximate the macromolecular target with a single rigid structure.

In 1994, the most common DOCK scoring function involved only two terms, a coulombic potential and a van der Waals 6-12 potential with parameters adapted from AMBER(17). Development of new scoring functions has been a topic of intense activity over the past five years. Much of the work has gone into a myriad of empirical scoring functions developed to optimize the fit of calculated and experimental ΔG's for a series of compounds(39, 40, 41). Although the initial fittings are often quite impressive, with average errors as low as 1 kcal/Mol, these methods often fail to produce such splendid results when extrapolated to more potent compounds or when applied to a receptor not represented in the training set. There have also been significant developments in inclusion of solvation terms into molecular mechanical force-fields, either by the Poisson-Boltzmann approximation(42), or by the generalized-Born method(43, 44, 45). Although these methods have recently seen dramatic increases in speed, they still cannot be used to screen hundreds of thousands of compounds; furthermore, it is not yet clear how accurate they are at predicting binding energies across a diverse set of examples.

In 1994, initial attempts to increase the number of compounds being screened had taken the form of *de novo* design programs(11, 39, 46, 47, 48, 49). These programs used a variety of random (non reaction-based) changes to the chemical structures of molecules in order to optimize their interactions with the receptor. De Novo design programs fell short in two critical ways. First, the molecules they generated were often too complex to be synthesized. Second, approximations in the scoring functions made it too risky that the newly synthesized compound would not make the predicted interaction. These limitations made the field ripe for the integration of small molecule combinatorial chemistry.

Combinatorial sy
logical polymers such as
of drug molecules has be
binatorial methods have c
libraries such as well kno
based protease inhibitor o
in both solid and liquid p
Regardless of the strategy,
more potential compounds

The integration of s
extremely successful becau
complement on another. C
thesize a remarkable numbe
they could synthesize is exp
methods allow rapid approxi
in a statistically significant p
structure-based design has th
ds offer the combinatorial c
pounds and focus on a subse
compounds. The combinator
pounds allow a computationa
number of compounds, where
do not carry such a heavy imp

Combinatorial synthesis has long been used in biological systems to build up biological polymers such as DNA, proteins, and polyketides. Recently, however, the synthesis of drug molecules has been revolutionized by combinatorial chemistry methods(50). Combinatorial methods have effected all areas of synthesis including drug-like "priviledged" libraries such as well known as receptor antagonist benzodiazepines and the mechanism-based protease inhibitor (hydroxyethyl)amines. Libraries are now commonly synthesized in both solid and liquid phases and in single compound as well as small mixture formats. Regardless of the strategy, one must confront the fundamental problem that there are many more potential compounds than can practically be synthesized.

The integration of structure-based design and combinatorial chemistry has been extremely successful because the strengths and weaknesses of these two powerful tools complement on another. Combinatorial synthesis methods allow a single chemist to synthesize a remarkable number of compounds; however, the potential number of compounds they could synthesize is expanded by an even greater number. Structure-based design methods allow rapid approximation of the binding constant of many potential compounds in a statistically significant population of compounds; however, in any individual case, structure-based design has the potential to fail. When integrated, the computational methods offer the combinatorial chemist a means to assess an entire library of virtual compounds and focus on a subset of those compounds enriched with the most interesting compounds. The combinatorial synthesis of tens, hundreds, or even thousands of compounds allow a computational chemist to make predictions on a statistically significant number of compounds, where the approximations necessary for computational feasability do not carry such a heavy impact. The complementary integration of combinatorial chemistry and structure-based

tion.

Combinatorial ch
ligands. Merging combin
ples allows the exploratio
developed three tools for
UC_Select is an internet-
virtual library (Chapter 2
generate virtual libraries f
is a variation of the structu
designing combinatorial lib
application of structure-bas
HIV-1 Reverse Transcriptas
Phosphoribosyltransferase (
Plepsin II (Chapter 8). The
design inhibitors of medicin

istry and structure-based design is a successful means to inhibitor discovery and optimization.

Combinatorial chemistry makes it possible to efficiently generate large families of ligands. Merging combinatorial chemistry strategies with structure-based design principles allows the exploration of virtual libraries containing billions of compounds. We have developed three tools for the efficient construction and design of combinatorial libraries. UC_Select is an internet-based tool that allows synthetic chemists to select reagents for a virtual library (Chapter 2). Diversify is a program that allows a computational chemist to generate virtual libraries for molecular docking (Chapter 4). The final tool, CombiDOCK, is a variation of the structure-based design program DOCK, which has been optimized for designing combinatorial libraries (Chapter 3). We describe these tools and demonstrate the application of structure-based design and combinatorial chemistry to the inhibition of HIV-1 Reverse Transcriptase (Chapter 5), *T. foetus* Hypoxanthine-Guanine-Xanthine-Phosphoribosyltransferase (Chapter 6), Cathepsin D (Chapter 7), and *P. falciparum* Plasmepsin II (Chapter 8). The tools described and demonstrated here can be used to rapidly design inhibitors of medicinally important enzymes.

Bibliography

1. A. E. Hanson, in *Ancient Medicine-Medicina Antiqua* L. T. Pearcy, Ed. (http://www.ea.pvt.k12.pa.us/medant, 1999).

2. B. Katzung, *Basic and Clinical Pharmacology* (Appleton and Lang, Stamford, CT, ed. 7th Edition, 1998).

3. A. D. Buss, R. D. Waigh, in *Burger's Medicinal Chemistry and Drug Discovery* M. E. Wolff, Ed. (John Wiley & Sons, New York, 1995), vol. 1, pp. 983-1033.

4. M. B. Smith, *Organic Synthesis* (McGraw-Hill, San Francisco, 1994).

5. A. Burger, in *Burger's Medicinal Chemistry and Drug Discovery* M. E. Wolff, Ed. (John Wiley & Sons, New York, 1995), vol. 1, pp. 3-8.

6. F. H. Stillinger, A. Rahman, *The Journal of Chemical Physics* **60**, 1545-1557 (1974).

7. N. L. Allinger, *The Journal of the American Chemical Society* **99**, 8127-7134 (1977).

8. A. T. Hagler, S. Lifson, *The Journal of the American Chemical Society* **96**, 5327-5335 (1974).

9. S. R. Niketic, K. Rasmussen, *The Consistent Force Field: A Documentation* (Springer Verlag, Berlin, 1977).

10. W. L. Jorgensen
Journal of Chemical Ph

11. P. J. Goodford, J.

12. S. J. Weiner, et al.
1984)

13. L. Z. Benet, et al.
Wolff, Ed. (John Wiley &

14. A. W. Czarnik, Ac

15. T. J. A. Ewing, I. D
1997).

16. I. D. Kuntz, J. M. B
Molecular Biology 161, 26

17. E. C. Meng, B. K. S
505-524 (1992).

18. J. Gasteiger, M. Mar

19. C. S. Ring, et al., Pro
States Of America 90, 3583-3

20. B. K. Shoichet, R. M
445-1450 (1993).

10. W. L. Jorgensen, J. Chandrasekhar, J. D. Madura, R. W. Impey, M. L. Klein, *The Journal of Chemical Physics* **79**, 926-935 (1983).

11. P. J. Goodford, *Journal of Medicinal Chemistry* **28**, 849-857 (1985).

12. S. J. Weiner, et al., *The Journal of the American Chemical Society* **106**, 765-784 (1984).

13. L. Z. Benet, et al., in *Burger's Medicinal Chemistry and Drug Discovery* M. E. Wolff, Ed. (John Wiley & Sons, New York, 1995), vol. 1, pp. 111-250.

14. A. W. Czarnik, *Accounts Of Chemical Research* **29**, 112-113 (1996).

15. T. J. A. Ewing, I. D. Kuntz, *Journal Of Computational Chemistry* **18**, 1175-1189 (1997).

16. I. D. Kuntz, J. M. Blaney, S. J. Oatley, R. Langridge, T. E. Ferrin, *Journal of Molecular Biology* **161**, 269 (1982).

17. E. C. Meng, B. K. Shoichet, I. D. Kuntz, *Journal Of Computational Chemistry* **13**, 505-524 (1992).

18. J. Gasteiger, M. Marsili, *Tetrahedron* **36**, 3219-3288 (1980).

19. C. S. Ring, et al., *Proceedings Of the National Academy Of Sciences Of the United States Of America* **90**, 3583-3587 (1993).

20. B. K. Shoichet, R. M. Stroud, D. V. Santi, I. D. Kuntz, K. M. Perry, *Science* **259**, 1445-1450 (1993).

21. R. L. Desjarlais.

United States Of Americ

22. D. A. Gschwend.

Function and Genetics 2

23. D. L. Bodian, et a

24. L. R. Hoffman, I.

1997).

25. T. S. Haque, et al.

26. A. Rusinko, J. M. S

CORD. 192nd American C

1986).

27. I. D. Kuntz, *Scienc*

28. M. D. Miller, S. K. K

puter-Aided Molecular Desi

29. J. M. Blaney, DGEO

30. A. R. Leach, I. D. Ku

1992).

31. T. J. A. Ewing, A. G.

21. R. L. Desjarlais, et al., *Proceedings Of the National Academy Of Sciences Of the United States Of America* **87**, 6644-6648 (1990).

22. D. A. Gschwend, W. Sirawaraporn, D. V. Santi, I. D. Kuntz, *Proteins-Structure Function and Genetics* **29**, 59-67 (1997).

23. D. L. Bodian, et al., *Biochemistry* **32**, 2967-2978 (1993).

24. L. R. Hoffman, I. D. Kuntz, J. M. White, *Journal Of Virology* **71**, 8808-8820 (1997).

25. T. S. Haque, et al., *Journal Of Medicinal Chemistry* **42**, 1428-1440 (1999).

26. A. Rusinko, J. M. Skell, R. Balducci, C. M. McGarity, R. S. Pearlman, CONCORD, 192nd American Chemical Society National Meeting, Anaheim, California (1986).

27. I. D. Kuntz, *Science* **257**, 1078-1082 (1992).

28. M. D. Miller, S. K. Kearsley, D. J. Underwood, R. P. Sheridan, *Journal Of Computer-Aided Molecular Design* **8**, 153-174 (1994).

29. J. M. Blaney, DGEOM, Metaphorics Inc. (San Franciso, 1990).

30. A. R. Leach, I. D. Kuntz, *Journal Of Computational Chemistry* **13**, 730-748 (1992).

31. T. J. A. Ewing, A. G. Skillman, I. D. Kuntz, *in preparation* (1999).

32. M. Rarey, B. Kr

470-489 (1996).

33. W. Welch, J. Ru

34. R. M. A. Knegte

424-440 (1997).

35. V. Schnecke, L. A

Solvated Binding Sites w

burg, Germany (1999).

36. W. Wang, J. Wang,

394-402 (1999).

37. A. E. Mark, Rapid

ities, Virtual Screening Wo

38. T. Hansson, J. Marc

12, 27-35 (1998).

39. H. J. Bohm, *Journal*

40. T. I. Oprea, G. R. Ma

1998).

41. D. Bouzida, et al., *Int*

42. B. Honig, A. Nicholls,

32. M. Rarey, B. Kramer, T. Lengauer, G. Klebe, *Journal Of Molecular Biology* **261**, 470-489 (1996).

33. W. Welch, J. Ruppert, A. N. Jain, *Chemistry & Biology* **3**, 449-462 (1996).

34. R. M. A. Knegtel, I. D. Kuntz, C. M. Oshiro, *Journal Of Molecular Biology* **266**, 424-440 (1997).

35. V. Schnecke, L. A. Kuhn, Screening and Docking of Flexible Organic Ligands to Solvated Binding Sites with Induced Complementarity, Virtual Screening Workshop, Marburg, Germany (1999).

36. W. Wang, J. Wang, P. A. Kollman, *Proteins: Structure, Function, and Genetics* **34**, 394-402 (1999).

37. A. E. Mark, Rapid Non-empirical Methods for Estimating Relative Binding Affinities, Virtual Screening Workshop, Marburg, Germany (1999).

38. T. Hansson, J. Marelius, J. Aqvist, *Journal Of Computer-Aided Molecular Design* **12**, 27-35 (1998).

39. H. J. Bohm, *Journal Of Computer-Aided Molecular Design* **6**, 593-606 (1992).

40. T. I. Oprea, G. R. Marshall, *Perspectives In Drug Discovery and Design* **9**, 35-61 (1998).

41. D. Bouzida, et al., *International Journal Of Quantum Chemistry* **72**, 73-84 (1999).

42. B. Honig, A. Nicholls, *Science* **268**, 1144-1149 (1995).

43. M. Scarsi, J. Apostolakis, A. Caflisch, *Journal Of Physical Chemistry a* **101**, 8098-8106 (1997).

44. M. Scarsi, J. Apostolakis, A. Caflisch, *Journal Of Physical Chemistry B* **102**, 3637-3641 (1998).

45. X. Zou, Y. Sun, I. D. Kuntz, *Journal of the American Chemical Society* (1999).

46. R. A. Lewis, A. R. Leach, *Journal Of Computer-Aided Molecular Design* **8**, 467-475 (1994).

47. A. Miranker, M. Karplus, *Proteins-Structure Function and Genetics* **23**, 472-490 (1995).

48. G. Lauri, P. A. Bartlett, *Journal Of Computer-Aided Molecular Design* **8**, 51-66 (1994).

49. J. B. Moon, W. J. Howe, *Proteins-Structure Function and Genetics* **11**, 314-328 (1991).

50. B. A. Bunin, J. A. Ellman, *Journal Of the American Chemical Society* **114**, 10997-10998 (1992).

At first blush, it
developed as part of this
ceutical chemistry. On t
integration of chemistry
on the premise that there
several fields.

After completing
sing "fun" projects. I rea
of computer program desi
ming with HTML and CG
pages on the fly (Common
sits grew UC_Select, a p
about one year earlier at th
of the "zero cost seat". UC
shortcomings I'd seen thr
Ellen Kick and on the HIV
Maurer. Finally, in late M
tures, and flow based on se
whose input, friendship, a
that UC_Select has been v
have received electronic lic
tions, such as Stanford Un

Prologue to Chapter 2

At first blush, it is ironic that UC_Select, the most innovative program I have developed as part of this thesis, came into being only after I relaxed my focus on pharmaceutical chemistry. On the other hand, perhaps it is fitting that the best idea was from an integration of chemistry and computational knowledge, when my training has been based on the premise that there are unique insights to be gained by working at the interface of several fields.

After completing my oral exam in March of 1997, I decided to spend a month pursuing "fun" projects. I read The Essence of Program Design, a book about the philosophy of computer program design principles, and Foundations of World Wide Web Programming with HTML and CGI, a book about computer programs to generate interactive web pages on the fly (Common Gateway Interface or CGI programs). Out of these "fun" pursuits grew UC_Select, a program built with the Daylight CGI toolkits I'd originally heard about one year earlier at the MUG 96' conference during Dave Weininger's presentation of the "zero cost seat". UC_Select is reagent selection software based on the needs and shortcomings I'd seen through my experience working on the cathepsin D project with Ellen Kick and on the HIV-1 reverse transcriptase project with Meg Stauber and Karl Maurer. Finally, in late March 1997, I developed the program specifications, datastructures, and flow based on several long conversations with Meg Stauber, a synthetic chemist whose input, friendship, and great blocking I will always appreciate. Happily, I can report that UC_Select has been very well received in the medicinal chemistry community. We have received electronic license agreements from 49 institutions, including academic institutions, such as Stanford University, University of California, San Diego, University of

Copenhagen, University of Tokyo, and Erasmus University in Rotterdam as well as from pharmaceutical companies such as Abbott, Glaxo Wellcome, Ontogen, Parke-Davis, Pfizer, Smith Kline Beecham, and Boeringer Ingleheim.

Chapter 2

UC_Select: Development and Implementation of A Common Nomenclature Method to Search Chemical Databases.

by

A. Geoffrey Skillman, Tasir Haque, and Irwin D. Kuntz

presented as "Recursive SMARTS for Synthetic Chemists." at MUG98, February 1998, Santa Fe, New Mexico.

Abstract

Combinatorial l
compounds which have
advantage of a combina
molecules which practic
program which allows e
appropriate for a combin
two examples. First, we c
hydroxyethylamine con
and 2,221 acylating agent
able Chemicals Database
can be easily derivatized u
cules we identified in the A
leged" subset of the ACD c
lead compounds; therefore
efforts.

Introduction

A general problem i
organizing the pertinent inf
people with the knowledge a
information to them in a form

Abstract

Combinatorial libraries have become an extremely efficient tool for identifying compounds which have specific biological and physical properties. In order to take full advantage of a combinatorial library, you must be able to describe each of the potential molecules which practically can be synthesized in that library. UC_Select is a web-based program which allows chemists to use common nomenclature to select chemical reagents appropriate for a combinatorial library synthesis. We elucidate the utility of this tool in two examples. First, we demonstrate selection of amine and acylating reagents for a (hydroxyethyl)amine combinatorial library. A set of 426 primary amines and sets of 1,752 and 2,221 acylating agents compatible with the synthesis were identified from the Available Chemicals Database (ACD). Second, we use UC_Select to identify molecules which can be easily derivatized using common combinatorial reactions. Each of the 5286 molecules we identified in the ACD represents a potential combinatorial library. This "privileged" subset of the ACD contains molecules which have the potential to be outstanding lead compounds; therefore, we may wish to give them special attention in screening efforts.

Introduction

A general problem in information science consists of: defining, collecting, and organizing the pertinent information necessary to solve a problem; next, identifying the people with the knowledge and skills to solve the problem; and finally, delivering the information to them in a format they can understand and utilize by a means which they will find practical and efficient(1). Today, collection and organization of data occurs on computers, so the information must be stored by a means which is efficient for storage as well as retrieval by computational methods. Too often, information systems are built to the developer's specification rather than the appropriate end user's specification. Indeed, a significant portion of solving an information problem can be determining who is most qualified to use the information. Delivering the appropriate information to the wrong user can be at least as deleterious as delivering incorrect information to the appropriate user. The information must be made accessible to the final user by a method which allows them to intuitively identify and retrieve the information by a procedure which parallels the way they think about solving a problem. Finally, the information should be delivered in a format which is convenient for the end user to utilize and must include all details necessary to solve the problem.

Definition of the scope and size of a virtual library based on a well-defined combinatorial synthesis is a problem which falls into this framework of information-based problem solving. The synthetic chemists familiar with the scope of the reactions used to make the library are the best qualified to define the virtual library. Thus, the databases of available chemicals must be delivered to synthetic chemists in a format they understand, find useful, and are willing to use. These databases of available compounds are large, containing hundreds of thousands of very heterogeneous compounds, each with accompanying physical biological, medicinal, or commercial data(2). Efficient storage of chemical information is a long-standing problem which has been solved by linear string representations of chemical compounds such as Wiswesser Line Notation (WLN)(3) or SMILES(4,5). These representations contain all of the atomic content and bonding connectivity of the molecules, and can be

tations can be canonical

and retrieval as well[4]

Despite the adva

thetic chemist to describ

nomenclature. However,

chemists often speak of

formal incarnation grew

gity, its ease of use and

about reagent selection. A

Kekulé structural diagram

mented in both the xxMer

preferred method for iden

rational library reagents it

specificity and without an

searches. Further, these me

chemist's laboratory and in

The importance of convenie

small libraries, many chemi

rather than from a more tho

To overcome non-spe

grams, more complex search

SMARTS search strings are

molecules, and can be "read" directly as text by a practiced user. Further, the text representations can be canonicalized allowing unique representation as well as reliable data storage and retrieval as well(4).

Despite the advantages of these solutions, they are not an intuitive means for a synthetic chemist to describe reagents. Chemical structures are formally described by IUPAC nomenclature. However, this system is far too rigid and cumbersome for everyday use, so chemists often speak of chemicals using the common nomenclature, from which the more formal incarnation grew. Although common nomenclature allows for some points of ambiguity, its ease of use and flexibility make it central to the way synthetic chemists think about reagent selection. Another avenue for chemists to designate molecules is through Kekulé structural diagrams. Search methods based on this technique have been implemented in both the xvMerlin(6) and ISIS-base(7) search engine clients. While this is the preferred method for identification of individual compounds, when searching for combinatorial library reagents it becomes quite time consuming, lacking in functional group specificity and without an easy means to carry out complex, multi-functional group searches. Further, these methods suffer because they are often not directly available in the chemist's laboratory and involve a moderate learning curve to become an efficient user. The importance of convenient information access is emphasized by the anecdote that for small libraries, many chemists prefer to select reagents from a chemical vendor catalog rather than from a more thorough ISIS or Merlin search.

To overcome non-specificity of substructure searches derived from Kekulé diagrams, more complex search languages such as SMARTS have been developed(8). SMARTS search strings are a superset of SMILES strings and include versatile search features such as atom and bond wildcards, boolean functions, as well as valence, charge, connectivity, and stereochemistry specifications. SMARTS also contains two other features which are essential for developing sophisticated searches. First, any search can define the necessary *environment* of an atom (or functional group) as well as the atom (or functional group) itself. Second, any SMARTS search string can be associated with a variable name. These variables can then be used in subsequent SMARTS strings, being recursively replaced before the search is carried out. These features allow complex search strings to be devised, developed, and maintained in a reliable manner.

Here we describe the development of a search method based on common nomenclature. We have developed specific SMARTS strings for about seventy common functional groups, atom types, steric definitions, and relative atomic positions. By identifying and utilizing a "root atom" for each functional group, we have constructed the SMARTS so that they can be easily combined with boolean functions or modified with relative positions of nearby functional groups. As a second level of abstraction, we have combined these initial SMARTS to generate search strings for secondary functional considerations such as "beta hydroxy carboxylic acids." They have also been combined to form "meta" functional groups such as "nucleophiles" or "hydrogen-bond donors." These common nomenclature SMARTS definitions can be used as the intuitive language for synthetic chemists to search for combinatorial reagents.

In order to deliver this intuitive search method to synthetic chemists, we have developed an internet based search tool. The internet provides a very low-overhead method to distribute information to synthetic chemists(9). Most chemists are familiar with internet browsers and internet forms, which have been developed by third-party suppliers to be simple and comfortable interfaces. In our implementation, chemists define the search using a simple HTML form on any computer with an internet browser and web connection. The chemist chooses the reagent by selecting its primary (and secondary) functional groups from a list of common chemical nomenclature names. This simple interface has been enhanced to improve the control the chemist has during reagent selection. In addition to the primary and secondary functional groups, chemists can discard toxic, reactive, or highly metabolized reagents. Chemists can limit the search using physical properties such as molecular weight and rotatable bonds. After a search is defined and executed, pertinent information is returned to the chemist's browser so search results can be refined, utilized, and recorded. By having the search defined locally by the chemist, but carried out centrally, a single database can be maintained and searched by many chemists. We have developed an internet-based search interface which allows synthetic chemists to select chemical reagents for combinatorial libraries using common chemical nomenclature.

Methods

We have developed a recursive nomenclature to describe chemical reactions and functional groups using an extended set of Daylight's SMARTS search language. The nomenclature allows simple descriptions of functional groups and secondary interactions. However, when the nomenclature definitions are used to resolve these simple descriptions, the results are complex SMARTS search keys which define the relevant chemistry in a way which is both sensitive and specific. We have implemented these functional group descriptors into a common-nomenclature based search method. UC_Select is an internet-based (CGI)[10] implementation of our common chemical nomenclature search method which allows synthetic chemists to identify potential reagents for combinatorial libraries using their internet browser.

Common Nomenclature SMARTS definitions

We developed a general SMARTS representation for chemical functional groups. We desired these functional group definitions to be easily combined to describe boolean searches as well as the chemical and steric environments of functional groups. Initially, we analyzed a list of approximately 70 common functional groups(11) (*e.g.* carboxylic acid, alcohol, isocyanate), developed a set of common components of functional groups (*e.g.* hydroxyl, carbonyl), and identified the unique atoms which make up these structures (*e.g.* carbonyl carbon, carbonyl oxygen). For each functional group we defined a root atom as the atom which synthetic chemists refer to when identifying the relative positions of functional groups in common nomenclature (*e.g.* the carbonyl carbon is the root of carboxylic acids). SMARTS descriptions for each atom, component, and functional group were constructed in a hierarchical manner. The SMARTS for each functional group was defined as the root atom plus its environment (see appendix 1 of this chapter). The SMARTS for relative positions (*e.g.* alpha, beta, gamma, ortho, meta, and para) were adapted from the Daylight User Manual(8). The atom root of the SMARTS definitions allow functional groups to be combined for boolean searches (*e.g.* isocyanate *or* isothiocyanate), relative positions (*e.g.* meta-chloro aniline), or steric environment (*e.g.* beta-branched ester) in a systematic manner without excessive numbers of special cases (Table 1). Furthermore, the construction of complex search keys from simple building blocks minimizes debugging and maintenance of the definitions because corrections and updates need only be made in one place.

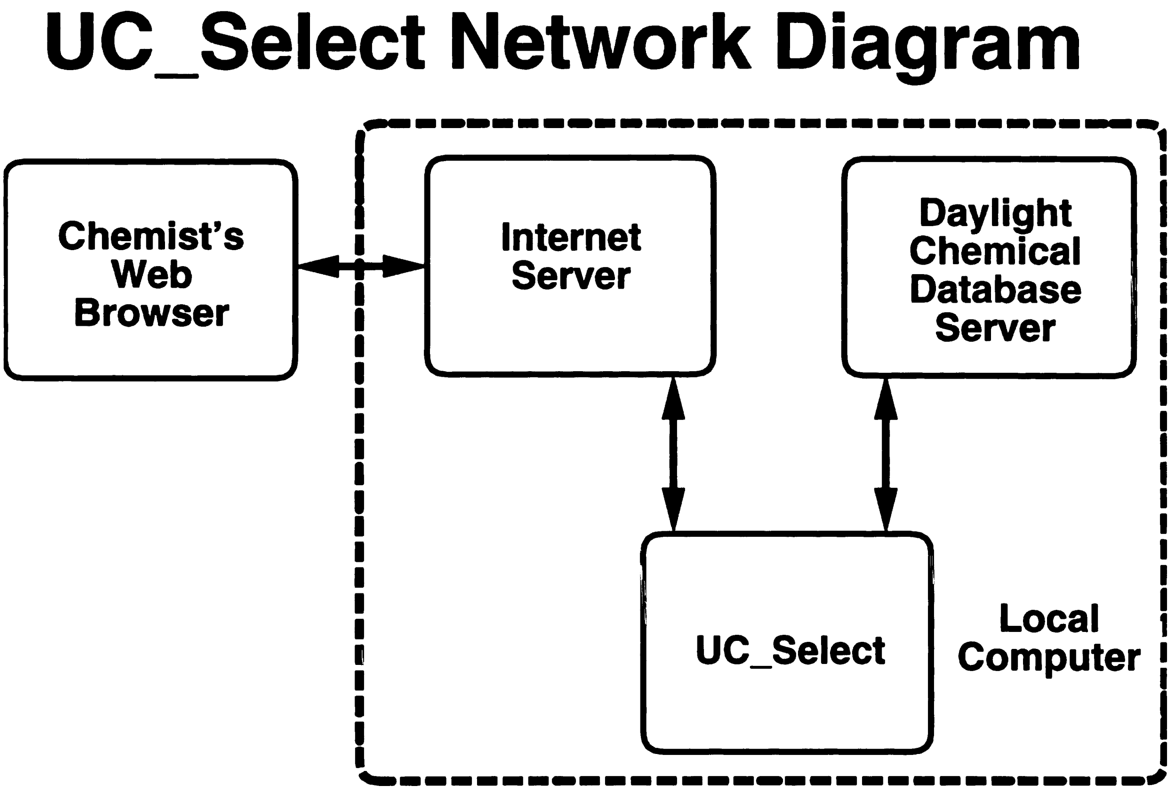


Figure 1: Network diagram for UC_Select. UC_Select is both a common gateway interface (CGI) program related to the web server and a Merlin client program, related to the Chemical database server.

UC_Select

UC_Select is a CGI program which is also a client program to Daylight's Merlin Server (figure 1)(6). Merlin is an extremely efficient search engine for managing chemical structures and information. UC_Select provides a synthetic chemist, via their web-browser, a simple interface to large databases of compounds. UC_Select allows chemists to build complex searches by filling out a simple HTML form(10). There are four phases to UC_Select's search process. First, an essential functional group is selected (generally the functional group which will participate in the combinatorial reaction) (figure 2).

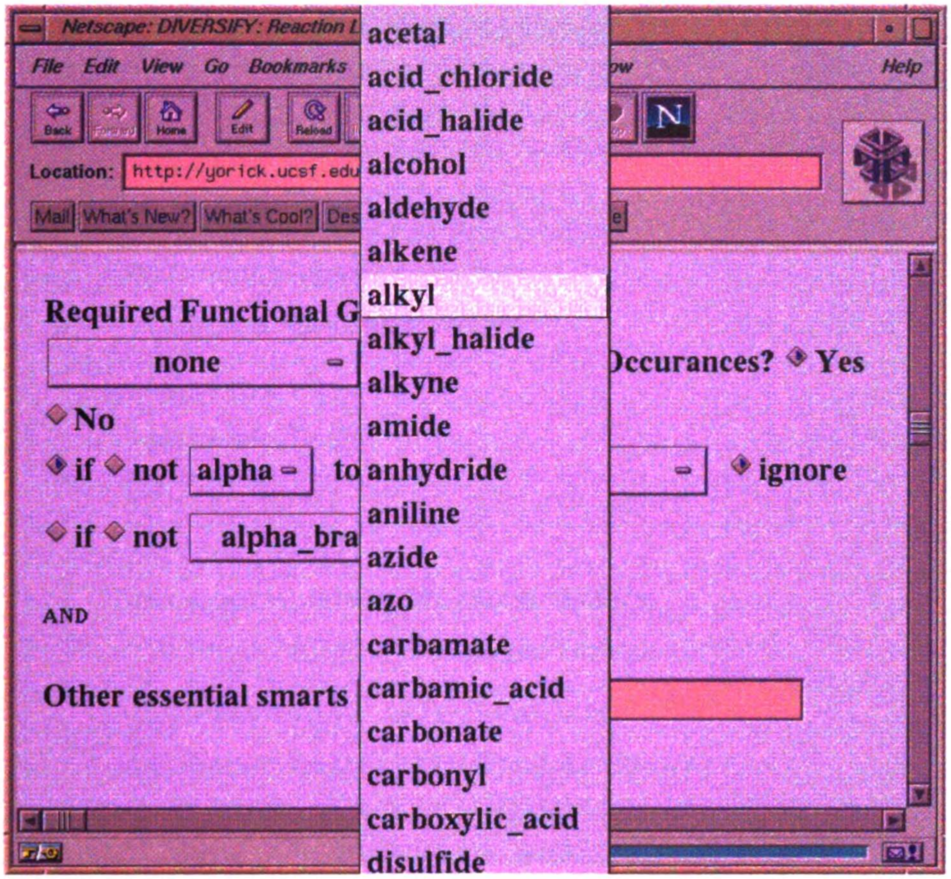
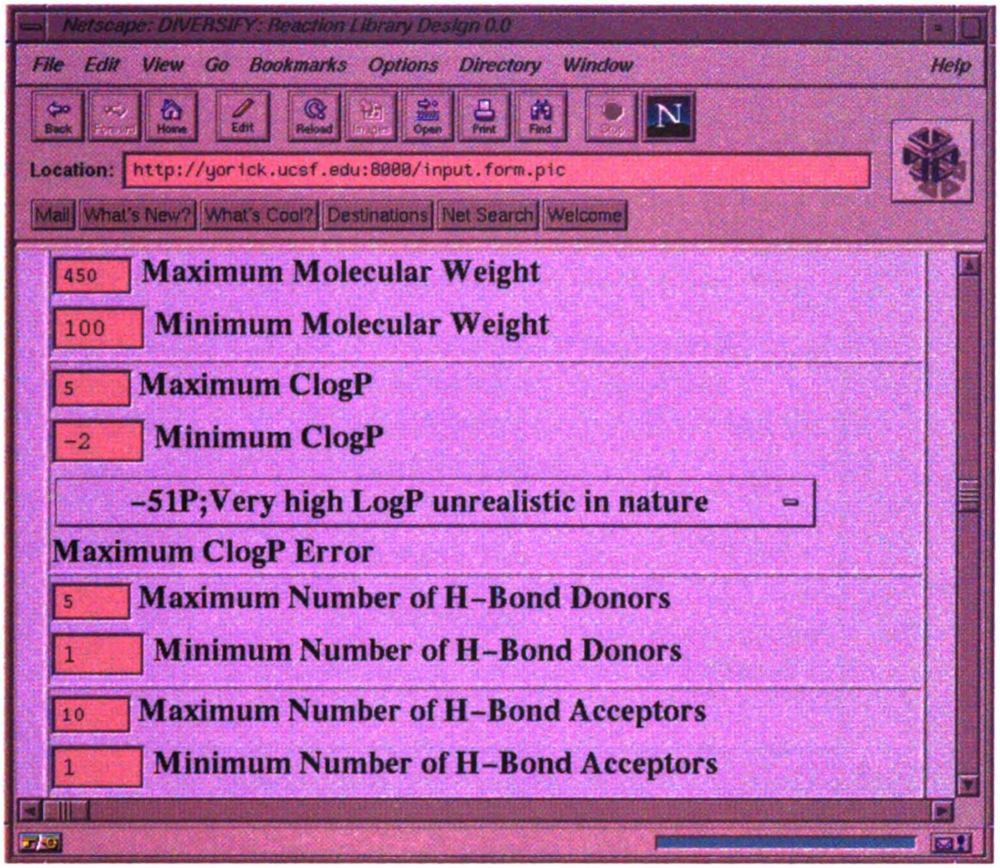


Figure 2: Essential Functional Group Selection. This illustrates the initial portion of the user interface as viewed within the Netscape browser. By selecting combinations of buttons and lists, the chemist builds the essential functional group as well as its chemical and steric environment.

This first step allows the chemist to specify secondary interactions, such as nearby functional groups and nearby steric environments, in the selection of an essential functional group. Second, compounds are culled according to a series of physical properties, including molecular weight, calculated logP(o/w)[8,12], number of rotatable bonds, number of hydrogen bond donors, number of hydrogen bond acceptors, and number of formal charges (figure 3). Third, compounds which contain medicinally less desirable functional groups are eliminated by default, however, the chemist has the option to keep any of these "bad" functional groups (figure 4). Fourth, the chemist selects functional


Netscape: DIVERSIFY: Reaction Library Design 0.0
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Edit Reload Images Open Print Find Stop
Mail What's New? What's Cool? Destinations Net Search Welcome

Location: http://yorick.ucsf.edu:8080/input.form.pic

450 Maximum Molecular Weight

100 Minimum Molecular Weight

5 Maximum ClogP

-2 Minimum ClogP

-51P;Very high LogP unrealistic in nature

Maximum ClogP Error

5 Maximum Number of H–Bond Donors

1 Minimum Number of H–Bond Donors

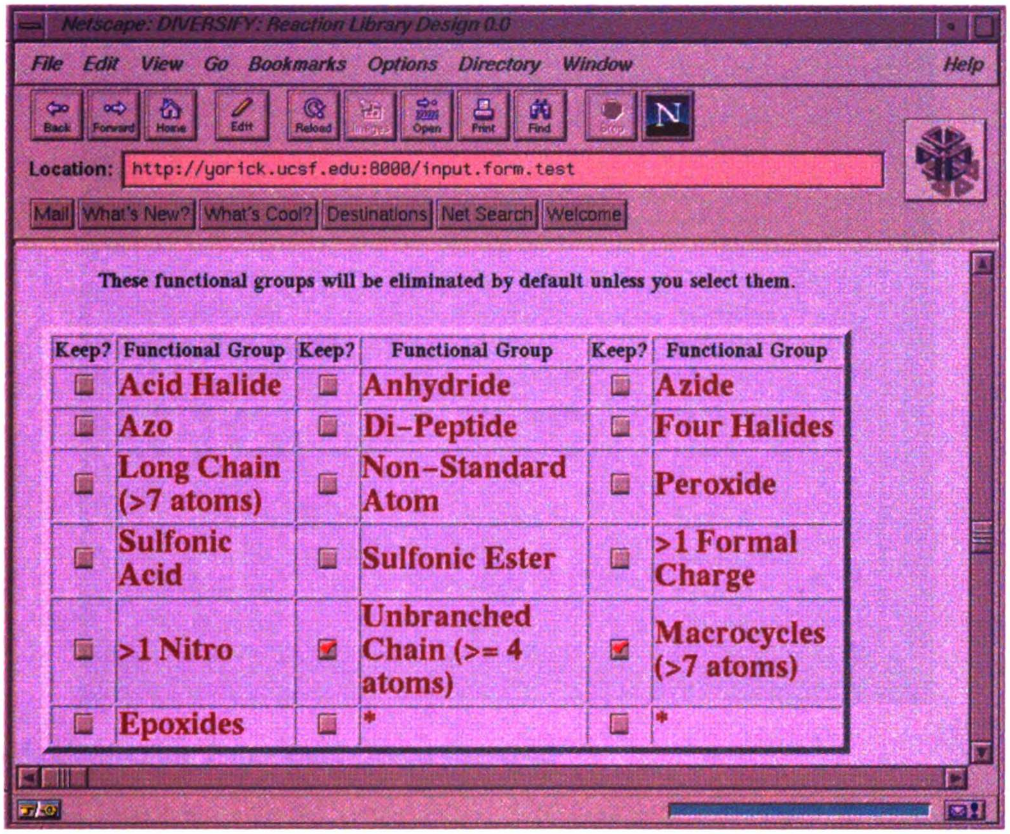
10 Maximum Number of H–Bond Acceptors

1 Minimum Number of H–Bond Acceptors

Figure 3: Physical Property Selection.

groups not compatible with the particular synthesis of interest from a large list of generally acceptable functional groups (figure 5). These compounds with incompatible functional groups are eliminated. Finally, based on more practical yet essential grounds, compounds only available from undesirable suppliers or compounds which are too expensive for a particular application are eliminated (figure 6).

Once the HTML form is filled out and submitted by the chemist, the web-server passes the results of the form to UC_Select as a set of variables. UC_Select parses the variables into a series of Merlin searches (SMARTS, SMILES, and parametric searches)


Netscape: DIVERSIFY: Reaction Library Design 0.0
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Edit Reload Images Open Print Find Stop
Mail What's New? What's Cool? Destinations Net Search Welcome

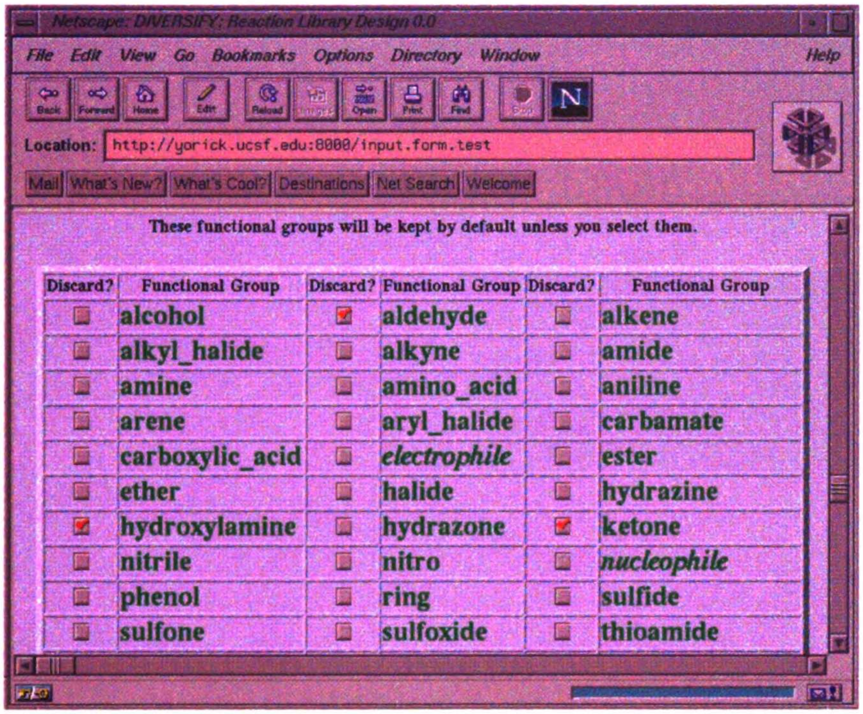
Location: http://yorick.ucsf.edu:8000/input.form.test

These functional groups will be eliminated by default unless you select them.

Keep?	Functional Group	Keep?	Functional Group	Keep?	Functional Group
□	Acid Halide	□	Anhydride	□	Azide
□	Azo	□	Di-Peptide	□	Four Halides
□	Long Chain (>7 atoms)	□	Non-Standard Atom	□	Peroxide
□	Sulfonic Acid	□	Sulfonic Ester	□	>1 Formal Charge
□	>1 Nitro	☑	Unbranched Chain (>= 4 atoms)	☑	Macrocycles (>7 atoms)
□	Epoxides	□	*	□	*

Figure 4: Undesired Functional Group Removal.

and carries out the searches. While the searches progress, UC_Select sends updates on the status of the searches back to the chemist via the chemist's web-browser (figure 7). When the search is complete, UC_Select sends the results to the chemist in one of several possible formats (selected by the chemist) which can include HTML links to ordering information on the compounds and small depictions of each compound (figure 8). This program brings the very powerful chemical search capabilities of Daylight's Merlin server into the hands of the user with the most knowledge about the synthetic problem at hand. The web-based form interface allows complex information to be encoded by the chemist in a nomenclature they already know while still taking advantage of many of the complex

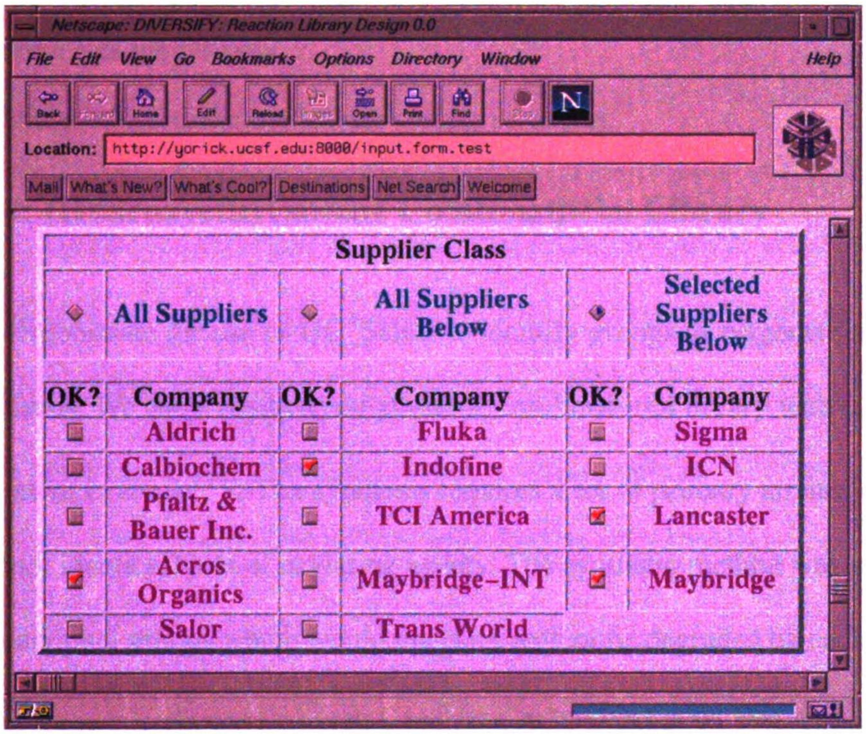

Netscape: DIVERSIFY: Reaction Library Design 0.0
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Edit Reload Images Open Print Find Stop
Mail What's New? What's Cool? Destinations Net Search Welcome

Location: http://yorick.ucsf.edu:8000/input.form.test

These functional groups will be kept by default unless you select them.

Discard?	Functional Group	Discard?	Functional Group	Discard?	Functional Group
□	alcohol	☒	aldehyde	□	alkene
□	alkyl_halide	□	alkyne	□	amide
□	amine	□	amino_acid	□	aniline
□	arene	□	aryl_halide	□	carbamate
□	carboxylic_acid	□	*electrophile*	□	ester
□	ether	□	halide	□	hydrazine
☒	hydroxylamine	□	hydrazone	☒	ketone
□	nitrile	□	nitro	□	*nucleophile*
□	phenol	□	ring	□	sulfide
□	sulfone	□	sulfoxide	□	thioamide

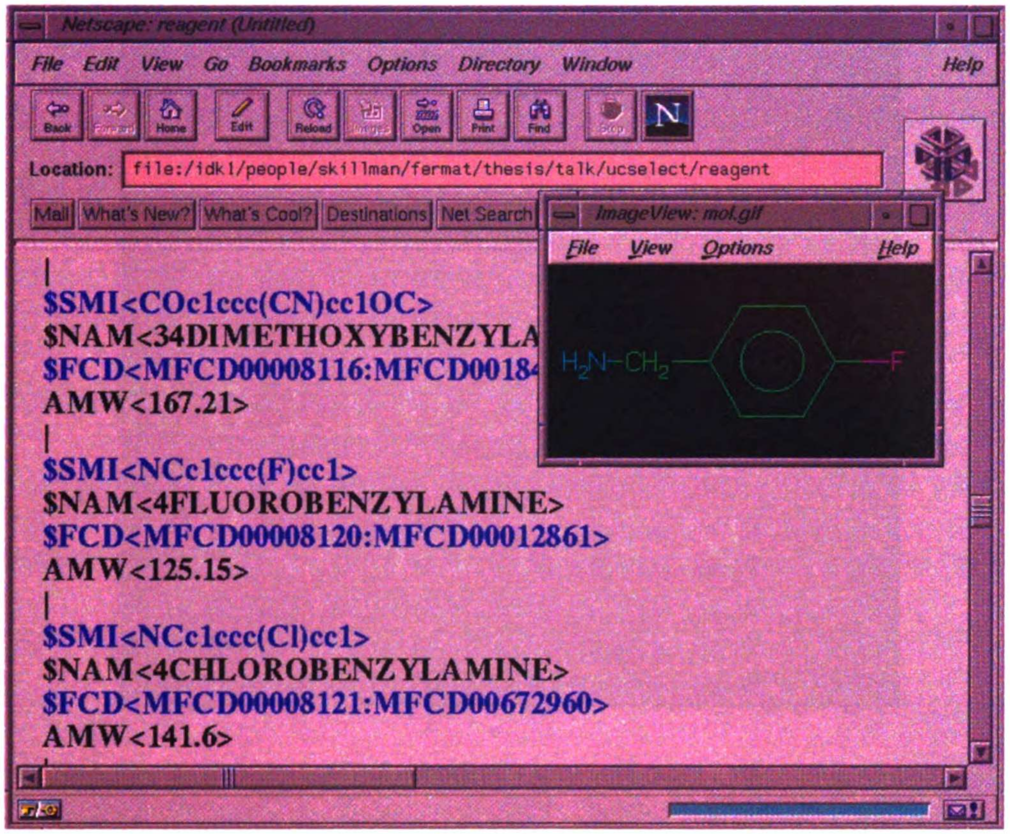
Figure 5: Conflicting Functional Group Removal


Netscape: DIVERSIFY: Reaction Library Design 0.0
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Edit Reload Images Open Print Find Stop
Mail What's New? What's Cool? Destinations Net Search Welcome

Location: http://yorick.ucsf.edu:8000/input.form.test

Supplier Class					
○	All Suppliers	○	All Suppliers Below	●	Selected Suppliers Below
OK?	Company	OK?	Company	OK?	Company
□	Aldrich	□	Fluka	□	Sigma
□	Calbiochem	☒	Indofine	□	ICN
□	Pfaltz & Bauer Inc.	□	TCI America	☒	Lancaster
☒	Acros Organics	□	Maybridge-INT	☒	Maybridge
□	Salor	□	Trans World		

Figure 6: Supplier Selection


Netscape: reagent (Untitled)
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Edit Reload Images Open Print Find Stop
Mail What's New? What's Cool? Destinations Net Search
ImageView: mol.gif
File View Options Help
H_2N–CH_2
F

Location: file:/idk1/people/skillman/fermat/thesis/talk/ucselect/reagent

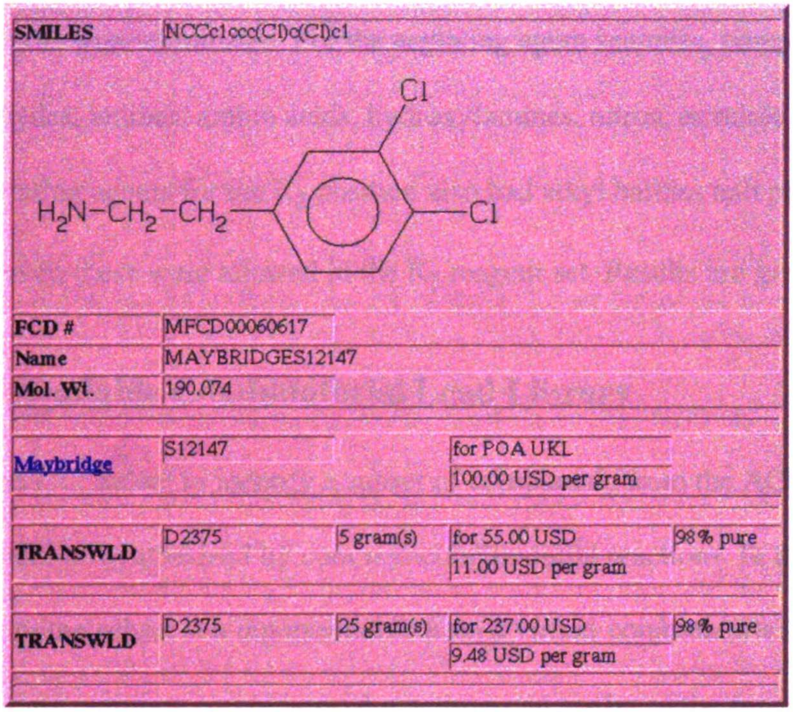
```
|
$SMI<COc1ccc(CN)cc1OC>
$NAM<34DIMETHOXYBENZYLA
$FCD<MFCD00008116:MFCD0018
AMW<167.21>
|
$SMI<NCc1ccc(F)cc1>
$NAM<4FLUOROBENZYLAMINE>
$FCD<MFCD00008120:MFCD00012861>
AMW<125.15>
|
$SMI<NCc1ccc(Cl)cc1>
$NAM<4CHLOROBENZYLAMINE>
$FCD<MFCD00008121:MFCD00672960>
AMW<141.6>
```

Figure 7: Search Results Display

search operations which make the Merlin system so useful.

Example 1: (Hydroxyethyl)amine Combinatorial Library

We demonstrate the use of UC_Select to identify groups of reagents for a combinatorial synthesis. We use a (hydroxyethyl)amine combinatorial library with three substituent groups as an example[13]. The synthesis requires a set of primary amines and two similar (but not identical) sets of acylating agents. The synthetic method was analyzed to determine functional groups which would interfere with or be degraded by each step of the synthesis and these criteria were used to search the ACD version 95.1[2]. The groups of acylating agents included carboxylic acids, acid halides, isocyanates, isothiocyanates,



SMILES	NCCc1ccc(Cl)c(Cl)c1			
FCD #	MFCD00060617			
Name	MAYBRIDGES12147			
Mol. Wt.	190.074			
Maybridge	S12147		for POA UKL 100.00 USD per gram	
TRANSWLD	D2375	5 gram(s)	for 55.00 USD 11.00 USD per gram	98% pure
TRANSWLD	D2375	25 gram(s)	for 237.00 USD 9.48 USD per gram	98% pure

Figure 8: Supplier and Ordering Information

and sulfonyl halides as the primary functional groups. Molecules with multiple copies of the essential functional group were eliminated. Reagents were selected from a list of those available from "preferred" suppliers (Table 2) and a maximum price limit of $100 per gram was set. The molecular weight of the side-chains were limited to 100-275 amu. In all of the searches, compounds with the following reactive, toxic, degradable, or difficult to model atoms or functional groups were removed: phosphoric acids; phosphonic acids; sulfonic acids; sulfonic esters; anhydrides; peroxides; azides; azos; atoms other than C, O, N, S, F, Cl, Br, I, H, or B; four or more halides; two or more formal charges; two or more nitro groups; a dipeptide; or a macrocycle.

For each search, additional functional groups which were not compatible with this specific synthesis were also removed. For the primary amine search, compounds with alcohols, aldehydes, alkyl halides, amino acids, carboxylic acids, hydroxylamines, nitros, phenols, and thiols were eliminated. For the acylating agent searches, compounds with alcohols, aldehydes, amines, amino acids, hydroxylamines, nitros, or thiols were eliminated. The acylating agents for the R_2 position also had alkyl halides and phenols removed; however, these were allowed in the R_3 reagent set. Results are given below.

Example 2: Available Combinatorial Lead Library

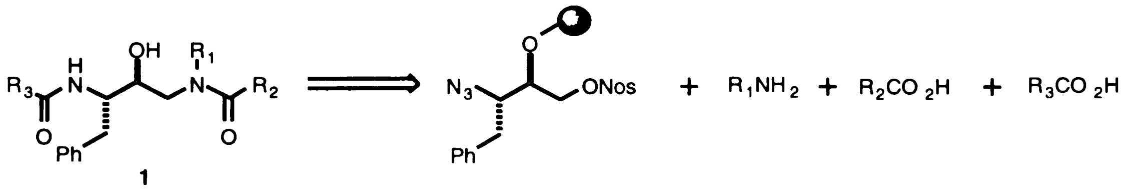
We used UC_Select to identify a subset of compounds from the ACD which have analogs that could be synthesized by common combinatorial reactions. In this manner, each molecule in the subset is a representative from a virtual combinatorial library. First, we selected twenty common combinatorial reactions and analyzed the functional group created in the product of the reaction (table 3). For instance, solid phase synthesis of peptides forms an amide functional group. We compiled a list of the functional groups from the common combinatorial reactions, which we'll refer to as the "combinatorial linkers." Further, we desired that each molecule represent a non-trivial combinatorial library, so we required that each molecule contain two rings each connected to a "combinatorial linker" by a path containing seven or fewer atoms. In addition, molecular weights were limited to 100-450 amu's. Molecules with reactive, unstable, and medicinally undesirable functional groups were also eliminated. UC_Select was used to carry out this search in the ACD 95.1(2). Approximately 14,500 compounds were identified which filled these search criteria. Unfortunately, many of them were medicinally uninteresting. To eliminate these, the additional criteria that at least one of the two rings be a heterocycle was applied. 5,286 available compounds were identified which fulfilled all of these search criteria. Not only are these compounds medicinally interesting, but they each represent a potential combinatorial library.

Results

Example 1: (Hydroxyethyl)amine Combinatorial Library

The original synthetic scheme for (hydroxyethyl)amine inhibitors can be seen in the retrosynthesis (figure 9)$^{(14)}$.

Figure 9



Components employed to prepare the (hydroxyethyl)amine library. Isocyanates and sulfonyl chlorides, which can be used to incorporate R_2 and R_3, provide ureas and sulfonamides, respectively.

The primary scaffold bound to solid phase first undergoes nucleophilic attach by a primary amine with the loss of the O-nosyl leaving group. Here the nucleophilic attack requires that the amines must not contain any facile leaving groups such as alkyl halides. Next, the second variable point is added by acylation of the newly created secondary amine. This constrains the second set of variable side-chains to not include strong nucleophiles. The scaffold azide is then reduced to a primary amine, unmasking the final site of acylation and requiring that the first two sets of side-chains not contain any easily reducible functional groups. The third variable group is then added by acylation of the newly unmasked primary amine. Finally, the library is acid cleaved from the solid phase with tri-floro-acetic acid. This requires that all three sets of side-chains must be stable to mild acids. All of these constraints were

Initially all sel

who combine to offer

costs. two major suppl

reduced the number of

were again considered

finding potent molecule

nated 16.360 compound

tional compounds. No h

hydrogen-bond donors

The R1 search w

tional groups or steric hi

vide *supra*) left a pool o

mary amines was carried

SMILES search for comp

pounds. When the specif

SMARTS. 1.420 compou

mary amines and were el

undesirable as well as th

from this set. The numbe

Table 3. Note that these r

were eliminated may ha

final set of remaining re

these constraints were considered when selecting reagents.

Initially all selections were made using the complete set of convenient suppliers who combine to offer 127,310 unique reagents (table 2). However, because of reagent costs, two major suppliers, Maybridge and SALOR were eliminated. This dramatically reduced the number of unique reagents to 41,803. Later, these more expensive reagents were again considered for more focused optimization libraries, where the likelihood of finding potent molecules was higher. The molecular weight limit of 100-275 amu eliminated 16,360 compounds, and a price limit of 100 US dollars per gram eliminated 87 additional compounds. No limits were placed on the charge, rotatable bonds, or number of hydrogen-bond donors or acceptors in this group.

The R1 search was for primary amines without any constraints on nearby functional groups or steric hindrance. The supplier selections and physical property constraints (*vida supra*) left a pool of 25,356 compounds from the ACD 95.1[2]. The search for primary amines was carried out in two stages, first SMILES, then SMARTS. The fast SMILES search for compounds containing aliphatic nitrogen identified 11,084 compounds. When the specificity of the search was constrained to primary amines with SMARTS, 1,420 compounds were identified, but 134 of these contained two or more primary amines and were eliminated. Undesired functional groups (those that are generally undesirable as well as those which conflict with the reaction at hand) were then removed from this set. The number of compounds removed for each functional group can be seen in table 3. Note that these numbers are order dependent because some compounds which were eliminated may have contained multiple undesired functional groups. However, the final set of remaining reagents was *not* dependent on the search order. Four-hundred twenty-six primary amines which fulfilled all of the criteria for our combinatorial synthesis were identified.

An individual search was carried out for each of the five acylating functional groups in order to track their contributions separately. All searches began with the same 25,352 available compounds as were used in the primary amine search (*vida supra*). Details of each search can be seen in table 4. 1,754 acylating agents compatible with the synthesis at R2 were identified (1,308 carboxylic acids, 207 acid halides, 30 isocyanates, 105 isothiocyanates, and 104 sulfonylhalides). The addition of 227 phenols and 240 alkyl halides allowed 2,221 compatible acylating agents at R3.

Example 2: Available Combinatorial Lead Library

UC_Select was used to select compounds from the ACD 95.1$^{(2)}$. Fourteen convenient and reliable suppliers were used resulting in 127,310 compounds (table 2). The molecular weight of the compounds (not including counter-ions) was limited to 100-450 amu, resulting in 111,704 compounds. One-hundred thirty seven compounds were eliminated because they cost more than $100/gram. The initial search was for a ring atom singly bound to a non-ring atom, yielding 97,512 compounds. Next, the primary search was carried out. First, the meta-functional groups "combi-linker" and "any connection" were created. The "combi-linker" functional group is the boolean "or" of the products of 20 common combinatorial reactions (*vida supra*, table 3). The "any connection" meta-functional group is a boolean "or" of from 0 to 7 bridging atoms. The primary search was for a ring atom with "any connection" to a "combi-linker" with "any connection" to an atom from a second ring. When applied, 17,542 compounds fulfilled this primary search. The final step was to eliminate compounds with the following undesired functional groups;

phosphoric acid (20).

esters (28), anhydride

unbranched chains >4

mal charges (3), aliph..

dipeptides (104), and r

fulfilled all of the searc

rings be a heterocycle.

dently carried out many

by the user, the combi-

library (5,286 members

Discussion

The development

here facilitates sophistica

describe chemical reagen

they find this search techn

know common nomenclat

necessary and most chem

familiarization. The intern

convenient locations inclu

giving better access to rea

thetic chemist, rather than

much richer detail of reage

phosphoric acid (20), phosphoric ester (19), acid halides (7), sulfonic acids (11), sulfonic esters (28), anhydrides (38), peroxides (0), unusual atoms (11), azides (33), azo (30), unbranched chains >4 atoms in length (680), four or more halides (1429), two or more formal charges (3), aliphatic chains >7 in length (394), two or more nitro groups (156), dipeptides (104), and macrocycles with >7 atoms (37). This left 14,542 compounds which fulfilled all of the search criteria. When the additional constraint that at least one of the rings be a heterocycle, the list was further reduced to 5,286. While UC_Select independently carried out many sub-searches in this case, these data represent only two searches by the user, the combi-linker library (14,542 members) and the heterocycle combi-linker library (5,286 members), emphasizing the ease and efficiency of UC_Select.

Discussion

The development of the common chemical nomenclature search method described here facilitates sophisticated chemical searching by synthetic chemists. Chemists regularly describe chemical reagents for a combinatorial library using common nomenclature, so they find this search technique natural and are willing to use it. Because chemists already know common nomenclature and most are familiar with internet browsers, little training is necessary and most chemists feel comfortable using UC_Select after a short period of familiarization. The internet interface allows chemists to use UC_Select in a variety of convenient locations including their laboratory desks and homes at no additional cost. By giving better access to reagent information, better decisions can be made. Having the synthetic chemist, rather than a computational chemist, define the scope of reagents yields a much richer detail of reagent selection. The synthetic chemist is encouraged to think more systematically about reagent selection than if they were asked to explain selection criteria to a computational chemist or if a computational chemist were to interpret the synthetic scheme and select reagents. Further, if the chemist doing the synthesis selects reagents, they can determine and incorporate information about how far they are willing to push a particular synthetic reaction. For example, in the second set of acylating agents in the (hydroxyethyl)amine inhibitors, although initially phenols and alcohols were excluded, when a second chemist selected reagents with UC_Select, he was reluctant to eliminate both alcohols and phenols, and eventually phenols were included in the library. By bringing reagent database information to chemists in a convenient and intuitive form, specific and sensitive sets of reagents are produced.

Although UC_Select is not the first example, it is too significant to not reiterate the power of internet conduits to efficiently disseminate chemical information. The classic model for chemical databases was to have a database server on one machine and client software to connected to the server. In the new model, UC_Select acts as the client program to the chemical database, but also interacts with a web-server to generate internet access to the database. In the old model, for each new user, the client software had to be installed and maintained on a local computer. In the new model, only one primary client exists (UC_Select) and any internet browser can act as the secondary client, eliminating the need for installation and maintenance of multiple client programs. Further, since the browsers and web-servers are developed and supported by third-party suppliers, no development of client software across multiple platforms is necessary. Not only does this make it less expensive to maintain current searching capabilities, it greatly facilitates vast expansion of accessibility to synthetic chemists (e.g. laboratory bench computers and home computers)[9]. This new server-primary client-internet server-secondary client (browser) paradigm raises significant licensing and access issues. In the old paradigm, licensing was simply controlled through client software and a maximum number of clients for each server. However, in the new paradigm, the chemical database server sees only one client, and through it, any number of people receive information via their internet browsers. When internet browsers are used as client software through CGI programs, such as UC_Select, it creates a new paradigm in information access. Internet-based distribution of chemical information and chemical calculations has the potential to have enormous impact on the process of drug discovery.

Although science has always been based on building new ideas “on the shoulders” of previous ideas, this is accentuated in the development of programs like UC_Select. Efficient design of programs to widely distribute chemical information require integration of many independent programing tools. Although integration may be problematic, incorporation of third-party expertise represented in their software is essential. UC_Select is based around the fundamental new idea of atom-rooted functional group SMARTS to allow development of a common chemical nomenclature search interface. The importance of this development has been magnified by integration with third-party software. MDL provides the primary chemical information which UC_Select distributes. Internet browsers running on a variety of platforms interpret and represent UC_Select HTML search forms to the user. The HTTPd server interprets variables from the browser and executes UC_Select while passing the appropriate variables as input[10]. Daylight’s MERLIN server carries out UC_Select’s SMILES, SMARTS, and numerical searches and returns the results. Programs like UC_Select can be even further developed by integration of more recent commercial programs as demonstrated by Pat Walters$^{(15)}$. There are many browser plug-ins which facilitate representation of chemical information through browsers such as the ChemDraw plug-in for two-dimensional representation. Perl, the most popular CGI programming language, also has several graphing utility functions which can be installed. A discussion of Java applications is beyond the scope of this chapter, but suffice it to say that it has enormous potential to impact the distribution of chemical information. The impact of new developments in chemical information and drug design can be significantly enhanced by incorporating them into an integrated system for information delivery.

Although the emphasis of this work has been on delivering reagent information to synthetic chemists, it is important to recognize and develop the secondary information their reagent searches represent. Development of a new synthetic scheme for a combinatorial library is a significant scientific achievement, often requiring chemical insight as well as multiple person-years of work. As more and more combinatorial scaffolds are synthesized, it becomes even more important to collect and archive the information in a way which allows continued use and refinement. With UC_Select, a combinatorial library can be thought of and cataloged as a series of molecular transformation, each associated with the variables (search keys) which define a search for compounds compatible with the synthesis. The search keys provide an excellent vehicle to define the scope of the library. This has several far reaching implications and advantages. First, one can develop a database of combinatorial libraries in this format, and when new reagent databases become available, they can simply be passed over the search variables to yield the new reagents compatible with each library synthesis. Second, when new chemical methods become available which may, for instance, allow a functional group to be compatible with the synthesis that formerly was excluded, the search can be modified simply by changing a single variable in the search description. Finally, it should be noted that the way a chemist is willing to think about a synthesis changes as a project moves along. For instance, while protecting incompatible functional groups is generally not acceptable in an exploratory library, the same protecting schemes may be acceptable in a smaller library for refining a lead compound. The scaffold plus reagent-selection key abstraction is a compact and efficient representation for archiving databases of combinatorial libraries.

The development of the Combi-linker meta-functional group, by boolean combination of the atom-rooted functional group SMARTS string, also merits further discussion. The select set of compounds identified by applying the Combi-linker search key to a database (*vida supra*) have some intriguing and desirable properties. First, all of these compounds are nominally drug-like, having reasonable molecular weights, numbers of rotatable bonds, formal charges, no common reactive or toxic functional groups, and being available from a "preferred" set of reagent suppliers. More significantly, each of these compounds would make an outstanding screening hit. Each compound in this subset may be derivatized by reactions commonly used in combinatorial syntheses. In essence, each of these 5,286 compounds represents an entire combinatorial library. Unfortunately, experimental screening of only 5,286 compounds, much less computational screening of this small set may not turn up any inhibitors because there may simply be no specific inhibitors of a given target in so small a sample. However, the potential to rapidly optimize these compounds should encourage us to pay special attention, perhaps through additional calculations, to this subset of the database. The Combi-linker meta-functional group is one demonstration of the power of being able to combine functional group SMARTS to describe and identify

Acknowledgm

We are gratefu

thank Meg Stauber. El

the practice of synthet

DayPerl and Jeremy Y

work was carried out u

poration and the Pfizer

describe and identify subsets of a database with complex sets of desirable properties.

Acknowledgments

We are grateful to Connie Oshiro and Donna Hendrix for helpful discussions. We thank Meg Stauber, Ellen Kick, Karl Maurer, and Tasir Haque for giving us a glimpse into the practice of synthetic chemistry. We would like to thank Alex Wong for help installing DayPerl and Jeremy Yang for helping us make all the Daylight pieces work together. This work was carried out under NCI grant GM31497 and with the support of the Daiichi Corporation and the Pfizer Corporation.

Bibliography

1 Dietrich. S. W. *Che*
S W., Ed.: John Wiley
2 *ACD: The Available*
Leandro. California. 1
3 Smith. E. G. *The W*
York. 1968.

4 Weininger. D.: Wein
Computer Sciences **198**

5 Weininger. D. *J. chem*

6 *Daylight Merlin Sever*
Santa Fe. New Mexico.

7 Adamson. G. W.: Bird
25. 90-92.

8 Weininger. D. *Dayligh*
Systems Inc.: Santa Fe.

9 Weininger. D. *The Zer*

10 Tittle. E.: Gaither. M.
gramming with HTML an
1995.

11 Streitwieser. A. J.: He
ed.: Macmillan Publishin

12 Hansch. C.: Leo. A. *S*

Bibliography

1)Dietrich, S. W. *Chemical Information Computing Systems in Drug Discovery*; Dietrich, S. W., Ed.; John Wiley & Sons: New York, 1995; Vol. 1, pp 416-496.

2)*ACD: The Available Chemicals Directory*; 95.1 ed.; Molecular Design Limited: San Leandro, California, 1995.

3)Smith, E. G. *The Wiswesser Line-Formula Chemical Notation*; McGraw-Hill: New York, 1968.

4)Weininger, D.; Weininger, A.; Weininger, J. L. *Journal of Chemical Information and Computer Sciences* **1989**, *29*, 97-101.

5)Weininger, D. *J. chem. Inf. comput. Sci* **1988**, *28*, 31-36.

6)*Daylight Merlin Sever*; version 4.61; Daylight Chemical Information Systems, Inc.: Santa Fe, New Mexico, 1999.

7)Adamson, G. W.; Bird, J. M.; Palmer, G.; Warr, W. A. *J. Chem. Inf. Comput. Sci.* **1985**, *25*, 90-92.

8)Weininger, D. *Daylight Software Version 4.61 Manual*; Daylight Chemical Information Systems Inc.: Santa Fe, New Mexico, 1999.

9)Weininger, D. *The Zero Cost Seat*; Mug96; Santa Fe, New Mexico, 1996.

10)Tittle, E.; Gaither, M.; Hassinger, S.; Erwin, M. *Foundations of World Wide Web Programming with HTML and CGI*; IDG Books Worldwide, Inc.: Foster City, California, 1995.

11)Streitwieser, A. J.; Heathcock, C. H. *Introduction to Organic Chemistry*; 3rd Edition ed.; Macmillan Publishing Company: New York, New York, 1985.

12)Hansch, C.; Leo, A. *Substituent Constants for Correlation Analysis in Chemistry and*

Biology; John Wiley & Sons: New York, 1979.

13)Kick, E. K.; Roe, D. C.; Skillman, A. G.; Liu, G. C.; Ewing, T. J. A.; Sun, Y. X.; Kuntz, I. D.; Ellman, J. A. *Chemistry & Biology* **1997**, *4*, 297-307.

14)Kick, E. K.; Ellman, J. A. *J. Med. Chem.* **1995**, *38*, 1427.

15)Walters, P.; Charifson, P. S.; Corkery, J. J.; Ajay; Murcko, M. A. *Virtual Screening - An Integrated Approach*; Virtual Screening Workshop; Marburg, Germany, 1999.

Tables

Table 1: Example Build-up Procedure for Functional Group SMARTS definitions.

Variable	SMARTS definition[a]
C_carbonyl	[C;$(C=[$O_carbonyl])]
hydroxyl	[O;$([H1&-0,H0&-1])]
C_carboxylic_acid	[$C_carbonyl;$(C[$hydroxyl]);$(C[#6,#1])]
carboxylic_acid	[$C_carboxylic_acid]
boolean search	[$isocyanate,$isothiocyanate]
relative position	[$analine;$([$ortho][$chloro])

a. For a full explanation of SMARTS definitions see the Daylight Manual(8). Briefly for this example; **[ ]** indicate the description of a single atom, **$( )** indicates the description of an atom's environment, **;** or **&** indicates a boolean and, **,** indicates a boolean or, ***** is the atom wildcard, **D** indicates the number of explicit bonds, **#** indicates the atomic number, and any variable (**$var**) is recursively replaced by the variable definition before searches are executed.

Table 2: Preferred Suppliers Used in Reagent Selection

Preferred Suppliers
Aldrich
Fluka
Sigma
Cal Biochem
ICN
Pfaltz and Bauer
TCI America
Lancaster
ACROS Organics
Maybridge International
TransWorld
Maybridge[a]
SALOR[a]

[a] Only used in the Available Combinatorial Lead Library reagent selection.

Table 3: A) Common Combinatorial Reactions

C-C Bond Formation	C-X Bond Formation
Suzuki	Amide
Heck	Ester
Stille	Urea, Thiourea
Wittig (Horner-Wadsworth-Emmons)	Carbamate
Organometallics	Mitsonobu
Reformansky	Reductive Amination (Imine Formation)
Enolate Alkylation	Alkylation:
Aldol	Amines, Amides
Michael Addition	Alcohols

Table 3: B) Combinatorial Linker Functional Groups

Functional Groups
amide
secondary amine
sulfonamide
urea
ester
ether
tertiary amine
carbamate
imino
hydrazone
thioether
thioamide
thiourea
thiocarbamate
thioester

Table 4: Number
Type a

Functional Group	Prin Am
Final Total Found	42
SMILES search	14.
SMARTS search	9.6
2 Copies of Primary Group	13
Alcohol	16
Thiol	13
Aldehyde	0
Carboxylic Acid	46
Nitro	14
Aniline	N.R
Amine	N.R
Phenol	40
Amino-Acid	100
Hydroxyl-amine	3
Alkyl-halide	23
Phosphonic Acid	9
Phosphonic Ester	9
Phosphoric Acid	12
Phosphoric Ester	1

Table 4: Number of Compounds Removed From Each Pool According to Search Type and Functional Group (25,356 compounds initially).

Functional Group	Primary Amine	Carboxylic Acid	Acid Halide	Iso-cyanate	Isothio-cyanate	Sulfonyl -halide
Final Total Found	**426**	**1308**	**207**	**30**	**105**	**104**
SMILES search	14,272	18,708	14,385	25,167	25,063	24,302
SMARTS search	9,664	3,371	10,664	0	0	989
2 Copies of Primary Group	134	493	23	19	10	0
Alcohol	162	215	0	0	0	0
Thiol	13	25	0	0	0	0
Aldehyde	0	16	0	0	0	1
Carboxylic Acid	467	N.R.[a]	0	N.R.[a]	N.R.[a]	N.R.[a]
Nitro	14	140	11	18	14	12
Aniline	N.R.[a]	N.R.[a]	1	97	137	8
Amine	N.R.[a]	510	1	0	5	2
Phenol	40	185	0	0	0	2
Amino-Acid	100	192	1	15	13	0
Hydroxyl-amine	3	3	0	0	3	0
Alkyl-halide	23	135	53	6	7	14
Phosphonic Acid	9	0	0	0	0	0
Phosphonic Ester	9	0	0	0	0	0
Phosphoric Acid	12	11	0	0	0	0
Phosphoric Ester	1	3	0	1	0	0

Table 4: Number
Type a

Functional Group	Pri An
Acid Halide (Carboxylic)	0
Sulfonic Acid	1
Sulfonic Ester	0
Anhydride	0
Peroxide	0
Non-standard Atoms[b]	1
Azide	0
Azo	0
Four Halides	0
>2 Formal Charges	0
dipeptides	0

a. N.R. = Not Remove

b. Many of these comp
constructed.

Table 4: Number of Compounds Removed From Each Pool According to Search Type and Functional Group (25,356 compounds initially).

Functional Group	Primary Amine	Carboxylic Acid	Acid Halide	Iso-cyanate	Isothio-cyanate	Sulfonyl -halide
Acid Halide (Carboxylic)	0	0	N.R.[a]	0	0	3
Sulfonic Acid	14	8	0	0	0	2
Sulfonic Ester	0	0	0	1	0	2
Anhydride	0	3	1	0	0	0
Peroxide	0	0	0	0	0	0
Non-standard Atoms[b]	11	12	0	2	1	0
Azide	0	2	0	0	0	0
Azo	0	5	1	0	0	0
Four Halides	0	11	8	0	0	2
>2 Formal Charges	0	0	0	0	0	0
dipeptides	0	0	0	0	0	0

a. N.R. = Not Removed.

b. Many of these compounds are pre-screened out of the ACD when the reagent database is being constructed.

Appendix 1: Functional Group SMARTS

Functional Group	Rooted SMARTS
amino_acid	[N][C;!$(C=*)][C;$(C=O);$(C[!#6])]
dipeptide	O=CCNC(=O)CN
two_charges	[$charge].[$charge]
two_nitros	[$nitro].[$nitro]
unbranched_chain	[R0;D2][R0;D2][R0;D2][R0;D2]
charge	[$acid,$base]
acid	[*&$(*=*)&$(*[$hydroxyl]),$malonic]
base	[n,N&!D3&!$(N*=[!#6])]
malonic	[C;H1,H2;$(C([$Ccarbonyl])[$Ccarbonyl])]
four_halides	[$halide].[$halide].[$halide].[$halide]
long_chain	[A;R0][A;R0][A;R0][A;R0][A;R0][A;R0][A;R0][A;R0]
macrocycle	[r8,r9,r10,r11,r12,r13,r14,r15,r16,r17,r18]
nonstandardatom	[!#1;!#2;!#3;!#5;!#6;!#7;!#8;!#9;!#11;!#12;!#15;!#16;!#17;!#19;!#20;!#35;!#53]
nucleophile	[$alcohol,$primary_amine,$secondary_amine,$aniline,$phenol,$azide,$hydrazine,$hydroxylamine,$peroxide,$thiol,$oxime]
alkyl	[$Calkyl]
combi_any	[$combi_fcn,$combi_linker]
combi_linker	[$amide,$secondary_amine,$sulfonamide,$urea,$ester,$ether,$tertiary_amine,$carbamate,$imino,$hydrazone,$thioether,$thioamide,$thiourea,$thiocarbamate,$thioester]
combi_fcn	[$ketone,$aldehyde,$primary_amine,$secondary_amine,$amide,$alkylating_agent,$aniline,$alcohol,$phenol,$thiol,$isocyanate,$isothiocyanate,$carboxylic_acid,$acid_halide,$hydrazine,$aryl_mono_BrI]
mono_alkene	[$alkene;$!($alkene.$alkene)]
mono_alkyne	[$alkyne;$!($alkyne.$alkyne)]
aryl_mono_BrI	[c;$(c[Br,I])]

Functional Group	Rooted SMARTS
urea	[$Nurea]
alcohol	[$Oalcohol]
thiol	[$Sthiol]
alkene	[$Calkene]
alkyne	[$Calkyne]
lactam	[$Clactam]
amide	[$Camide]
thioamide	[$Cthioamide]
anhydride	[$Canhydride]
aniline	[$pseudo_amine;$(N[$aryl]);!$(N~[!#6])]
aniline_unsubstituted	[$pseudo_amine;D1;$(N[$aryl]);!$(N~[!#6])]
azide	[$Nazide]
triazine	[$N1triazine,$N12triazine]
azo	[$Nazo]
thiocarbamate	[$Othiocarbamate,$Nthiocarbamate]
carbamate	[$Ocarbamate,$Ncarbamate]
carbamic_acid	[$Ccarbamic_acid]
carbonate	[$Ocarbonate]
thiourea	[$Nthiourea]
carbonyl	[$Ccarbonyl]
thiocarbonyl	[$Cthiocarbonyl]
carboxylic_acid	[$Ccarboxylic_acid]
acid_halide	[$Cacid_halide]
acid_chloride	[$Cacid_chloride]
thioester	[$Cthioester]
ester	[$Cester]
lactone	[$Clactone]

Functional Group	Rooted SMARTS
aldehyde	[$Caldehyde]
ketone	[$Cketone]
sulfonic_acid	[$Ssulfonic_acid]
sulfonic_ester	[$Ssulfonic_ester]
phosphonic_acid	[$Pphosphonic_acid]
phosphonic_ester	[$Pphosphonic_ester]
phosphoric_acid	[$Pphosphoric_acid]
phosphoric_ester	[$Pphosphoric_ester]
epoxide	[$Cepoxide]
hydrazine	[$Nhydrazine]
hydrazone	[$Nhydrazone]
isocyanate	[$Nisocyanate]
isothiocyanate	[$Nisothiocyanate]
nitrile	[$Cnitrile]
nitro	[$Nnitro]
peroxide	[$Operoxide]
phenol	[$Ophenol]
primary_amine	[$Nprimary_amine]
secondary_amine	[$Nsecondary_amine]
tertiary_amine	[$Ntertiary_amine]
sulfide	[$Ssulfide]
sulfone	[$Ssulfone]
sulfoxide	[$Ssulfoxide]
disulfide	[$Sdisulfide]
alkylating_agent	[$Xalkylating_agent]
alkyl_halide	[$Xalkyl_halide]
aryl_halide	[$Xaryl_halide]
ether	[$Oether]

Functional Group
thioether
acetal
ketal
hemiacetal
hemiketal
sulfonamide
sulfonyl_halide
imino
oxime
dithioacetal
organometallic
oxalyl
enamine
enol_ether
Sdisulfide
Xalkylating_agent
Xalkyl_halide
Xaryl_halide
Sthioether
Oacetal
Cacetal
Oketal
Cketal
OEhemiacetal
OHhemiketal
OEhemiacetal
OHhemiketal
Chemiacetal

Functional Group	Rooted SMARTS
thioether	[$Sthioether]
acetal	[$Cacetal]
ketal	[$Cketal]
hemiacetal	[$Chemiacetal]
hemiketal	[$Chemiketal]
sulfonamide	[$Ssulfonamide]
sulfonyl_halide	[$Ssulfonyl_halide]
imino	[$Cimino]
oxime	[$Coxime]
dithioacetal	[$Cdithioacetal]
organometallic	[$Corganometallic]
oxalyl	[$Coxalyl]
enamine	[$Cenamine]
enol_ether	[$Cenol_ether]
Sdisulfide	[S;$(S([$Calkyl])S[$Calkyl])]
Xalkylating_agent	[$lg_halide;$(*[$Calkyl])]
Xalkyl_halide	[$halide;$(*[$Calkyl])]
Xaryl_halide	[$halide;$(*[$aryl])]
Sthioether	[S;$(S([$Calkyl])[$Calkyl])]
Oacetal	[O;$(O[$Cacetal])]
Cacetal	[C;H1,H2;$(C(O[$Calkyl])O[$Calkyl])]
Oketal	[O;$(O[$Cketal])]
Cketal	[C;H0;$(C(O[$Calkyl])O[$Calkyl])]
OEhemiacetal	[O;$Oether;$(O[$Chemiacetal])]
OHhemiketal	[O;$hydroxyl;$(O[$Chemiketal])]
OEhemiacetal	[O;$Oether;$(O[$Chemiacetal])]
OHhemiketal	[O;$hydroxyl;$(O[$Chemiketal])]
Chemiacetal	[C;H1,H2;$(C(O[$Calkyl])[$hydroxyl])]

Functional Group	
Chemiketal	
Ssulfonamide	
Ssulfonyl_halide	
Nimino	
Cimino	
Ooxime	
Noxime	
Coxime	
Sdithioacetal	
Cdithioacetal	
Corganometallic	
Ooxalyl	
Coxalyl	
Cenamine	
Oenol_ether	
Cenol_ether	
Oether	
Nurea	
Curea	
Oalcohol	
Sthiol	[
Nlactam	[
Clactam	[
Nthioamide	[
Cthioamide	[
Namide	[
Camide	[

Functional Group	Rooted SMARTS
Chemiketal	[C;H0;$(C(O[$Calkyl])[$hydroxyl])]
Ssulfonamide	[S;$(S(=O)(=O)N)]
Ssulfonyl_halide	[S;$(S(=O)(=O)[$halide])]
Nimino	[N;$(N=[$Cimino])]
Cimino	[C;$(C=[N;!$(N~[$hetatm])])]
Ooxime	[O;$(O[$Noxime])]
Noxime	[N;$(N=[$Coxime])]
Coxime	[C;$(C=N[$hydroxyl])]
Sdithioacetal	[S;$(S[$Cdithioacetal])]
Cdithioacetal	[C;$(C1SCCCS1)]
Corganometallic	[C;$(CB),$(C[Mg][$halide]),$(C[Li]),$(C[Cu][Li]),$(C([Ag])#C)]
Ooxalyl	[O;$(O=[$Coxalyl])]
Coxalyl	[$Ccarbonyl;$(C[$Ccarbonyl])]
Cenamine	[C;$(C=C[N;!$Nnitro])]
Oenol_ether	[O;$(OC=[$Cenol_ether])]
Cenol_ether	[C;$(C=C[$Oether])]
Oether	[O;$(O([$Cstd])[$Cstd])]
Nurea	[N;$(N[$Curea])]
Curea	[$Ccarbonyl;$(C(=O)(N)N)]
Oalcohol	[$hydroxyl;$(O[C;!$(C=[!#6])])]
Sthiol	[$mercapto;$(S[#6;!$(C=[!#6])])]
Nlactam	[Namide;R]
Clactam	[Camide;R]
Nthioamide	[N;$(N[$Cthioamide])]
Cthioamide	[$Cthiocarbonyl;$(CN);!$(C(N)(=S)[!#6])]
Namide	[N;$(N[$Camide])]
Camide	[$Ccarbonyl;$(CN);!$(C(N)(=O)[!#6])]

Functional Group	Rooted SMARTS
Canhydride	[$Ccarbonyl;$(CO[$Ccarbonyl])]
Nazide	[N;$(N=[N+]=[N-])]
N1triazine	[N;$(N=N-N);D2]
N12triazine	[N;$(N-N=N);D2,D3]
Nazo	[N;D2;$(N=[N;D2]);!$(N[$hetatm]);!$(N=N[$hetatm])]
Ccarbamic_acid	[$Ccarbamate;$(C[$hydroxyl])]
Othiocarbamate	[O;$(O[$Cthiocarbamate])]
Nthiocarbamate	[N;$(N[$Cthiocarbamate])]
Cthiocarbamate	[$Cthiocarbonyl;$(C(=S)(O)N)]
Ocarbamate	[O;$(O[$Ccarbamate])]
Ncarbamate	[N;$(N[$Ccarbamate])]
Ccarbamate	[$Ccarbonyl;$(C(=O)(O)N)]
Ocarbonate	[O;$(O[$Ccarbonate])]
Ccarbonate	[$Ccarbonyl;$(C(=O)(O)O)]
Nthiourea	[N;$(N[$Cthiourea])]
Cthiourea	[$Cthiocarbonyl;$(C(=S)(N)N)]
Ocarboxylic_acid	[$hydroxyl;$(O[$Ccarboxylic_acid])]
Ccarboxylic_acid	[$Ccarbonyl;$(C[$hydroxyl]);$(C[#6,#1])]
Cacid_chloride	[$Cacid_halide;$(CCl)]
Cacid_halide	[$Ccarbonyl;$(C[$halide]);$(C[#6,#1])]
Clactone	[$Cester;R]
Cthioester	[$Cthiocarbonyl;$(C(=S)O[#6]);$(C[#6,#1])]
Cester	[$Ccarbonyl;$(C(=O)O[#6]);$(C[#6,#1])]
Caldehyde	[$Ccarbonyl;$([H1,H2]);!$(C-[$hetatm])]
Cketone	[$Ccarbonyl;$(C([#6])[#6])]
Ssulfonic_acid	[S;$(S(=O)(=O)[$hydroxyl])]
Ssulfonic_ester	[S;$(S(=O)(=O)O*)]
Pphosphonic_acid	[P;$(P(=O)(=O)[$hydroxyl])]

Functional Group
Pphosphonic_ester
Pphosphoric_acid
Pphosphoric_ester
Oepoxide
Cepoxide
Nhydrazine
hydroxylamine
Nhydrazone
Cisocyanate
Nisocyanate
Cisothiocyanate
Nisothiocyanate
Nnitrile
Cnitrile
Nnitro
Operoxide
Ophenol
Nprimary_amine
Nsecondary_amine
Ntertiary_amine
ring
amine
pseudo_amine
Ssulfide
Ssulfone
Ssulfoxide
Ccarbonyl
Ocarbonyl

Functional Group	Rooted SMARTS
Pphosphonic_ester	[P;$(P(=O)(=O)O*)]
Pphosphoric_acid	[P;$(P(=O)(O)[$hydroxyl])]
Pphosphoric_ester	[P;$(P(=O)(O)O*)]
Oepoxide	[O;$(O([$Cepoxide])[$Cepoxide])]
Cepoxide	[C;$(C1CO1)]
Nhydrazine	[N;$(N-[N;D1]);!$(N=C)]
hydroxylamine	[$pseudo_amine;$(N[$hydroxyl]);!$(N=*)]
Nhydrazone	[N;$(N[N;D2]=C)]
Cisocyanate	[C;$(C=[$Nisocyanate])]
Nisocyanate	[N;$(N(=C=O)*)]
Cisothiocyanate	[C;$(C=[$Nisothiocyanate])]
Nisothiocyanate	[N;$(N(=C=S)*)]
Nnitrile	[N;$(N#[$Cnitrile])]
Cnitrile	[C;$(C#[N;D1])]
Nnitro	[N;+0,+1;$(N(=O)~[O;H0;-0,-1])]
Operoxide	[O;$(O[$hydroxyl])]
Ophenol	[$hydroxyl;$(Oc)]
Nprimary_amine	[$amine;D1]
Nsecondary_amine	[$amine;D2]
Ntertiary_amine	[$amine;D3]
ring	[R]
amine	[N;!$(N*=[!#6]);!$(N~[!#6]);!$(Na);!$(N#C);!$(N=C)]
pseudo_amine	[N;!$(N*=[!#6])]
Ssulfide	[S;D2;$(S([#6])[#6])]
Ssulfone	[S;$(S(=O)(=O)([#6])[#6])]
Ssulfoxide	[S;D3;$(S(=O)([#6])[#6])]
Ccarbonyl	[C;$(C=[$Ocarbonyl])]
Ocarbonyl	[O;D1;$(O=C)]

Functional Group	Rooted SMARTS
Cthiocarbonyl	[C;$(C=[$Scarbonyl])]
Scarbonyl	[S;D1;$(S=C)]
hetatm	[!#6;$([N,O,S,F,Cl,Br,I,P])]
halide	[!#6;$([F,Cl,Br,I])]
lg_halide	[!#6;$([Br,I])]
mercapto	[S;$([H1&-0,H0&-1])]
hydroxyl	[O;$([H1&-0,H0&-1])]
Cstd	[#6;!$(*=[!#6])]
Calkyl	[C;!$(C=[!#6])]
Calkene	[C;$(C=C)]
Calkyne	[C;$(C#C)]
Caryl	[#6;a]
arene	[c]
aryl	[a]

Prolog to Chapter 3

The fundamental power and problem with combinatorial chemistry are the vast numbers of compounds which can be synthesized. By combining a relatively small number of reagents in a similar manner, when using a preoptimized synthetic scheme, a chemist can synthesize thousands of compounds in a week or less. Unfortunately, this kind of "productivity" grows very rapidly, making enumeration, much less structure-based screening, of all the molecules in a large combinatorial library computationally intractable. The fundamental assumption, that for some parts of the calculation, the side-chains can be treated independently, makes studying combinatorial libraries feasible. This assumption has two effects. First, the combinatorial library is abstracted to a scaffold, with one or more "attachment points," and a series of sets of side-chains, one set associated with each "attachment point." Second, the calculations are split into a computationally inexpensive linear phase, where side-chains are considered independently, and a computationally expensive combinatorial phase, where the scaffold and side-chains are considered together as product molecules. The critical strategy is to calculate as many properties as possible in the linear phase and use them to limit the number of compounds which must be considered in the combinatorial phase. Judicious yet purposeful use of this strategy has yielded a very efficient method for screening combinatorial libraries.

In 1996 Yax Sun, with assistance from Todd Ewing and myself, implemented an algorithm specifically for Docking combinatorial libraries (CombiDock v1.0) into a preliminary version of the Dock 4.0 code. This algorithm and its application to the retrospective analysis of a library of (hydroxyethyl)amine compounds against cathepsin D are described in part 1 of this chapter. Despite not allowing translational, rotational, or torsional minimization, Yax's algorithm performed spectacularly in my hands. It was far superior to Dock 4.0 and CombiBuild at selecting the cathepsin D inhibitors from the 1000 compound (hydroxyethyl)amine test library with a variety of scoring functions. This outstanding performance encouraged me to develop CombiDock further.

Part 2 of this chapter describes several modifications to the algorithm which increased its execution speed by more than two orders of magnitude. This faster algorithm allows us to screen an entire database of large combinatorial libraries with CombiDock. This calculation has allowed us to address several interesting issues surrounding scaffold selection, side-chain selection, and the effects of applying filters for secondary properties, such as molecular weight. More importantly, this is the first example of a technique which can take advantage of the accumulated literature on the solid phase synthesis of "drug-like" combinatorial libraries. In the future, perhaps one will be able to search an "Available Libraries Database" rather than the conventional "Available Chemicals Database."

Chapter 3: CombiDock

Part 1: Combinatorial DOCK: Structure-based Combinatorial Docking and Library Design

by

Yax Sun, Todd J. A. Ewing, A. Geoffrey Skillman,

and Irwin D. Kuntz

Published in The Journal of Computer-Aided Molecular Design, November 1998, 12(6):597-604.

Part 2: Combinatorial DOCK Version 2.0: Screening a Database of Combinatorial Libraries

by

A. Geoffrey Skillman, Yax Sun and Irwin D. Kuntz

In preparation for submission to The Journal of Medicinal Chemistry.

Part 1: Combinatorial DOCK: Structure-based Combinatorial Docking and Library Design

by

Yax Sun, Todd J. A. Ewing, A. Geoffrey Skillman,

and Irwin D. Kuntz

Reproduced with permission from
The Journal of Computer-Aided Molecular Design, November 1998,
12(6):597-604

Abstract

We have developed a strategy for efficiently docking a large combinatorial library into a target receptor. For each possible scaffold orientation, all potential fragments are attached to the scaffold, their interactions with the receptor are scored and factorial combinations of fragments are made. To test its effectiveness, it is compared to two simple control algorithms. Our method is more efficient at selecting best scoring molecules and at selecting fragments for the construction of an exhaustive combinatorial libraries. We also carried out a retrospective analysis of the experimental results of a 10X10X10 exhaustive combinatorial library. An enrichment factor of about 4 was found for identifying the compounds in the library that are active at 330 nM.

Introduction

One of the most exciting new developments in medicinal chemistry in recent years is combinatorial chemistry (1). The modular display of functional groups allows a large number of compounds to be considered for synthesis. Coupled with automation technologies and high through-put screening, it offers great potential for the discovery of drug leads. Nonetheless, even though billions of compounds can be proposed, it remains difficult to validate and assay such numbers of compounds. Typically, unless the library is based on oligomeric units, only very small subsets of fragments are selected for actual synthesis, in a process known as combinatorial library design. One of the challenges for computational chemistry is to select optimum sets of functional groups that have the best potential for the discovery of new leads for a given target.

The structure-based drug design method utilizes the information contained in receptor structures by analyzing how well potential lead compounds might bind to the receptors (2). Since the number of protein structures available, computational methodologies, and computer resources are all improving at a rapid pace, it is inevitable that using the information of target structures in drug design will become increasingly important. A recent pioneer study of combining structure-based design and combinatorial chemistry yielded encouraging results (3). In that study, structure-based calculations were done by assuming fixed scaffold orientations and fragments were scored independently for each attachment site. Using fixed scaffold orientations was possible in that case because of experimental evidences and because of limited orientational and conformational freedom for the scaffold. However, to be generally applicable to the combinatorial library design problem, the structure-based design method has to be able to take into account the interdependency of fragments at different binding sites, without prior knowledge of the scaffold orientation. To do that, one must deal with the large number of combinations produced by combinatorial chemistry. If all the combinations are created and examined individually, as in traditional database screening approach, then millions, even billions, of compounds will have to be screened. Such numbers are far beyond present-day computational resources. In this work we report a method that could be used to carry out efficient docking calculations for such large virtual combinatorial libraries.

In the second part of our study, we will use the combinatorial docking method to analysis the experimental results obtained in the previous study (3). In that study, one thousand molecules from a 10X10X10 exhaustive library were synthesized on solid support and assayed individually. It is not often that 1000 compounds are assayed at several concentrations against a single target under the same experimental protocol. These experimental data offer us a unique opportunity to test objectively how well our computational methods actually perform. We think this type of direct comparison over a large number of compounds will have wide implications for the future work in the development of better scoring functions and in the design of experiments.

Computational Method

DOCK. The basic DOCK algorithm has been described in detail elsewhere (4, 5). Four steps are needed to carry out the calculation: 1) the negative-image of the receptor active site is represented by a set of spheres; 2) internal distance matches between a subset of spheres and a subset of ligand atoms are searched; 3) for every match, the ligand is juxtapositioned onto the active site; 4) a score is calculated for the ligand in that orientation. For a single compound in a typical database screen against an enzyme target, up to 10,000 matches could be generated and up to 1,000 of them pass though the check for not bumping with the receptor. These orientations are finally scored, using force field or empirical functions to approximate interaction energies.

Combinatorial DOCK. The combinatorial docking strategy is a simple variation of the basic DOCK algorithm (Figure 1, Figure 2). The site sphere generation is unchanged as step 1. At step 2, only scaffold atoms are used instead of the entire ligand for the generation of matches with the spheres. At step 3 and 4, once a scaffold is matched onto the active site, all possible fragments are attached at all site positions, and scores are calculated for the scaffold and all fragments. As a final step, combinations of fragments are made and the best combinations are then checked for internal clashes and saved if no clashes are found. It should be noted that this kind of fragment superposition algorithm has been tried previously for non-combinatorial problems, such as directed database searching and conformational searching (6).

Although all combinations of fragments are, in theory, examined, the strength of this method is that the combination process is reduced to the simple numerical additions of the fragment scores at all sites. It is thus possible to use simple numerical techniques to speed up the combinatorial process. Specifically, after scoring all fragments at each scaffold orientation, the fragments are sorted according their scores and the combination process can be terminated once it is determined that no combinations better than a user-defined limit can be found. In addition, the internal clash checks, which are computationally expensive, are only necessary for combinations that have good enough scores to be eventually saved.

Test Cases

Part I. Combinatorial Docking and Library Design

We selected for our first test of the algorithm to dock a virtual library of benzodiazepine derivatives (7) to dihydrofolate reductase (DHFR). We chose the benzodiazepine library because of its historic role in combinatorial chemistry as one of the first nonoligomeric combinatorial libraries (Figure 3). 1,4-benzodiazepine derivatives have been shown to have a wide range of bioactivities (8). Partly because we do not have the crystallographic structures for the natural benzodiazepine receptors, we have chosen dihydrofolate reductase as the target for the benzodiazepine library. Since the main purpose of the study is to test the feasibility and efficiency of the combinatorial docking methodology, DHFR is a good target because of its large and deep binding pocket. This binding site provides an excellent test of the inter-dependency among fragments because the resulting "combined" molecules must fit properly into the pocket. This point will be discussed below.

We used the Available Chemical Directory (ACD) from MDL Information Systems, San Leandro, CA and found 308 acid chlorides (R_1), 305 amino acids (R_2) and 404 alkylating agents (R_3) that satisfy the synthetic requirements for building the virtual library at the three attachment sites. The total number of all potential combinations is about 36 million (308X305X404). A newly developed program, Diversify (9), using the Daylight Toolkit (10), was used to prepare the fragments. The leaving atoms on the fragment molecules were removed and tags identifying atoms connecting to the scaffold were added. The Concord program (11) was used to generate three-dimensional structures and the results were saved as mol2 files (12), with the connecting atom information stored in the @<TRIPOS>SET field. Similarly, the scaffold, 1,4-benzodiazepine, was built and the connecting atom information was also identified.

The combinatorial DOCK has been implemented in a new version of DOCK, version 4.0 (5). The only new parameter required is the number of torsional positions to be sampled, uniformly, for the connecting bond between the scaffold and each fragment. We searched six torsional positions in our tests. The regular DOCK force field scoring method was used with one modification. A positive score (penalty) of 0.5 kcal/mol was added for all non-hydrogen atoms of ligands. This modification was made to avoid the largest fragments always having the best scores. It is because that medicinal chemists favor potent yet small compounds as starting leads.

As controls, two other methods were also tested: 1) random selection: fragments were randomly selected from all available candidates; 2) single fragment docking: in this strategy, fragments at different sites were assumed to be independent. Each fragment was attached to the benzodiazepine scaffold by itself and the resulting mono-substituted molecule was docked. The best scoring fragments for each site were then selected as the best candidates for the combinations.

Two steps are needed in docking and designing combinatorial fragment libraries. The first step is to find the best scoring compounds made from all possible combinations of potential fragments. If compounds are to be synthesized individually, no more library design is needed. A completely exhaustive combination approach, i.e. making all possible combinations from selected fragments at each sites, however, is a more efficient experimental design for making equal number of compounds. If exhaustive combination is desired, then fragments at each site have to be selected based on the results from the first step. We will show the results obtained at both steps.

At the step one, i.e., finding the best scoring single molecules, the constraint of using similar amounts of computer time meant that only 20 fragments could be used for each site in the random selection method and single fragment method (still producing 8000 combinations!). It also limited that only one or two conformations per molecule could be used. The conformations were generated by randomly assigning the torsional angle connecting a fragment and the scaffold. To observe the dependency of the searching results on the number of conformations used, calculations were done for both one conformation per molecule and two conformations per molecule. Whenever combinations are made, internal clashes were checked and molecules with internal clashes were removed, typically about 10% of all combinations.

At the step two, i.e., constructing an exhaustive combinatorial library, the following procedures were used for the selection of fragments: 1) combinatorial docking: fragments were ranked and selected according to the frequencies they appeared in the top 1000 scoring combinations. 2) random selection: 10 fragments were selected randomly. 3) single fragment: fragments were ranked and selected according to docking scores of the mono-substituted compounds, i.e., compounds with one fragment attached to the scaffold.

Part II. Retrospective Analysis of the Experimental Results of a Combinatorial Library

Structure-based library design has been used to design fragment libraries for a hydroxyethylamine scaffold (13) targeting cathepsin D, an aspartyl protease. There are three fragment attachment sites on the scaffold. In the previous study, ten fragments were chosen for each site and incorporated in the final combinatorial synthesis (3). The resulting 1000 molecules were assayed for activity at 1 μM, 330 nM and 100 nM, with 67, 23, and 7 compounds having inhibition greater than 50% at each concentration respectively.

Our goal in this work is to analyze in more detail the experimental results for the compounds synthesized and assayed. This is a test for both the searching algorithm and the scoring function. Since only 10 fragments were finally used at each site, fragment conformations can be sampled more extensively than in the initial designing process. A systematic dihedral searching method was used to generate fragment conformations. For torsions with rotational barrier below 2 kcal/mol, according to AMBER force field (14), dihedral angles were sampled every 60 degrees. When a double bond was involved, then only the trans and cis forms were used. The conformational searches generated a total of 282, 152, and 225 molecular conformations for the 10 fragments at each site. We used the same scaffold conformation from the previous work (3), which was determined by matching the scaffold with the crystal structure of pepstatin in the complex with cathepsin D and torsional searching for the three undetermined dihedral angles (3).

Results and Discussion

I. Combinatorial docking of the benzodiazepine library to DHFR:

As mentioned before, the first step is to find the best scoring compounds from combinations of all potential fragments. The distribution of scores for the top 500 scoring molecules found with each method, together with the CPU time used to search for them, are shown in Figure 4. Searching was limited, as described in the method section, for the random selection method and the single fragment method so that each approach was given roughly the same computer CPU time as the combinatorial docking. The average scores of the top 500 scoring compounds are -25.6, -18.1, -15.7, respectively, for combinatorial docking, random selection and single fragment methods. It is interesting that selecting compounds based on one fragment at a time (single fragment method) is even worse than a random selection. The reason for this is that single fragment method assumes independence between fragments and it picked out similar fragments at all three positions that when studied as the mono-substituted scaffold, dock very well into the binding pocket. Once these fragments are put together in the same molecule, however, they interfere with each other. This often results in either inefficient docking in which fragments interact with the target weakly, or worse yet, one fragment bumps into the target and the combination must be discarded.

Having found the best scoring individual compounds, we next considered the design of an exhaustive combinatorial library. The goal is to select a small arbitrary number of fragments from all available fragments for each site to prepare the best library when the combinations are exhaustively made. In our test, we selected 10 fragments for each site yielding a 10X10X10 format for a 1000 compound library.

Using the fragment selection method described previously, three 10X10X10 libraries were constructed based on the results of the combinatorial docking and the two control algorithms. To compare these three libraries of 10X10X10 molecules, 25 random conformations were generated for each combination, again by randomly assigning connecting torsions. It should be mentioned that even with 25 conformations docked for each molecule, the conformational search is still quite limited. Conformations that had internal clashes were discarded. For each molecule, i.e. each combination of fragments, the conformation with the best docking score was saved as the final score for the molecule. The distributions of the scores are shown in Figure 5. As in the first step (Figure 4), combinatorial dock performed best, and random selection is better than single fragment approach. The average scores for the three libraries are -18.9, -11.2, and -6.7. However, there are now much more overlaps between the docking method and the two control algorithms. The primary reason for this is that exhaustive combinations force the inclusion of many not-so-good combinations. We should note, however, that this does not mean that we suggest synthesizing individual best scoring combinations instead of using exhaustive combinatorial approach, for the reason that the current scoring functions are not yet reliable enough to grant such an importance to their scores. In our test, we have attempted to separate the searching algorithm from the scoring function to demonstrate the efficiency of searching for a given scoring function. The reality, however, is that the quality of scoring functions is critical to the quality of predictions. The quality of predictions will in turn influence how the actual experiments should be optimally designed.

The scaffold-based combinatorial docking method presented here is best suited when there is a scaffold with fragments attached to it, even if it is a small scaffold. It also has the limitation that the scaffold can not be too flexible. If there are only a few low-energy conformations available for the scaffold, then these conformations could be treated independently and results from difference conformations can be combined at the end. On the other hand, it would be difficult if the scaffold has too many conformations, unless the conformation or even the orientation of the backbone is restricted or known inside the target, such as in the hydroxyethylamine based library targeting cathepsin D case (3).

II. Retrospective analysis of the experimental results of a combinatorial library

Although non-internal-clashing conformations can be found for all 1000 molecules in the library, in the context of the cathepsin D binding site, at least one non-internal-clashing and non-external-bumping orientation was found for only 752 compounds, based on the conformations we searched. Since some of the unsuccessful molecules showed inhibition, we can only assume the error is in our modeling. The most likely source of this problem is our limited ligand conformational searching and the neglect of receptor flexibility. Our main goal in this part of the study is to test our scoring functions, i.e. how well our calculated ranks relate to the experimental results. So for this analysis, we decided to use only the 752 compounds that we could readily score. We use as a measure of the quality of the calculation the enrichment factor: the initial ratio between the percentage of hits and the percentage of database used. As shown in Figure 6, when the experimental results at 330 μM is used, the enrichment factor is about 4. A completely random ranking would result in an enrichment factor of 1.

Conclusion

We have implemented and tested a combinatorial docking strategy. We have shown that it is able to find better scoring combinatorial molecules than the two control methods. When completely exhaustive combinations are required, fragments selected based on the results from the combinatorial docking also produced better scoring compounds. The combinatorial docking method is fast enough to allow using structure-based library design for general combinatorial chemistry problems when target structures are available. We have also analyzed the experimental results from a previous combinatorial library. An enrichment factor of 4 was obtained using the force field based scoring method.

Acknowledgments

This work was supported by NIH Grants GM39552 (C. Craik, principal investigator) and GM31497 (I.D.K.). We gratefully acknowledge additional support from Daiichi Corp. to Y. Sun and NIH Training Grants to T.J.A. Ewing and A.G. Skillman.

References

1. Thompson, L.A. and Ellman, J.A., Chem. Rev., 96 (1996) 550-600.
2. Kuntz, I.D., Science, 257 (1992) 1078-1082.; Kuntz, I.D., Meng, E.C. and Shoichet, B.K., Accts. Chem. Res. 27 (1994) 117-123.
3. Kick, E.K., Roe, D.C., Skillman, A.G., Liu, G., Ewing, T.J.A., Sun, Y., Kuntz, I.D. and Ellman, J.A., Chem. and Biol., 4 (1997) 297-307.
4. Kuntz, I.D., Blaney, J.M., Oatlet, S.J., Langridge, R. and Ferrin, T.E., J. Mol. Biol., 161 (1982) 269; Shoichet, B.K., Bodian, B.K, and Kuntz, I.D., J. Comput. Chem., 13 (1992) 380; Meng, E.C., Shoichet, B.K. and Kuntz, I.D., J. Comput. Chem., 13 (1992) 505.
5. Ewing, T.J.A. and Kuntz, I.D., J. Comput. Chem., 18 (1997) 1175-1189.
6. Shoichet, B.K et al. Science, 259 (1993) 1445-1450; Lorber, Shoichet, B.K., personal communication.
7. Bunin, B.A. and Ellman, J.A., J. Am. Chem. Soc., 114 (1992) 10997-10998; Bunin, B.A., Plunkett, M.J. and Ellman, J.A., Proc. Natl. Acad. Sci. USA, 91 (1994) 4708-4712; Plunkett, M.J. and Ellman, J.A., J. Am. Chem. Soc., 117 (1995) 3306-3307.
8. Hsu, M.-C., et al. Science 254 (1991) 1799-1802; Chambers, M.S., et al. Biomed. Chem. Lett., 3 (1993) 1919-1924; James, G.L., et al., Science 260 (1993) 1937-1942.
9. Skillman, A,G. and Kuntz, I.D., (see chapter 3).
10. Daylight Chemical Information Systems, Inc., Santa Fe., New Mexico.
11. CONCORD, Robert Pearlman, University of Texas at Austin.
12. SYBYL v6.2, Tripos, Inc., St. Louis, Missouri.

13. Kick, E.K. and Ellman, J.A., J. Med. Chem., 38 (1995) 1427-1430.

14. Cornell, W.D., Cieplak, P., Bayly, C.I., Gould, I.R. and Kollman, P.A., J. Am. Chem. Soc., 117 (1995) 5179-5197.

Figure Caption

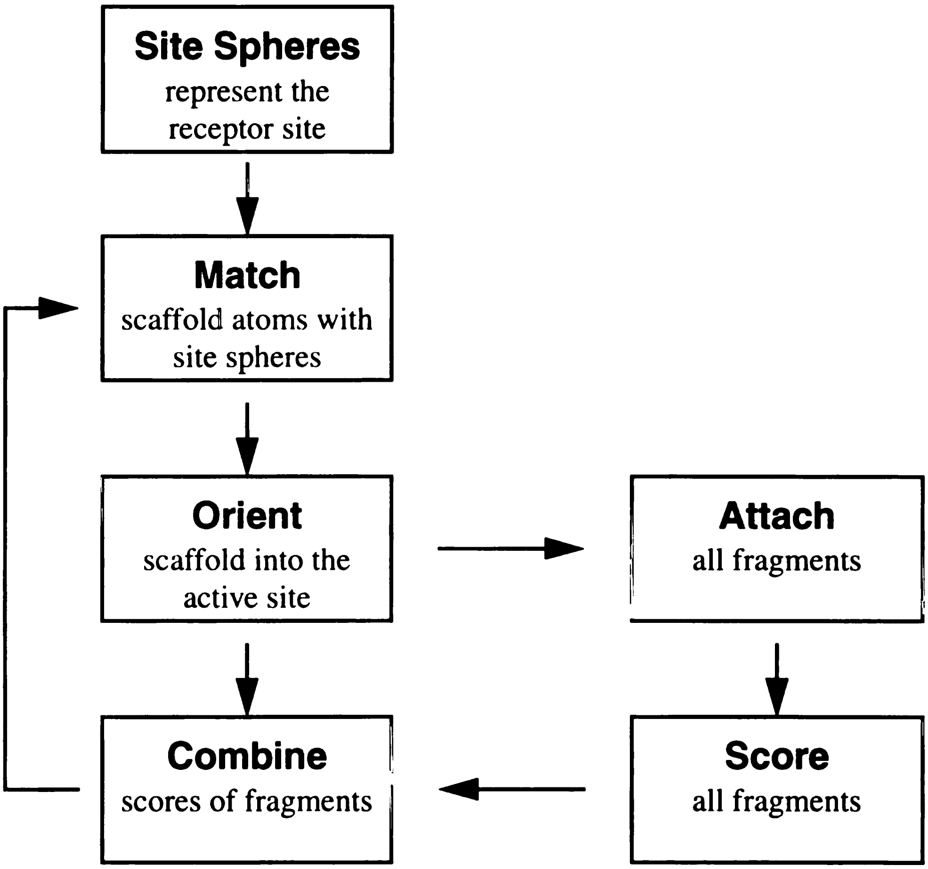

Site Spheres
represent the receptor site
Match
scaffold atoms with site spheres
Orient
scaffold into the active site
Attach
all fragments
Score
all fragments
Combine
scores of fragments

Figure 1: Combinatorial DOCK algorithm.

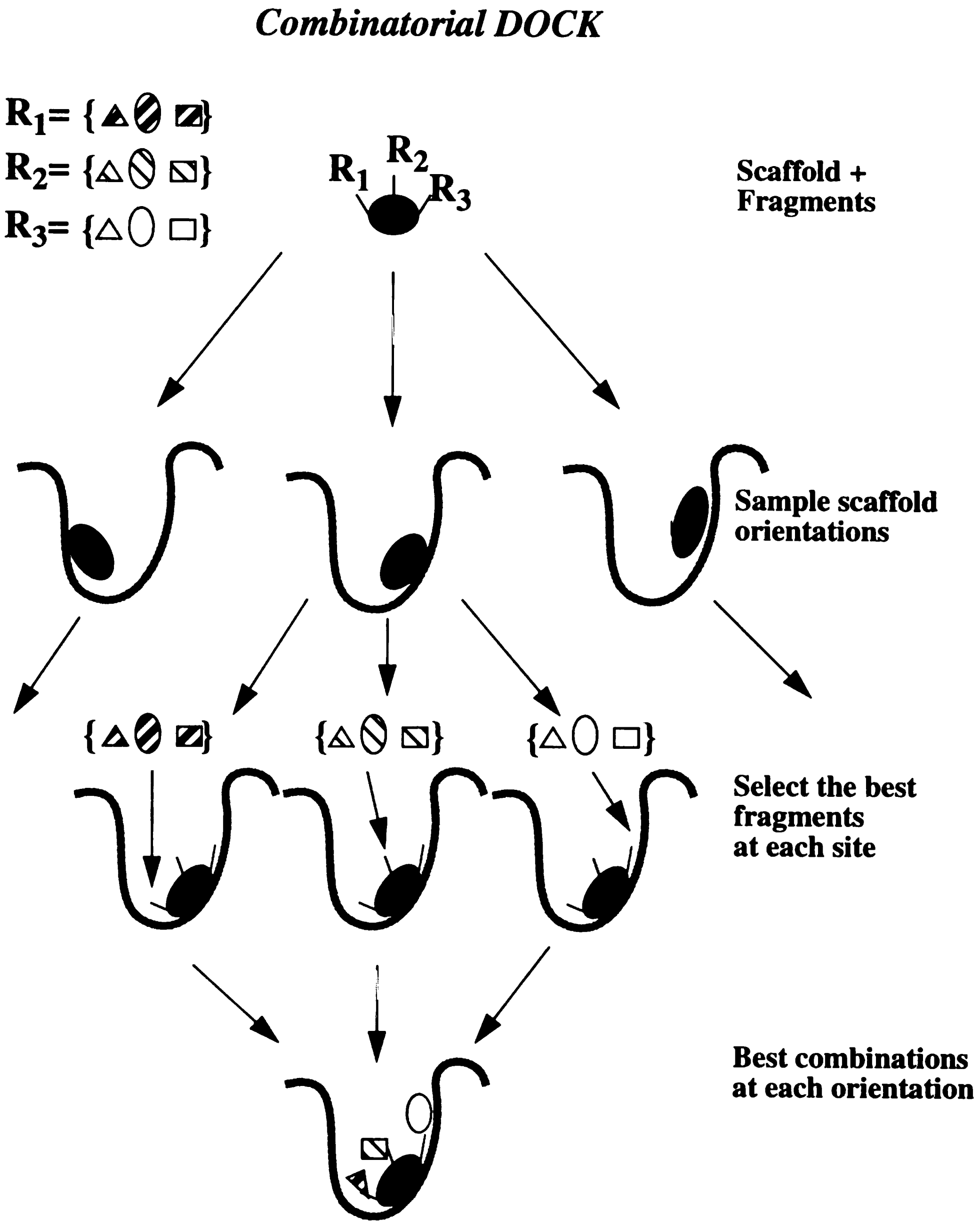

Combinatorial DOCK
R1= {▲ ▨ ▨}
R2= {△ ▨ ▨}
R3= {△ 0 □}
R1 R2 R3
Scaffold + Fragments
Sample scaffold orientations
{▲ ▨ ▨}
{△ ▨ ▨}
{△ 0 □}
Select the best fragments at each site
Best combinations at each orientation

Figure 2: Combinatorial DOCK illustration.

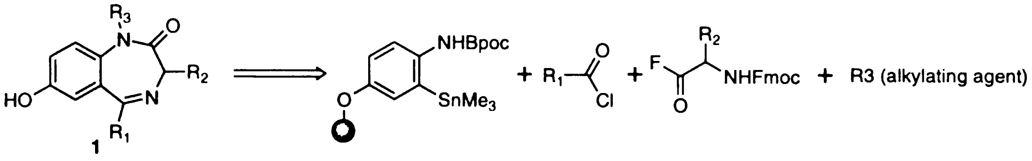


Figure 3: 1,4-Benzodiazopine combinatorial library.

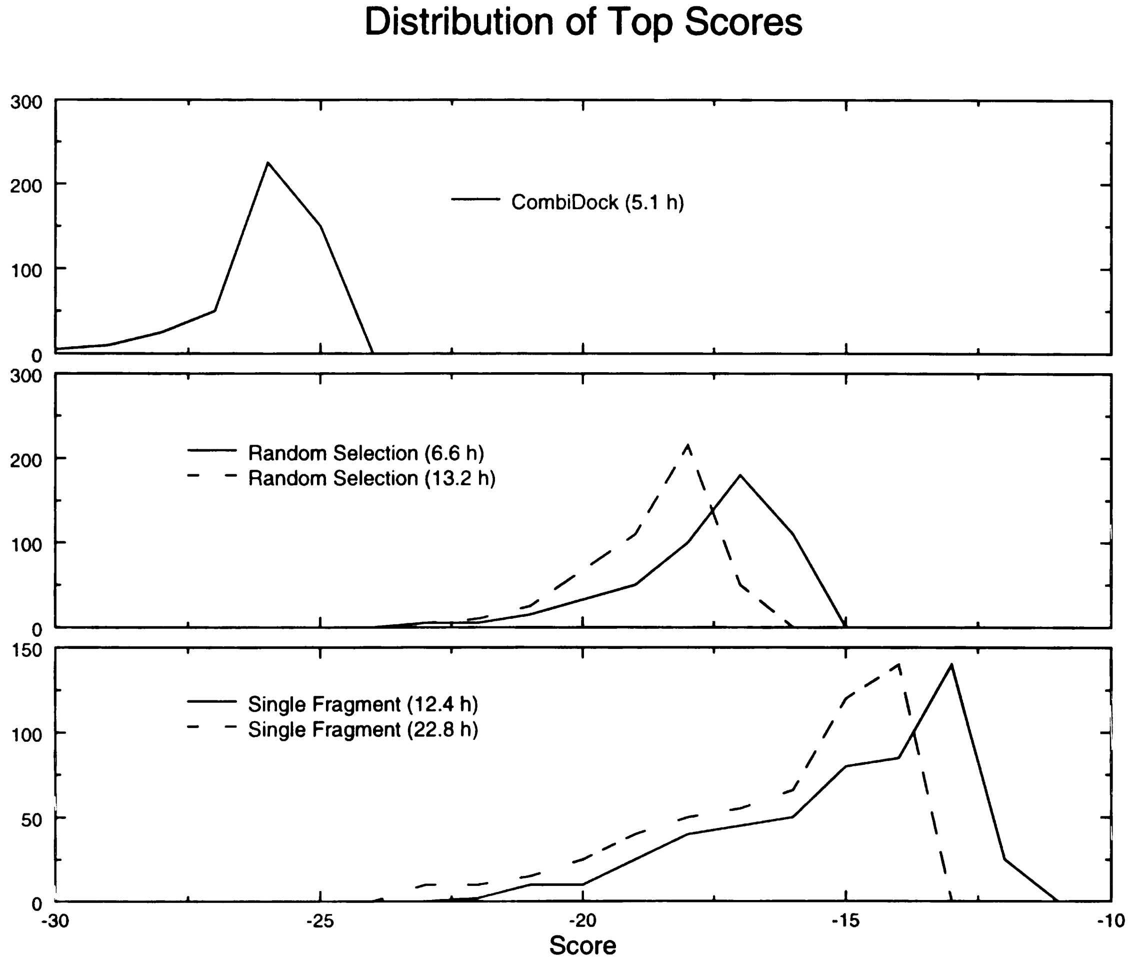


Figure 4: Distributions of the top 500 scoring molecules from three different searching methods. For the random selection method and the single fragment method, only one conformation per combined molecule was generated for the short runs (solid lines), and two conformations per combined molecule were generated for the long runs (dashed lines). All calculations were done on 200 MHZ R4400 Indigo2.

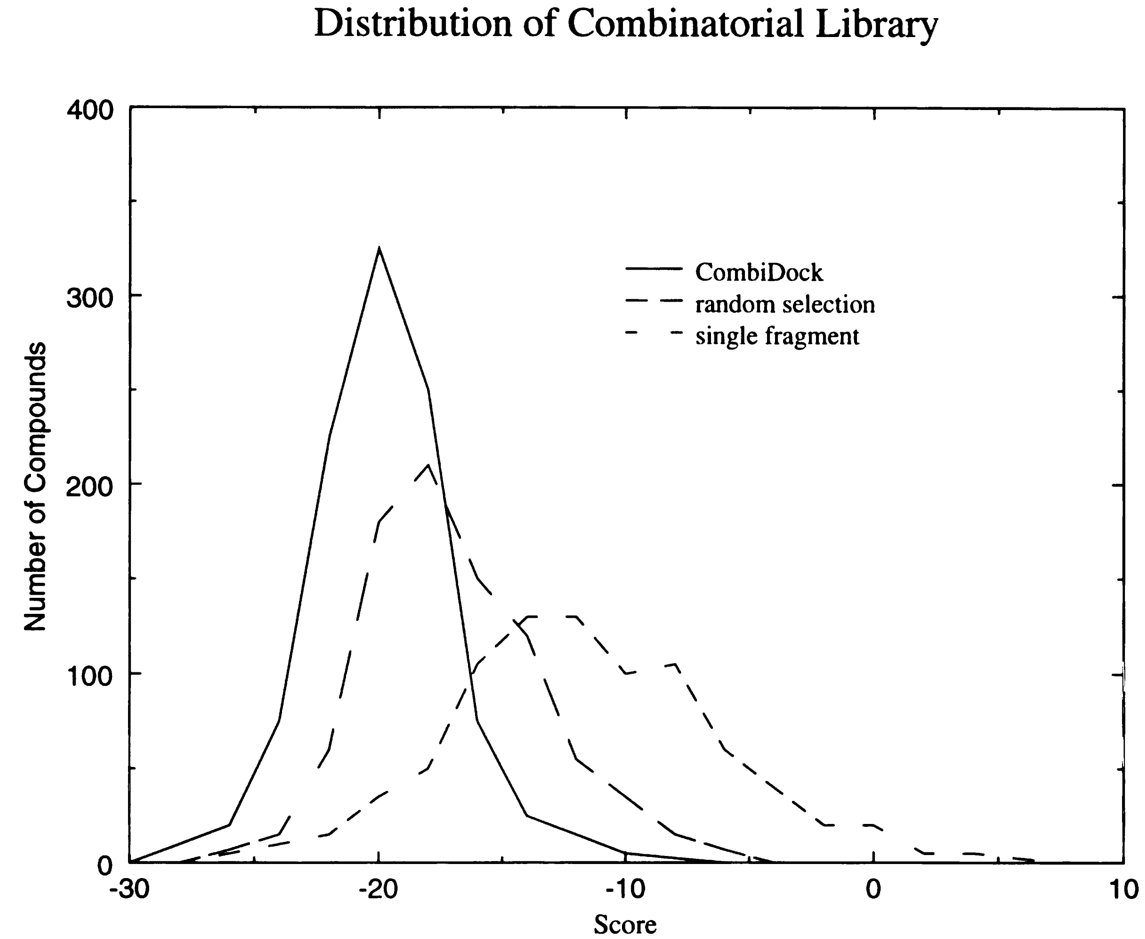


Figure 5: Distributions of the scores of the exhaustive combinatorial libraries. Number of compounds with the score of 10.0 includes compounds with the score of 10.0 or higher.

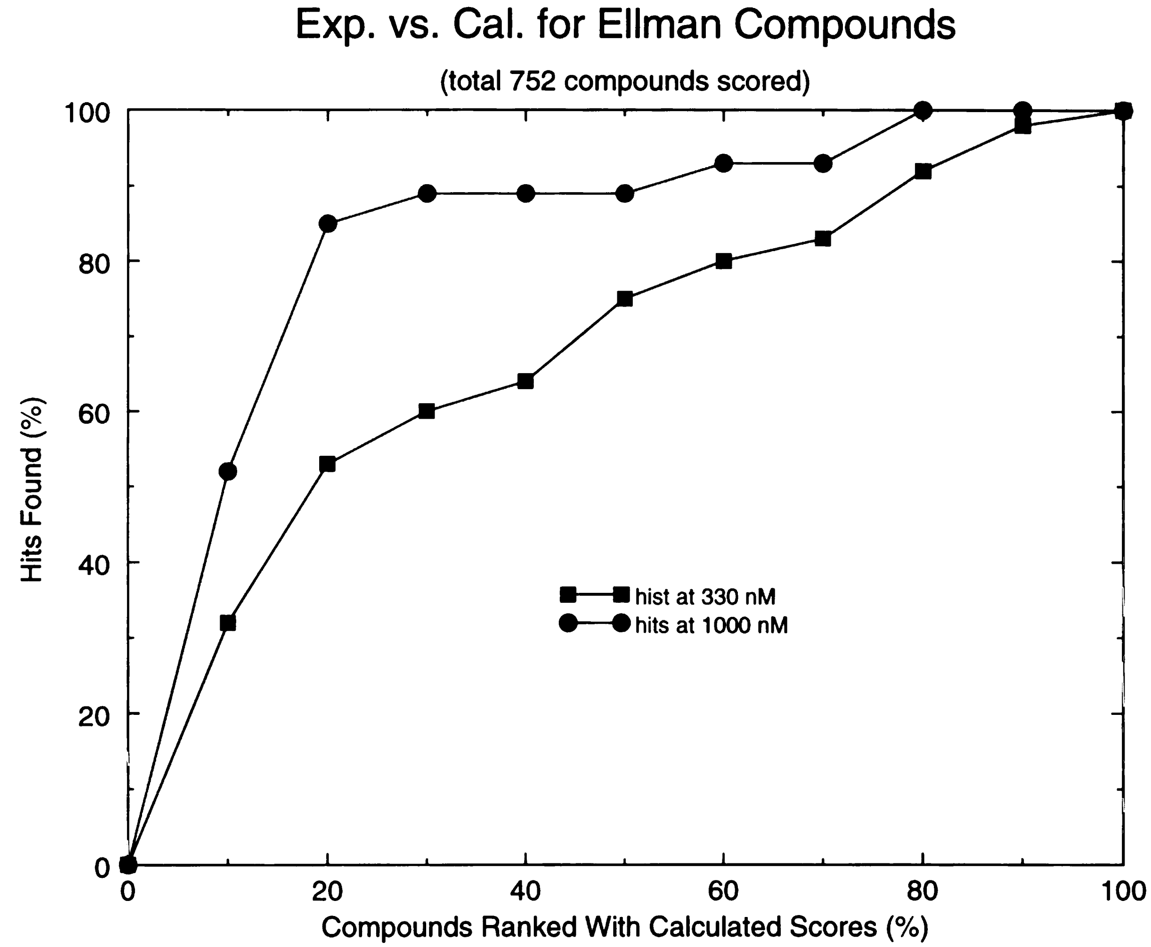


Figure 6: Enrichment Diagram. The relationship between calculated ranks and number of experimental hits for a combinatorial library targeting cathepsin D. The dashed line represents predicted results of a random ranking.

Part 2: Combinatorial DOCK Version 2.0: Screening a Database of Combinatorial Libraries

by

A. Geoffrey Skillman and Irwin D. Kuntz

In preparation for submission to The Journal of Medicinal Chemistry.

Abstract

Structure-based design can be successfully used to screen databases of small molecules in order to identify compounds with specific biological activities. Recently, the integration of combinatorial chemistry and structure-based design has become a powerful tool for identifying potent, non-peptide, small-molecule inhibitors of enzymes(1, 2). Previously we described a method for efficiently docking a single combinatorial library into a target receptor(3). We describe a method for structure-based screening of a database of 45 large (1262 X 1262 X 1262) combinatorial libraries in approximately 96 hours of CPU time on an SGI R10000 workstation. We have modified the algorithm to include two stages of focusing on the most complementary compounds as well as a look-ahead mechanism. This new algorithm is 2-3 orders of magnitude faster than our previous method and allows both significant increases in conformational sampling and comparison of multiple combinatorial libraries. We use the database screening data to demonstrate that screening multiple libraries may be more helpful that screening one very large library. To independently validate the method, we retrospectively analyze the selection of 23 330nM Cathepsin D inhibitors from a 10x10x10 compound combinatorial library. An enrichment factor of greater than 12 (over random) was found with an execution time of less than three minutes.

Introduction

One fundamental goal of chemistry is to design molecules with specific physical and biological properties. For over twenty years, chemists have been using simplified structural models to predict the properties of molecules(4). One of the first applications of structure-based molecular design calculations was the screening of a database of known compounds for those complementary to a crystal structure of the HIV-1 protease enzyme(5). It is now common to screen computationally databases of available compounds or potential (virtual) compounds to identify small molecules which interact with macromolecular targets of known atomic resolution structure. These computations (database mining) generally screen thousands to hundreds of thousands of small molecules(6, 7, 8, 9). The small molecules are modeled as a single low-energy conformation(10, 11, 12), a small collection of low energy conformations(13), or as completely flexible ligands(14, 15, 16). The receptors are usually considered to be rigid in order to reduce computational complexity (limited receptor flexibility now appears feasible in some cases(17)). The six orientational and translational degrees of freedom between ligand and receptor have been explored by a variety of methods. Some methods focus primarily on structural features of the receptor(18), while others make use of potential chemical interactions of the receptor(19, 20, 21). Despite outstanding results when screening databases of available compounds(6, 7, 8, 9) all of these methods are approximate and may fail in the limiting case of predicting whether or not an individual compound will have the desired inhibitory properties before it is synthesized.

Recently, the synthesis of drug molecules has been revolutionized by combinatorial chemistry methods. Combinatorial synthesis principles have long been used in biological systems for the synthesis of biological polymers such as DNA, proteins, and polyketides(22). Bunin and Ellman demonstrated the first extension of combinatorial methods to drug-like small molecules with the synthesis of a library of benzodiazepines(23). This was quickly followed by DeWitt *et. al.*'s work on Diversomers(24). In the ensuing years, combinatorial methods have exploded with the synthesis of a variety of drug-like libraries, including heterocycles, and mechanism-based inhibitors. One recent review estimates that over 250 combinatorial libraries have been published and many more remain proprietary(25, 26). Libraries are now commonly synthesized in both solid and liquid phases and in single compound as well as mixture formats(22). Library synthesis has become part of several different stages of drug-design. Although different nomenclatures have been introduced for these stages, there are three principal strategies for using combinatorial libraries. First, large exploratory or screening libraries are generated and used to identify initial inhibitors. Next, focused libraries can be synthesized to explore structures around an initial hit. Finally, optimization libraries are constructed which carefully sample molecular structures around a clinical lead in order to optimize the physical and pharmacological properties of the series. Regardless of the strategy, at each stage, one must confront the fundamental problem that there are often vastly more members of a library than can practically be synthesized.

A fundamental step forward occurred when Kick, Roe and co-workers combined structure-based design with combinatorial synthesis methods to generate two 1000 compound exploratory libraries of the mechanism-based (hydroxyethyl)amine inhibitors of the human aspartyl protease cathepsin D(22). They demonstrated that libraries designed by structure-based methods produced a larger number of more-potent inhibitors than diverse (or random) libraries of equal size. In a follow-up study, Haque, Skillman and co-workers showed that not only could structure-based design and combinatorial methods be integrated to identify potent inhibitors, but that pharmacokinetic properties could be optimized as well (see chapter 8)(2). In that design of (hydroxyethyl)amine inhibitors of the essential malarial aspartyl protease Plasmepsin II, Haque *et. al.* used a more detailed conformational search and libraries which focused on one side-chain at a time to identify single digit nanomolar inhibitors which also fulfil Lipinski's "Rule of Five," a structural surrogate for the ability to cross membranes(27). Indeed, when rule of five compatible Cathepsin D inhibitors were tested in a cell-culture tau protein processing assay, they inhibited when equally potent non-rule of five compatible cathepsin D inhibitors were unsuccessful(28).

The integration of structure-based design and combinatorial chemistry has been extremely successful because the strengths and weaknesses of these two powerful tools complement on another. Combinatorial synthesis methods allow a single chemist to synthesize a remarkable number of compounds; however, the potential number of compounds to be synthesized is an even greater number. Structure-based design methods allow rapid assessment of the approximate binding constant of many potential compounds in a statistically significant population of compounds; however, in any individual case, structure-based design, as implemented for rapid screening, has the potential to fail. When combinatorial chemistry and structure-based design are integrated, the computational methods offer the combinatorial chemist a means to assess an entire library of virtual molecules and focus on a subset of those compounds enriched with the most interesting compounds. Conversely, the combinatorial synthesis of tens, hundreds, or even thousands of compounds allow a computational chemist to make predictions on a statistically significant number of compounds. The complementary integration of combinatorial chemistry and structure-based design is a successful tool for drug discovery and optimization.

There have been several other examples of the integration of structure-based design and combinatorial methods to help determine which set of compounds will be synthesized from a larger virtual optimization library(29, 30, 31). However, most studies generally focus on one site of chemical variation at a time. Although there are many programs which have been adapted and used to design optimization libraries with a single variable position or a fixed scaffold orientation, there are few which address simultaneous exploration of multiple variable points. Although the former have an important role in drug development, they effectively avoid the combinatorial problem by focussing on only one point of variation, thus limiting the scope of problems for which they are applicable. The method of Roe was the first to examine multiple points of variation simultaneously(1). It used a fixed orientation of the scaffold in the active site and a probabilistic method to assess interactions between side-chains at different attachment points. In a second paper(3), we extended this idea to allow systematic exploration of hundreds or thousands of scaffold orientations in the active site. Further, by ranking the compounds by their optimum potential complementarity to the macromolecular binding site, we were able to overcome the combinatorial problem and examine the individual interactions among the side-chains of each of the best molecules. In this paper, we describe improvements that allow significant speed enhancement so that it becomes feasible to compare large numbers of libraries, each containing ca. 10^6-10^{10} discrete compounds. We improve the previous algorithm with two "greedy" focusing steps and a look ahead procedure for scaffold placement to greatly optimize the library docking process. We validate the algorithm's continued performance in retrospective analysis of mechanism-based cathepsin D inhibitors. Finally, we demonstrate that this new algorithm can be used to screen a database of 45 large combinatorial libraries against the macromolecular target, human dihydrofolate reductase.

Methods

Combinatorial DOCK version 1.

The combiDOCK version 2.0 algorithm is an extension of our previously described program (figure 1)(3). Both programs are based on the original DOCK procedure(11, 12, 32). Briefly, in pre-calculations, many conformations of all of the side-chains are generated and a negative image of the binding site of the target macromolecule is constructed from overlapping spheres(18). Scaffold orientations are generated using a clique finding algorithm which matches sets of ligand atoms with sets of receptor site spheres(32). For each scaffold orientation, all conformations of the side-chains are oriented in the same reference frame as the scaffold and scored according to their complementarity to the receptor. Scores for all complete molecules are generated and ranked using simple arithmetic operations. The best potential molecules are screened for intramolecular clashes between side-chains, and those without internal clashes are saved.

Combinatorial DOCK version 2.

Overview (figure 2). The combinatorial libraries are defined by a chemically unique scaffold with pre-determined sites of side-chain attachment. Lists of chemically appropriate side-chains are maintained for each attachment site. Multiple orientations of the scaffolds are generated and minimized in the active site using a small set of "probe" side-chains. For the highest scoring scaffold orientations, all of the conformations of all of the side-chains are attached to the scaffold and scored for complementarity to the target macromolecule. The side-chains are then ranked, and a list of the best scoring potential molecules is generated by simple arithmetic summation of the score of the individual fragment scores. These best-scoring potential molecules are screened for compliance with physical property profiles as well as for intramolecular clashes. Compounds which pass all of the filters are saved on a list of the best scoring *individual* molecules.

Receptor Preparation. The target is an atomic resolution structure of a macromolecules, generated by crystallography, NMR, or modeling(9). Crystallographic waters, ligands, and often ions and cofactors are removed. A negative image of the binding site made from overlapping spheres is created using the program SPHGEN(18). These spheres are used to direct the scaffold and probe side-chain fragments into the active site. Thus the spheres may be chosen to describe the entire site, or focus on a region of particular interest. Additional information can easily be included by supplementing the sphere set with atom centers from the crystallographic inhibitors, substrates, crystallographic waters, or cofactors which have been removed(8).

Ligand Preparation. Each combinatorial library must be abstracted to a scaffold with multiple attachment points, each associated with a set of available side-chains. Multiple conformations of the scaffolds and side-chains are pre-generated using any conformational sampling program(33, 34, 35). For the work here, we have used Dock 4.0 to pregenerate conformations(14). Multiple conformations of the "probe" side-chains, used in the look ahead procedure (*vida infra*) are also generated. As the scaffold and side-chains are read into CombiDOCK, the molecular weight, number of hydrogen-bond donor, and number of hydrogen-bond acceptors are calculated and stored.

Docking Calculation. The docking procedure has ten steps, broken into two phases: the look-

fold "look-aheac

molecules are bu

the scaffold: 2) a

atoms to negative

ecule is scored ar

or only the van de

molecule are rank

scoring scaffold o

chain phase (figur

at run-time. **Side-**

chain conformatio

tions of that side-

tions (where S = n

the best scoring m

culate the total po

number of hydrog

property filters (vi

side-chains (figur

best molecules (fi

and steps 1-10 are

optimization phas

score filter in the 1

phases: the look-ahead scaffold docking phase and the side-chain phase (figure 2, 3). **Scaffold "look-ahead" Phase:** 1) a small set (usually 100's - 1000's) of "look-ahead" probe molecules are built by attaching all combinations of the probe side-chain conformations to the scaffold; 2) all of these molecules are oriented into the active site by matching ligand atoms to negative-image site points (spheres) (figure 3B); 3) each orientation of each molecule is scored and minimized based on either the full AMBER intermolecular potential, or only the van der Waals portion of the potential (figure 3C); 4) all orientations of each molecule are ranked according to their complementarity to the active site and the N best scoring scaffold orientations (without the probe side-chains) are passed on to the side-chain phase (figure 3D). Here N is the scaffold greedy parameter which is set by the user at run-time. **Side-Chain Phase:** 5) for each scaffold orientation, attach and score all side-chain conformations; 6) for each side-chain, rank and select the M best scoring conformations of that side-chain; 7) for each attachment point, rank all M*S side-chain conformations (where S = number of side-chains and M is defined in step 6) (figure 3E, 3F); 8) for the best scoring molecules, use simple arithmetic operations on the fragment values to calculate the total potential score, molecular weight, number of hydrogen-bond donors, and number of hydrogen bond acceptors; 9) if the compound passes all scoring and physical property filters (*vida infra*) (figure 4A, 4B), check for intramolecular clashes between the side-chains (figure 3G); 10) save the best scoring molecules without clashes on a list of the best molecules (figure 3H). Steps 5-10 are repeated for each of the N scaffold orientations, and steps 1-10 are repeated for each combinatorial library. By using a look ahead scaffold optimization phase, two greedy "focusing" steps and three physical property filters and a score filter in the linear phase of the calculation, the cost of the final combinatorial clash-checking phase is minimized. This method allows extremely efficient structure-based screening of multiple combinatorial libraries.

Library Database Generation. To generate a database of combinatorial libraries for software demonstration purposes, we first searched the available chemicals directory (ACD)(36) for potential scaffold fragments. We searched for substituted ring structures rich in the functional groups which can be used as chemical handles for synthesizing derivatives (primary or secondary amines, acids, primary or secondary alcohols, ketones, aldehydes etc.). 228 structures were identified and fifty representative compounds were choose by clustered using Daylight's fingerprints(37) and a complete linkage hierarchical clustering algorithm(38). Two or three atoms for each scaffold were chose as attachment points for side-chains based loosely on plausible chemistry. Two sets of test-case side-chains were selected, a large set of 1262, and a smaller set of 100 (for comparison purposes). The larger set of side-chains, 1262 acylating agents, was selected from the ACD (with their respective acylating group removed), while the smaller set of side-chains, 100 peptide-like fragments, were generated by hand. This database represents a virtual library of 36 billion compounds based on 45 different structural scaffolds. We emphasize that neither the scaffold library nor the side-chain libraries were selected to guide a drug design application. Nevertheless, they are sufficient for the test cases demonstrated here (*e.g.* figure 6). The libraries in this database are similar to many heterocyclic libraries which have appeared in the literature(22, 39) and contain compounds with appropriate physical properties (molecular weight, hydrogen-bond donors, hydrogen-bond acceptors, ClogPs). These methods could be easily be extended to a database of libraries reflecting actual combinatorial synthesis.

Test Cases

Part I. Retrospective Analysis of the Experimental Results of a Combinatorial Library. Kick, Roe, and co-workers previously reported the structure-based design of a 1000 compound library of mechanism based cathepsin D inhibitors (figure 5)(1). The entire 10x10x10 library of (hydroxyethyl)amine inhibitors was assayed at 1μM concentration, and compounds with more than 60% inhibition were assayed at successively lower concentrations as well. Sixty-seven, 23, and 7 inhibitors were identified with IC_{50}'s better than 1mM, 330nM, and 100nM respectively. Here we used CombiDock to rank the 1000 compounds and examine where the 23 inhibitors with IC_{50} less than 330 nM fall among the 977 non-inhibitors. We used Dock 4.0 to generated 1284, 704, and 1070 molecular conformations for the 10 fragments at sites R_1, R_2, and R_3 respectively. A single scaffold conformation generated for design of the original library was used(1). For this example, the probe-side chain option was turned off (with this very small library, its overhead was too expensive) and all 251 scaffold orientations were passed on to the side-chain phase, and the top-scoring 15 conformation of each side-chain were used to construct the final molecules.

Part II. Screening a database of Combinatorial libraries. The second test case was molecular docking of a database of combinatorial libraries against dihydrofolate reductase (accession number 4dfr). Dihydrofolate reductase was chosen because it is a well understood enzyme, which binds both the inhibitor methotraxate and the cofactor NADH, and because we, and other groups, have experience using it as a test case. CombiDock was used to screen a database of 45 large combinatorial libraries, 18 with 3 attachment point, and 27 with two attachment points. The larger database was constructed with 1262 side-chains modeled by 59,864 conformations at each attachment point, (vida supra) while the smaller database was constructed with 100 side-chains modeled by 5198 conformations at each attachment point. For the larger database, 3.62×10^{10} three attachment point molecules with 3.86×10^{15} conformations in 18 libraries and 4.3×10^{7} two attachment point molecules with 9.68×10^{10} conformations in 27 libraries were screened, while for the smaller database, 1.8×10^{7} three attachment point molecules with 2.53×10^{12} conformations in 18 libraries and 2.7×10^{5} two attachment point molecules with 7.3×10^{8} conformations in 27 libraries were screened. Two probe side-chains, phenyl and ethyl, were used in the look-ahead scaffold docking phase. For each library, fifteen scaffold orientations were passed from the look-ahead phase to the side-chain phase, and the top-scoring fifteen conformations of each side-chain were considered when constructing the best-scoring complete molecules. Each database of libraries was docked twice, once with no physical property limit, and once with a portion of Lipinski's "Rule of Five" (molecular weight<500, hydrogen-bond donors<5, and hydrogen-bond acceptors<10) in place(27). The ClogP portion of the Lipinski filter was not implemented because logP is not an additive property for the fragments used in this study. Since molecular weight and ClogP are often correlated molecular properties(27), the partial Lipinski filter implemented here remains useful. The 100 best scoring molecules from each scaffold were saved for a total of 4500 molecules from each docking.

Results

As in our previous work, CombiDock version 2.0 successfully identifies the known Cathepsin D inhibitors. When the 1000 compounds were ranked by CombiDock 2.0, nearly 80 percent of the 23 inhibitors are in the top 100 compounds (figure 7A). When normalized by the number of inhibitors identified by a random search, the enrichment ratio (CombiDock inhibitors/random inhibitors) extends to above 12 (figure 7B), a significant improvement over our previous work. The more dramatic difference with the new algorithm is the time the calculation requires. CombiDock version 2.0 completes the enrichment calculation in only 160 seconds on an SGI R10000 workstation, over 2.5 orders of magnitude faster than our previous version.

The dramatic increase in the speed of execution for CombiDock 2.0 allowed us to screen a database of 45 large combinatorial libraries. The 45 library scaffolds were screened four times against dihydrofolate reductase (4dfr). Two sets of side-chains, one small (N=100) and one large (N=1262) were each screened twice, once with no physical property filter, and once with a Lipinski filter(27) in place. The larger databases took an average of approximately 96 hours of CPU time on an SGI R10000 workstation, broken into about 3.5 hours per three attachment-point library, and about 1.3 hours per two attachment-point library. Execution times for the small databases were much shorter. The best score of any molecule and the ranking of the library for all four runs can be seen in Table 1. In each case, the best scaffolds display molecules with van der Waals scores ranging from -50 to -73. These scores compare very favorably with the minimized van der Waals score of -39 for the 2.5 nM crystallographic inhibitor methotrexate (40, 41). Some example scaffolds along with their maximum scores can be seen in figure 6. These small heterocyclic scaffolds are merely hypothetical library test-cases; none-the-less, they demonstrate the ability of CombiDock to carry out structure-based screening of an entire database of a variety of combinatorial scaffolds and libraries.

Analysis of the screening of a database of large combinatorial libraries has revealed several general principles. We first examine the range and ranking of scores of the libraries for each of the four runs (figure 11,12). In general, scaffolds with three side-chains scored better than those with two side-chains, and database runs constrained by the Lipinski filter did not score as well as database runs with no constraints. Finally, databases of very large combinatorial libraries (N=1262) scored better than the same scaffold database with smaller libraries (N=100). Special attention should be paid to the large library database run with the Lipinski filter, since this most closely resembles realistic a drug-design application. Despite the generalizations noted above, in this particularly important case (N=1262, Lipinski filter), scaffolds with only two side-chains are ranked favorably compared with those containing three side-chains, despite the fact that they represent a dramatically smaller number of the total compounds screened (figure 12).

Next, we examine the physical properties of the top 4500 compounds from each database run, and the effect of the Lipinski filter on these properties. When the Lipinski filter was turned on, compounds with molecular weight >500 amu, hydrogen-bind donors > 5, or hydrogen-bond acceptors >10 were discarded on-the-fly during library screening. The most dramatic effects of the filter can be seen in the molecular weight distributions where without the filter, the best compounds are distributed in a flat bell curve from 300 amu to 900 amu, while with the filter, most of the best scoring compounds are just below the 500 amu cutoff (figure 8). The library with more side-chains has an average molecular weight of 661 before the filter and 491 after, while the library with fewer side-chains has an average mole

filter has a large

expense of good

The effe

distributions (fig

pounds contain m

significant role in

bonds, such as th

the Lipinski filter

1262 side-chains

scoring molecule

a dramatic increa

molecular weight

+/- 2.4 to 5.7 +/-

the 1262 side-cha

strained compoun

the Lipinski filter

ity of identifying c

membranes far out

used in evaluating

The final qu

is whether side-cha

compounds. To gle

an average molecular weight of 559 before the filter and 472 after. Although the Lipinski filter has a large effect on the compounds selected, this limitation does not come at the expense of good scores.

The effects of the Lipinski filter are not very dramatic for hydrogen-bond donor distributions (figure 9). Even without the Lipinski filter, very few of the best scoring compounds contain more than five donors (1.1 percent). The HBD filter may play a much more significant role if the screening were to use a scoring function which favors hydrogen-bonds, such as the empirical scoring function of Bohm(20). There is a significant effect of the Lipinski filter on the much more common HBAs, particularly among the libraries with 1262 side-chains (figure 10). In this case, without the filter, more than 500 of the 4500 best scoring molecules contain more than 10 HBAs. Interestingly, with the filter on, there is not a dramatic increase in the number of compounds near the limit (as was seen with the molecular weight distributions), rather, the mean of the distribution was lowered from 7.6 +/- 2.4 to 5.7 +/- 1.8. Overall, only 3 percent of the best unconstrained compounds from the 1262 side-chain database would survive the filter, while 30 percent of the best unconstrained compounds from the 100 side-chain database would survive the filter. Although the Lipinski filter constraint does have some impact on the scores of compounds, the utility of identifying compounds with a higher probability of being able to cross biological membranes far outweigh this effect, particularly in light of the known approximations used in evaluating compound affinities(11).

The final question we attempted to address using the screening data generated here is whether side-chain or scaffold selection is more prominent in determining the score of compounds. To gleen answers to this question from the data, we compared the ranking of the scaffolds with few (N=100) versus many (N=1262) side-chains, and the ranking of the side-chains with one scaffold versus other scaffolds. We also examined the interplay of the Lipinski filter with these effectors. The correlations of the ranks of the scaffold among all four runs can be seen in figure 13. The results are complicated, an probably indicate (not unexpectedly), that it is a balance of good scaffold selection and good side-chain selection that generates good scores. Without the Lipinski filter, there is an excellent correlation (CC=0.9) between the scaffold ranking with the artificially small set of side-chains and the larger, more realistic set of side-chains, initially indicating that the scaffold selection may be a dominate force. This hints at the potential that scaffolds may be compared to one another using a relatively small number of side-chains at each position. Unfortunately, the picture becomes more interesting when we examine the large libraries with the Lipinski filter on, in which case the scaffold ranking does not correlate with any of the previous cases. We propose that with the larger number of side-chains with which to optimize binding, the Lipinski filter can have a more reasonable effect on the optimal scaffold. By contrast, with only 100 side-chains, the scores may reflect a less specific binding where effects of the Lipinski filter on suitable side-chains have less of an impact on scaffold ranking (*e.g.* since no particular side-chain scaffold combination is outstanding, it is easier to find a replacement combination with approximately the same score). This hypothesis would explain both the lack of correlation between the scaffold ranking of the large (N=1262) library between Lipinski on and off, as well as the much greater reduction in scores caused by the Lipinski filter in the larger library database versus the small library database.

To examine the importance of side-chains, we again focus on the most realistic case (N=1262, Lipinski filter on). When the 100 best compounds from each library in the database are considered, the libraries display between 19 and 92 different side-chains. Among the top scoring 4500 compounds (1800 with three side-chains, and 2700 with two side-chains) 641 unique side chains were used out of a maximum possible 1262 (51 percent). Each of these side-chains appears in an average of 4.5 scaffold lists. The best side-chain score is -32.7, with an average score of -11.7 for all of the side-chains considered here. The 4-guanidino-butyric acid side-chain appears on the most scaffolds (N=23), while 198 side-chains appear with only one scaffold. It appears that the more common side-chains are those that are highly flexible or small, whereas the rare side-chains have fewer rotatable bonds and often include one or more cyclic structures. The best scoring side-chains do not necessarily appear with multiple scaffolds, in fact, of the six best scoring side-chains, 4 occur with only 1 scaffold, while the other 2 occur with only 2 scaffolds, perhaps indicating some specificity that is dependent on both the scaffold and the side-chain.

Discussion

This is the first published example of the application of structure-based design to screening a database of large combinatorial libraries. This example represents the logical synthesis and extension of ideas from previous methods to 1) computationally screen a database of individual compounds for those which might bind to a macromolecular target, and to 2) computationally screen an individual combinatorial library for those members which might bind to a macromolecular target. Comparison of an entire database of combinatorial libraries allows insights into the best scaffold for a target, the best side-chains for a target, and the best combinations of scaffolds and side-chains for a target (*e.g.* whole individual molecules). These insights could be used to determine the size, number, and content of exploratory combinatorial libraries synthesized to discover inhibitors of individual macromolecular targets or entire classes of macromolecular targets. In addition to these specific questions, which could be assessed for each system which the method was applied to, molecular docking of a database of combinatorial libraries has allowed us to address three questions which are fundamental to the design of combinatorial libraries. These are: the effect of potential library size (three attachment sites versus two attachment sites) on value of the library to databases screening; the effect of placing additional molecular property constraints on the ligand screening process; and, the relative importance of identifying an appropriate scaffold versus appropriate side-chains. Each of these issues is discussed below.

A superficial analysis of the quality of a combinatorial library for screening can place too much emphasis on the number of potential compounds in the library. Because of the nature of combinatorics, the size of a potential combinatorial library is dominated by the number of attachment points to the scaffold.

$$\text{Library Size} = \prod_{n\,=\,1}^{n\,=\,\text{Number of Attachment Points}} (\text{Number of Side-Chains})_n$$

For example, the total number of compounds screened in this database of combinatorial libraries would be dwarfed by the number of compounds in a single four-component library which incorporated 1000's of side-chains at each site. Despite this, the data presented here indicates that screening additional libraries, even though they are insignificant by strict numerical comparison, may produce significant contributions to the best molecules. This is reflected by the data that, particularly with the Lipinski filter in place, some of the libraries with only two attachment points are quite highly ranked, despite the fact that they contain three orders of magnitude fewer compounds than the three attachment point libraries which are ranked below them. One potential implication of this result is that for ligand screening (whether experimental or computational), it may be more useful to balance screening very large libraries with screening large number of moderate size libraries. The balance must also consider that development costs and reagent costs can be much larger for multiple small libraries than for a single large library. This consideration does not impact computational screening to the same degree it does experimental HTS. When screening combinatorial libraries, it is more useful to screen many large libraries rather than a single extremely large library. It may be possible that a threshold exists for the optimal number of compounds to be screened from a single library scaffold at least when screening an exploratory library (or libraries) for modestly potent inhibitors.

A related issue, whose importance is magnified in light of the discussion of library size, is the relative importance of side-chain selection versus scaffold selection. It is evident from the important libraries with only two side-chains that correct scaffold selection is essential (*vida supra*). However, it appears that the data here support the idea that a good scaffold is essential but not sufficient, and must be accompanied by appropriate side-chains. This is not unexpected considering that, for molecules in most libraries, the side-chains contain two-thirds or more of the atoms. Particularly when additional constraints are placed on the problem (Lipinski filter), each side-chain makes important and specific contributions t

Develop
potency. Additi
branes), low-to
workers have d
membranes(27
potency(2, 28).
essential interac
with a few notab
biomembranes a
ligand screening
design phase ra
attempting to r
the advantages
reagents(42).
chapter 2), co
Indeed, we ar
with the sear
ing molecule
it allows us
combinatori
structure su
eager to in

contributions to the overall molecular properties.

Development of potential clinical lead molecules requires more than simple high potency. Additional necessary properties include *in vivo* potency (ability to cross biomembranes), low-toxicity, favorable metabolism, and high oral bioavailability. Lipinski and co-workers have developed a simple molecular structure surrogate for the ability to cross membranes(27) which in our experience aids the development of molecules with *in vivo* potency(2, 28). There is a natural conflict in the drug design process because, assuming essential interactions are satisfied, larger molecules tend to be more potent(27). However, with a few notable exceptions, very large molecules (> ca 500 amu) do not readily cross biomembranes and are not orally bioavailable. By incorporating the Lipinski filter in our ligand screening process, we benefit from integrating this downstream goal early in the design phase rather than first driving optimization toward high potency ligands, then later attempting to recover secondary properties. Gillett and co-workers have recently shown the advantages of evaluating the molecular properties of product compounds rather than reagents(42). Although it is easy to filter combinatorial libraries at the reagent level (see chapter 2), combinatorial explosion makes filtering of products much more difficult. Indeed, we are only able to filter product molecular properties by integrating the filters with the search for potent ligands. In the same manner that focusing only on the best scoring molecules allows us to avoid the combinatorial explosion in side-chain clash checking, it allows us to examine product molecular properties without being overwhelmed by the combinatorial explosion. Significant work is underway to develop additional molecular-structure surrogates for other medicinal properties(43, 44) and in the future we will be eager to incorporate these considerations into this early stage of ligand design.

The uti

stood. Althoug

inhibitors with

with moderately

based design is

database screen

ity of our screen

hydroxyethyl a

"gold standard"

ited by the trans

based aspartyl p

In the example

Dock was able

verted to likeli

screening is ab

compound not

than without t

assumption th

structure-base

database have

top 10% have

10% 45 time

cumstances.

The utility of structure-based design of small molecules is often poorly understood. Although it is easy to exaggerate its shortcomings (when it may fail to rank three inhibitors with vastly different K_i's) or its merits (when it correctly ranks three inhibitors with moderately different K_i's), a discussion of some reasonable expectations of structure-based design is warranted. Let us consider how likely a good scoring compound from a database screening is to be an inhibitor. First, we need an appropriate measure of the quality of our screen. Although the prevalence of hits in the 1000 compound cathepsin D (hydroxyethyl)amine library is much higher than our general database, this remains our "gold standard" for combinatorial screening. The implications developed below are limited by the transferability of structure-based screening performance from the mechanism-based aspartyl protease inhibitor library to the general database of combinatorial libraries. In the example presented here, when a "hit" was defined as a 330 nM inhibitor, CombiDock was able to find 80% (18 of 23) inhibitors in the top 10% of the database. When converted to likelihood ratios(45), these data indicate that a compound in the top 10% of a screening is about 10 times more likely to be an inhibitor than without the test, and that a compound not in the top 10% of a screening is about 4.5 times less likely to be an inhibitor than without the screening. For the purpose of this example, we make the optimistic assumption that, for the general database, 1 in 1000 compounds is a hit. If we now use structure-based design methods to screen the database, compounds in the top 10% of the database have about a 1% chance of being a 330nanomolar inhibitor, while those not in the top 10% have only a 1 in 4500 chance of being a hit. This makes compounds in the top 10% 45 times more likely to be inhibitors than those not in the top 10%! Under these circumstances, one would expect to have to assay *circa* 65-70 compounds in order to have a

50% chance of identifying an inhibitor with a K_i of 330 nM or better. On the other hand, assaying even 1000 compounds from the bottom 90% of the screening would only give you a 20% chance of identifying a good inhibitor. Despite the limitations noted above, this example simultaneously demonstrates both the power and limitations of structure-based design.

Conclusions

We have presented the first example of structure-based screening of a database of large combinatorial libraries. We've used this example to show that although identifying a good scaffold is essential for molecule design, it must also be complemented with appropriate side-chains. Further, we've demonstrated that early inclusion of pharmacokinetic properties (Lipinski's Rules) into the design process can dramatically alter library design, particularly for large libraries such as those used for initial exploratory libraries.

Acknowledgments

The authors are grateful to Todd J. A. Ewing for his development and documentation of the Dock 4.0 source code. We would like to thank Donna Hendrix, James Arnold, Michelle Lamb, Ken Brameld, and Keith Burdick for insightful discussions. This work was supported by GM31497 and by contributions from Daiichi and Pfizer.

Referen

1. E. K.

2. T. S.

3. Y. Su

12, 597-604 (

4. A. R.

Longman Lim

5. R. L. D

6. D. A. C

and Gene. 29, 5

7. L. R. H

8. J. R. So

9. C. S. Ri

10. A. Rusin

CORD. 192nd A

(1986).

11. E. C. Me

and Gene. 17, 26

12. B. K. Sh

1445-1450 (1993

13. M. D. Mi

Mol. Design 8, 15

14. T. J. A. E

References

1. E. K. Kick, *et.al.*, *Chem. & Biol.* **4**, 297-307 (1997).

2. T. S. Haque, *et.al.*, *J. Med. Chem.* **42**, 1428-1440 (1999).

3. Y. Sun, T. J. A. Ewing, A. G. Skillman, I. D. Kuntz, *J. Comp.-Aided Mol. Design* **12**, 597-604 (1998).

4. A. R. Leach, *Molecular Modelling Principles and Applications* (Addison Wesley Longman Limited, Essex England, 1996).

5. R. L. Desjarlais, *et.al.*, *Proc. Natl Acad. Sci. U.S.A.* **87**, 6644-6648 (1990).

6. D. A. Gschwend, W. Sirawaraporn, D. V. Santi, I. D. Kuntz, *Proteins-Struct. Func. and Gene.* **29**, 59-67 (1997).

7. L. R. Hoffman, I. D. Kuntz, J. M. White, *J. Virol.* **71**, 8808-8820 (1997).

8. J. R. Somoza, *et.al.*, *Biochemistry* **37**, 5344-5348 (1998).

9. C. S. Ring, *et.al.*, *Proc. Natl Acad. Sci. U.S.A.* **90**, 3583-3587 (1993).

10. A. Rusinko, J. M. Skell, R. Balducci, C. M. McGarity, R. S. Pearlman, CONCORD, 192nd American Chemical Society National Meeting, Anaheim, California (1986).

11. E. C. Meng, D. A. Gschwend, J. M. Blaney, I. D. Kuntz, *Proteins-Struct. Func. and Gene.* **17**, 266-278 (1993).

12. B. K. Shoichet, R. M. Stroud, D. V. Santi, I. D. Kuntz, K. M. Perry, *Science* **259**, 1445-1450 (1993).

13. M. D. Miller, S. K. Kearsley, D. J. Underwood, R. P. Sheridan, *J. Comp.-Aided Mol. Design* **8**, 153-174 (1994).

14. T. J. A. Ewing, A. G. Skillman, I. D. Kuntz, *in preparation* (1999).

15. M. Rarey, B. Kramer, T. Lengauer, G. Klebe, *J. Mol. Biol.* **261**, 470-489 (1996).

16. W. Welch, J. Ruppert, A. N. Jain, *Chem. & Biol.* **3**, 449-462 (1996).

17. V. Schnecke, L. A. Kuhn, Screening and Docking of Flexible Organic Ligands to Solvated Binding Sites with Induced Complementarity, Virtual Screening Workshop, Marburg, Germany (1999).

18. I. D. Kuntz, J. M. Blaney, S. J. Oatley, R. Langridge, T. E. Ferrin, *J. Mol. Biol.* **161**, 269 (1982).

19. P. J. Goodford, *J. Med. Chem.* **28**, 849-857 (1985).

20. H. J. Bohm, *J. Comp.-Aided Mol. Design* **6**, 593-606 (1992).

21. A. Miranker, M. Karplus, *Proteins-Struct. Func. and Gene.* **23**, 472-490 (1995).

22. J. A. Ellman, M. A. Gallop, *Current Opinion In Chemical Biology* **2**, 317-319 (1998).

23. B. A. Bunin, J. A. Ellman, *J. Amer. Chem. Soc.* **114**, 10997-10998 (1992).

24. S. H. DeWitt, *et.al.*, *Proc. Natl. Acad. Sci. U.S.A.* **90**, 6909-6913 (1993).

25. R. E. Dolle, *Mol. Diversity* **3**, 199-233 (1998).

26. R. E. Dolle, K. H. Nelson, Jr., *J. Combi. Chem.* **1**, 235-282 (1999).

27. C. A. Lipinski, F. Lombardo, B. W. Dominy, P. J. Feeney, *Advanced Drug Delivery Reviews* **23**, 3-25 (1997).

28. X. Bi, *et.al.*, *submitted* (1999).

29. M. Rarey, T. Lengauer, FlexX(c), Virtual Screening Workshop, Marburg, Germany (1999).

30. P. Walters, P. S. Charifson, J. J. Corkery, Ajay, M. A. Murcko, Virtual Screening - An Integrated Approach, Virtual Screening Workshop, Marburg, Germany (1999).

31. S. Mak

32. T. J. A.

33. G. Kleb

34. M. Stal

1999).

35. Sybyl, v

36. Availab

Leandro, Califo

37. D. Wein

mation Systems

38. P. Wille

& Sons, New Y

39. B. A. Bu

40. M. Poe,

try 11, 1023-103

41. M. Poe, H

42. V. J. Gill

(1997).

43. J. G. Top

217th National M

Anaheim, Califor

44. R. L. Des

Dock User's Grou

31. S. Makino, I. D. Kuntz, *J. Comp. Chem.* **19**, 1834-1852 (1998).

32. T. J. A. Ewing, I. D. Kuntz, *J. Comp. Chem.* **18**, 1175-1189 (1997).

33. G. Klebe, T. Mietzner, *J. Comp.-Aided Mol. Design* **8**, 583-606 (1994).

34. M. Stahl, OMEGA, version 1. (OpenEye Scientific Software, Santa Fe, California, 1999).

35. Sybyl, version 6.4. (Tripos Assicoates, St. Louis, MO, 1998).

36. Available Chemicals Database, version 97.2. (Molecular Design Limited, San Leandro, California, 1997).

37. D. Weininger, *Daylight Software Version 4.61 Manual* (Daylight Chemical Information Systems Inc., Santa Fe, New Mexico, 1999).

38. P. Willett, *Similarity and Clustering in Chemical Information Systems* (John Wiley & Sons, New York, NY, 1987).

39. B. A. Bunin, *The Combinatorial Index* (Academic Press, San Diego, 1998).

40. M. Poe, N. J. Greenfield, M. Hirshfiel, M. N. Williams, K. Hoogsteen, *Biochemistry* **11**, 1023-1030 (1972).

41. M. Poe, K. Hoogsteen, D. A. Matthews, *J. Biol. Chem.* **254**, 8143 (1979).

42. V. J. Gillet, P. Willett, J. Bradshaw, *J. Chem. Inf. and Comput. Sci.* **37**, 731-740 (1997).

43. J. G. Topliss, F. Yohida, QSAR Models for Drug Human Oral Bioavailability, 217th National Meeting of the American Chemical Society, Anaheim Convention Center, Anaheim, California (ACS, 1999).

44. R. L. Desjarlais, Attempts to Distinguish Orally from Non-orally Acting Drugs, Dock User's Group Meeting, Laurel Heights Conference Center, San Francisco, California

(UCSF. 1999).

45. A. M. C

Javanovich. Inc

46. E. T. Baldw

(UCSF, 1999).

45. A. M. Glenberg, *Data: An Introduction To Statistical Reasoning* (Harcourt Brace Javanovich, Inc., Orlando, Florida, 1988).

46. E. T. Baldwin, J. W. Erickson, *Proc. Natl. Acad. Sci. U.S.A.* **90**, 6796-6800 (1993).

Figure Captions

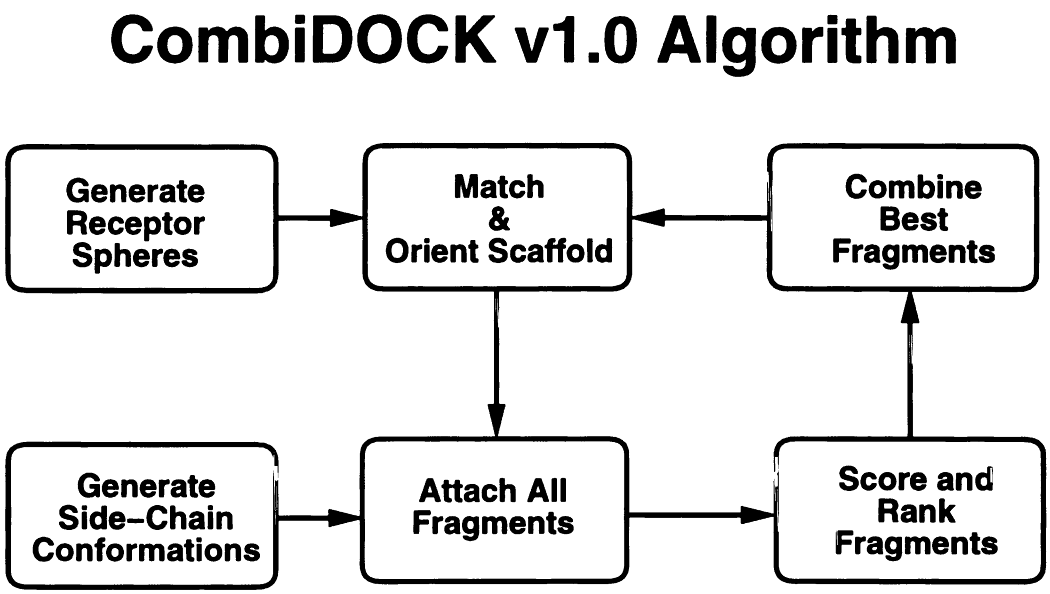


Figure 1: CombiDock version 1.0 algorithm. In two preprocessing steps, a negative image of the receptor site is generated with spheres, and multiple conformations of each side-chain are generated (top and bottom boxes on the left). The calculation runs as a cycle starting with orienting the scaffold in the active site by matching atom centers to sphere centers (top center). Next, all of the pre-generated side-chain conformations at each attachment point are attached to the scaffold in its new orientation in the active site (bottom center). All of the side-chain conformations are scored and ranked (bottom right). Finally, the best-scoring side-chains are combined to generate complete molecules and these are checked for intramolecular clashes (top right). This process is repeated for 100s to 1000s of scaffold orientations, and the best-scoring molecules are saved.

Figure 2: Com

in two places ir

scaffold dockin

before generati

ing the scaffold

which are used

probe side-chai

fold orientation

are used, where

ranking phase, fi

best-scoring con

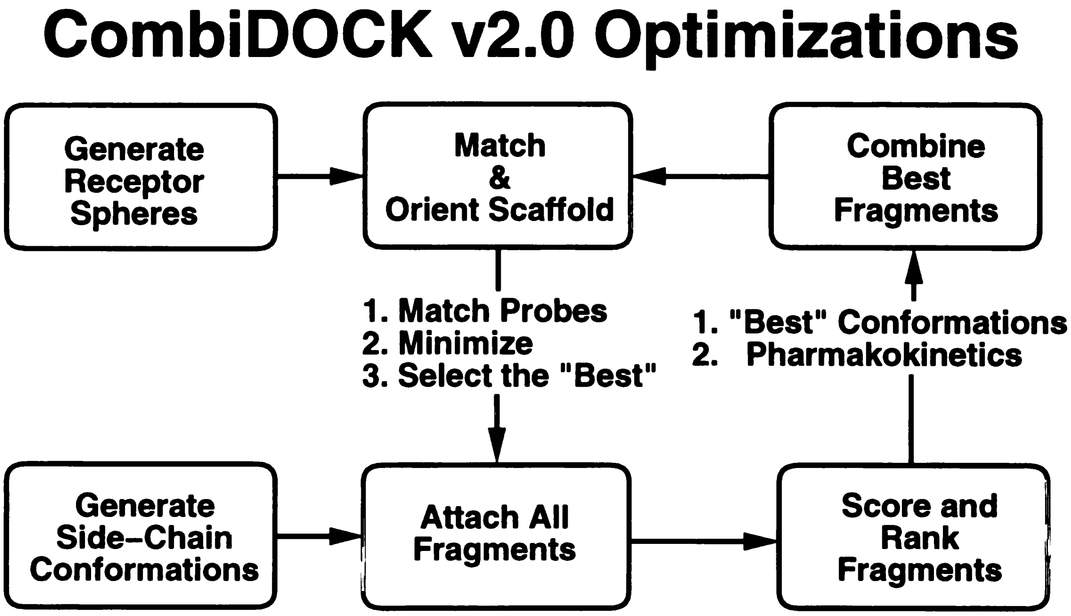


Figure 2: CombiDock version 2.0 algorithm. The new algorithm includes new features in two places in the cycle of library docking. There are three important changes in the scaffold docking process and two important changes in the way fragments are ranked before generating the complete molecules. In the scaffold docking, first, rather than docking the scaffold alone with no atoms attached, the user chooses a set of “probe” fragments which are used during scaffold orientation. Second, each orientation of the scaffold (and probe side-chains) are minimized. Third, rather than passing all of these minimized scaffold orientations on to the library docking cycle, the user the N best-scoring orientations are used, where N is a run-time parameter. In the side-chain conformation scoring and ranking phase, first, for each side-chain, all the conformations are scored, but only the M best-scoring conformations of each side-chain are passed on to the subsequent pharmaco-

kinetic and clash filters. Second, before the whole molecules (products) are screened for intramolecular clashes, the molecular weight, number of hydrogen-bond donor, and number of hydrogen-bond acceptors are calculated. Molecules which exceed user defined limits in any of these categories are discarded. The library docking cycle is executed N times (once for each scaffold orientation) and the best-scoring molecules are saved.

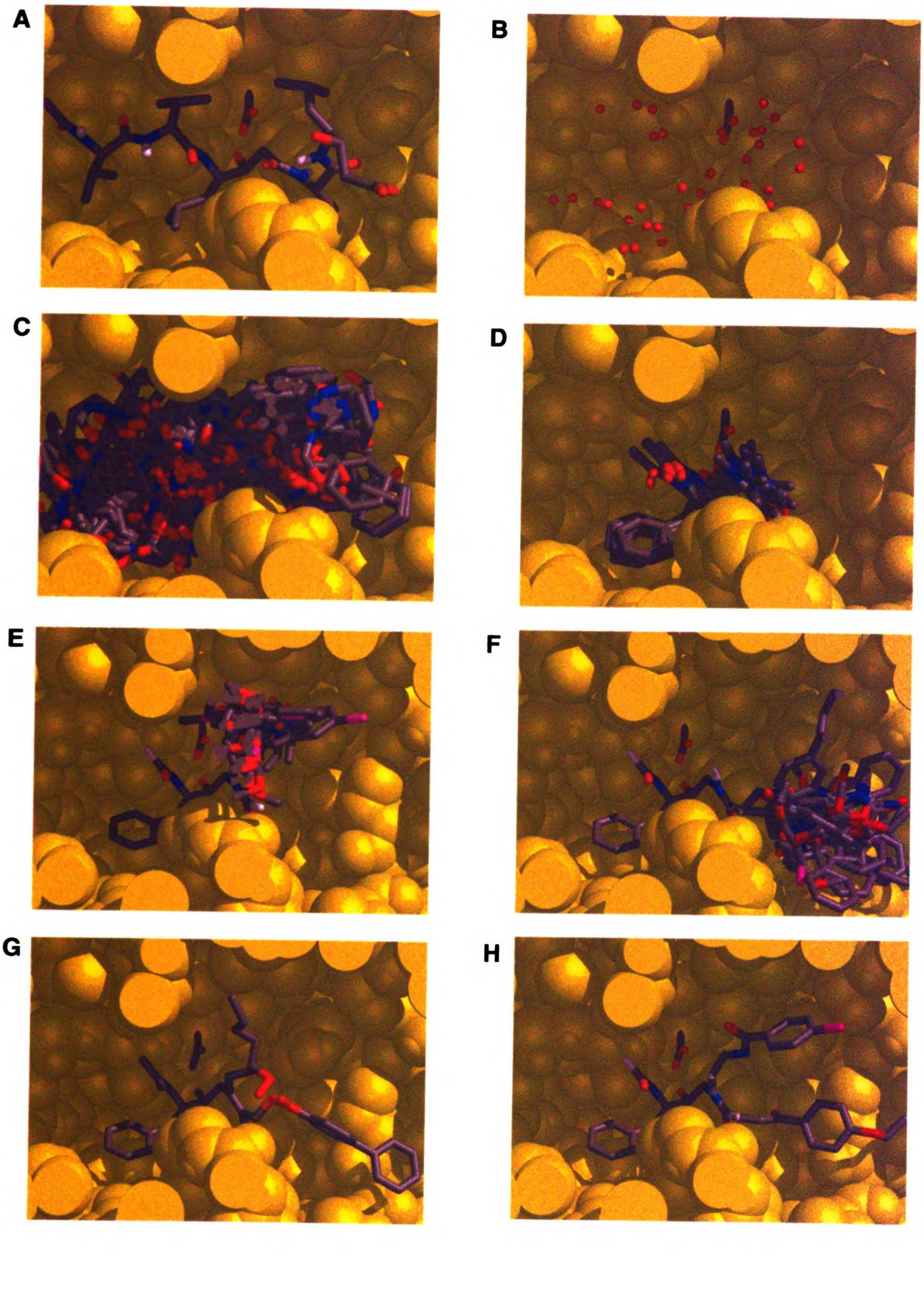

A
B
C
D
E
F
G
H

Figure 3: C
library in th
catalytic aspa
active site are
site. **C**) 251 or
fold orientatior
conformations
score. **F**) The p
attachment site
intramolecular
R_1 and R_2 side-
in the expensiv

Figure 3: CombiDock v2.0 Algorithm applied to screen a (hydroxyethyl)amine library in the cathepsin D active site(46). A) Pepstatin bound to the active site. Note one catalytic aspartyl is visible, the second is inferior and hidden. Portions of the flaps over the active site are cut away. **B)** Sphere centers describing the position and shape of the active site. **C)** 251 orientations of the scaffold in the active site. **D)** The eight best-scoring scaffold orientations after minimization (with methyl "probe" side-chains). **E)** All side-chain conformations at the R_1 attachment site are oriented, scored, and ranked according to their score. **F)** The process described in **E** is repeated at the R_2 attachment site and the R_3 attachment site (the later is not shown). **G)** The best potential molecules are examined for intramolecular clashes (clash highlighted in red). **H)** A ligand with *no* clash between the R_1 and R_2 side-chains. Steps **A-F** are in the linear phase (O(3N)), while steps **G** and **H** are in the expensive combinatorial phase ($O(N^3)$).

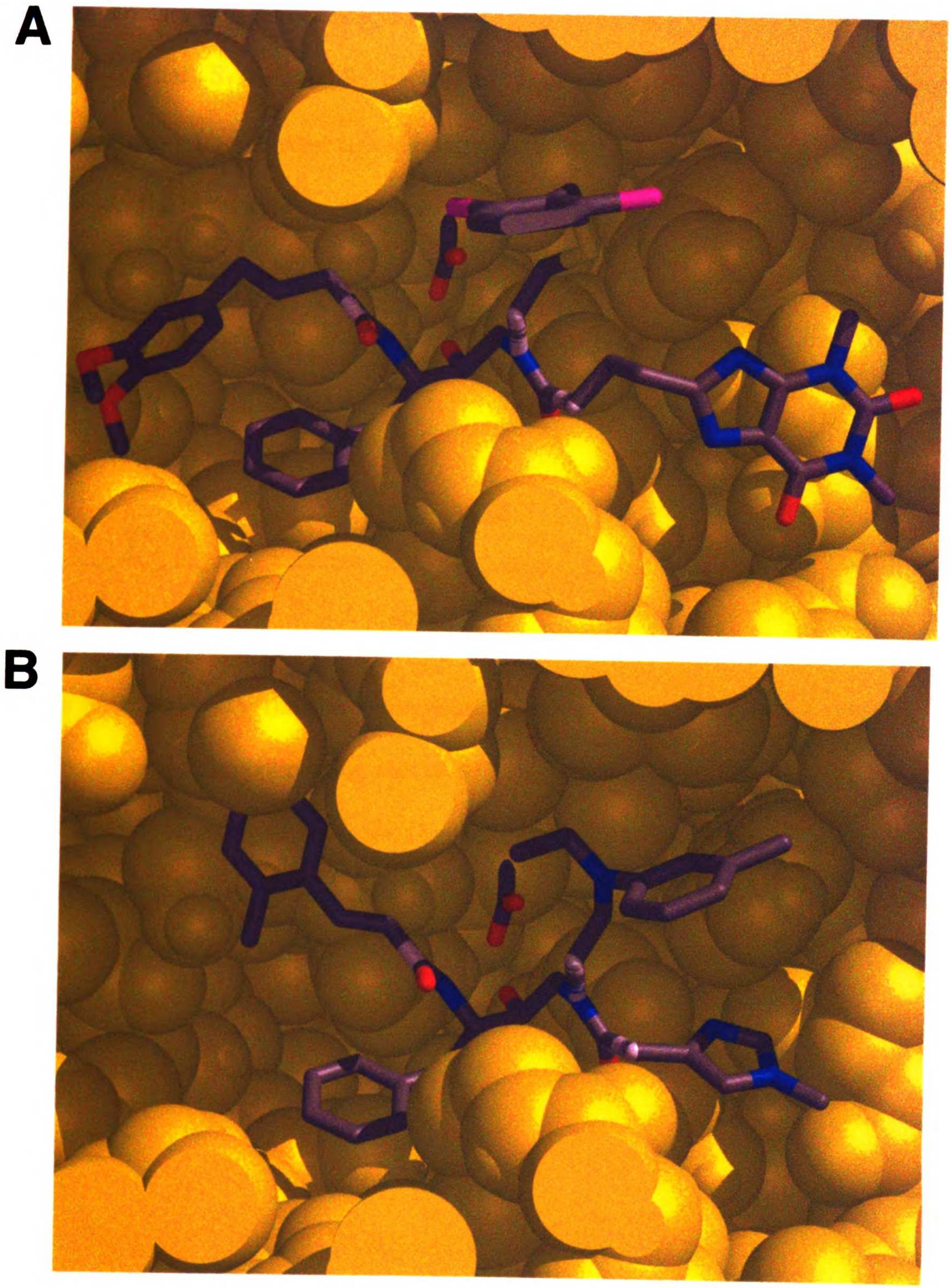


Figure 4: Example Ligands Constructed by CombiDock in the cathepsin D active

site. **A**) A large inhibitor selected without the Lipinski filter. The scaffold hydroxyl makes excellent contact with the catalytic aspartyl groups. Side-chains make good interactions in the S_3, S_2(partial), S_1, S_1', and S_2' pockets and also make good contacts with the covering flaps (cut-away). **B**) A smaller ligand selected with the Lipinski filter. Scaffold again makes essential hydroxyl-aspartyl interactions. Smaller, less flexible side-chains fill the S_2, S_1, S_1', and S_2' pockets.

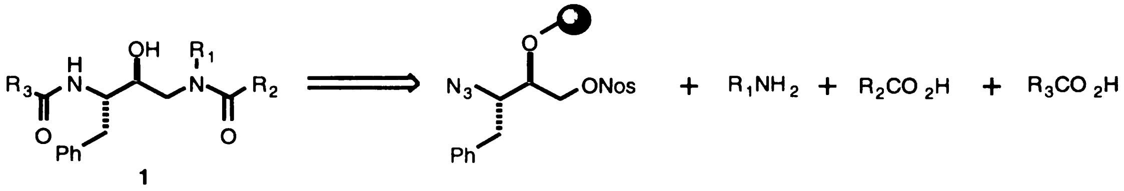
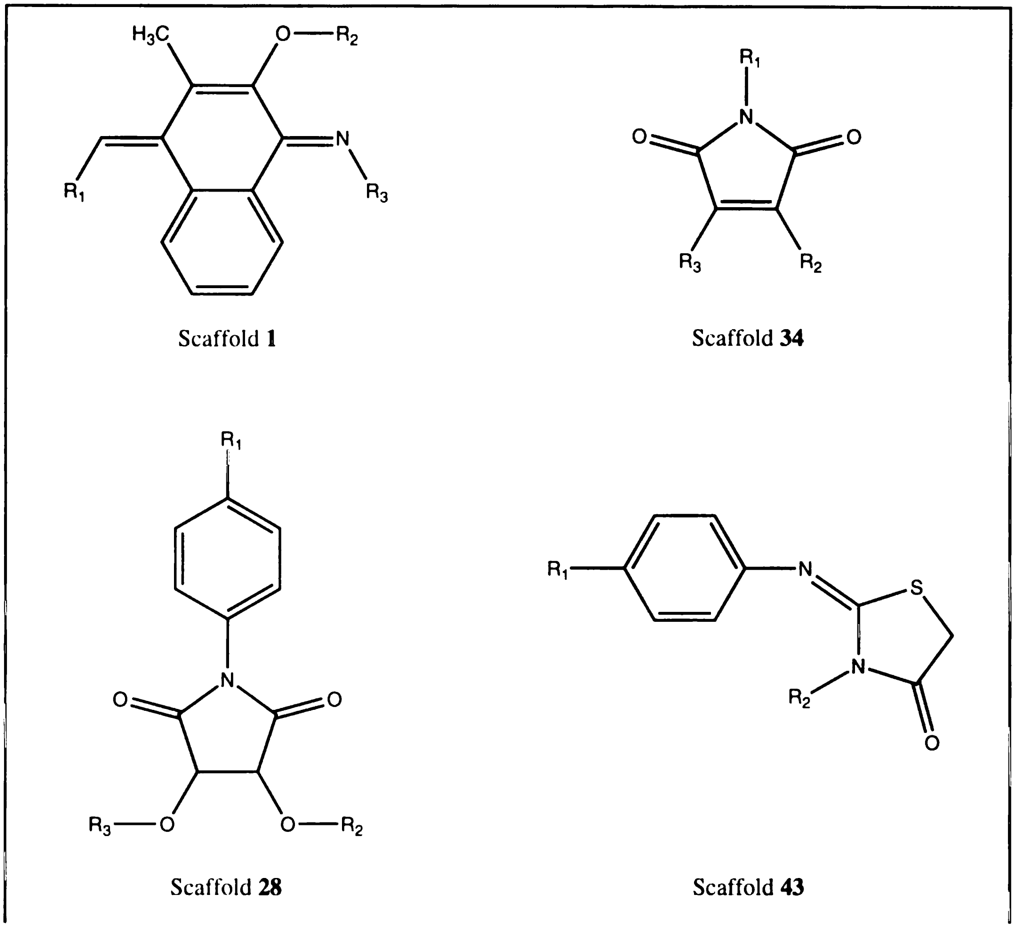


Figure 5: (hydroxyethyl)amine retrosynthetic scheme. The combinatorial library is synthesized on solid support using a set of primary amine reagents to generate the R1 side-chain. A set of acylating agents is used to generate the R2 side-chains, and another set of acylating agents is used to generate the R3 side-chains (after unmasking a second nucleophile in the scaffold).

Example Scaffolds:



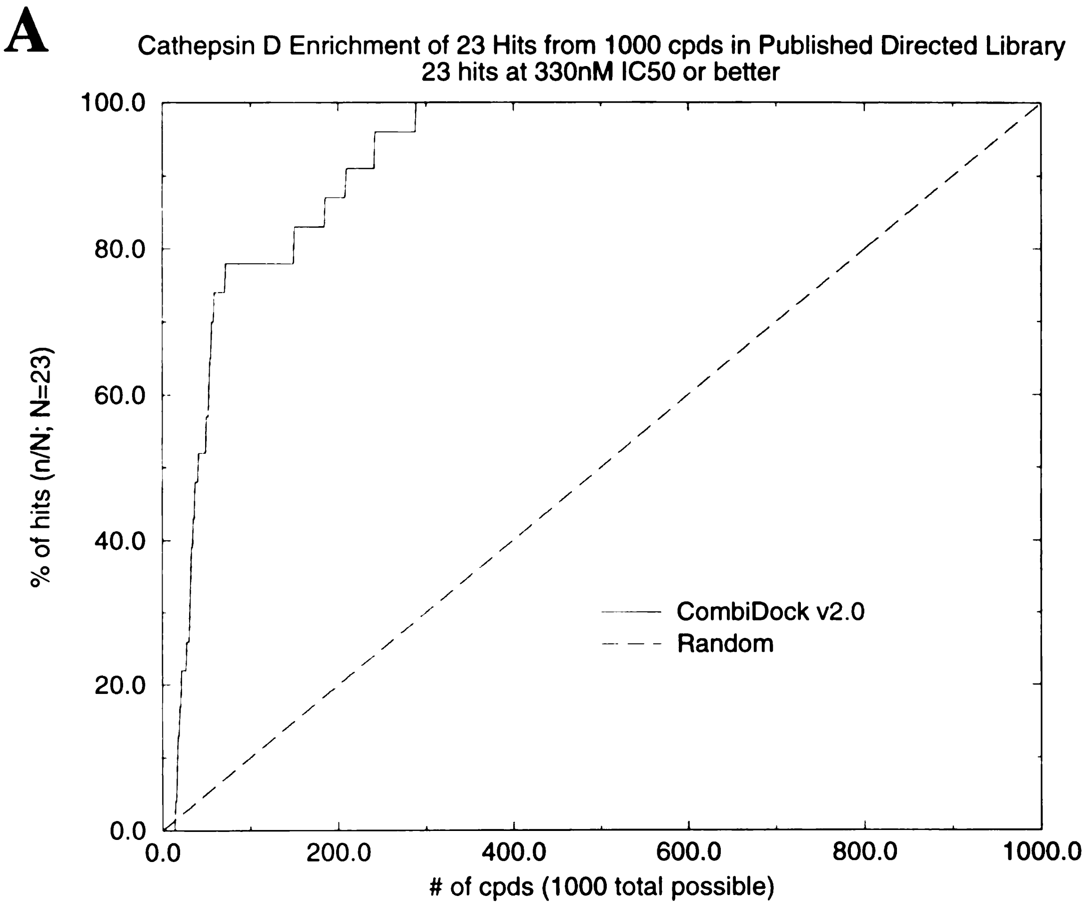
Scaffold **1**

Scaffold **34**

Scaffold **28**

Scaffold **43**

Figure 6: Example scaffolds. These are four of the forty-five scaffold used in the library database screening. Although they are not based on known combinatorial synthetic schemes, they are similar in size and composition to small heterocyclic scaffolds presented in the literature. Note scaffolds 1, 34, and 28 have three attachment points while scaffold 43 has only two. All of these scaffolds scored well in at least one of the four library database screenings.



Cathepsin D Enrichment of 23 Hits from 1000 cpds in Published Directed Library
23 hits at 330nM IC50 or better

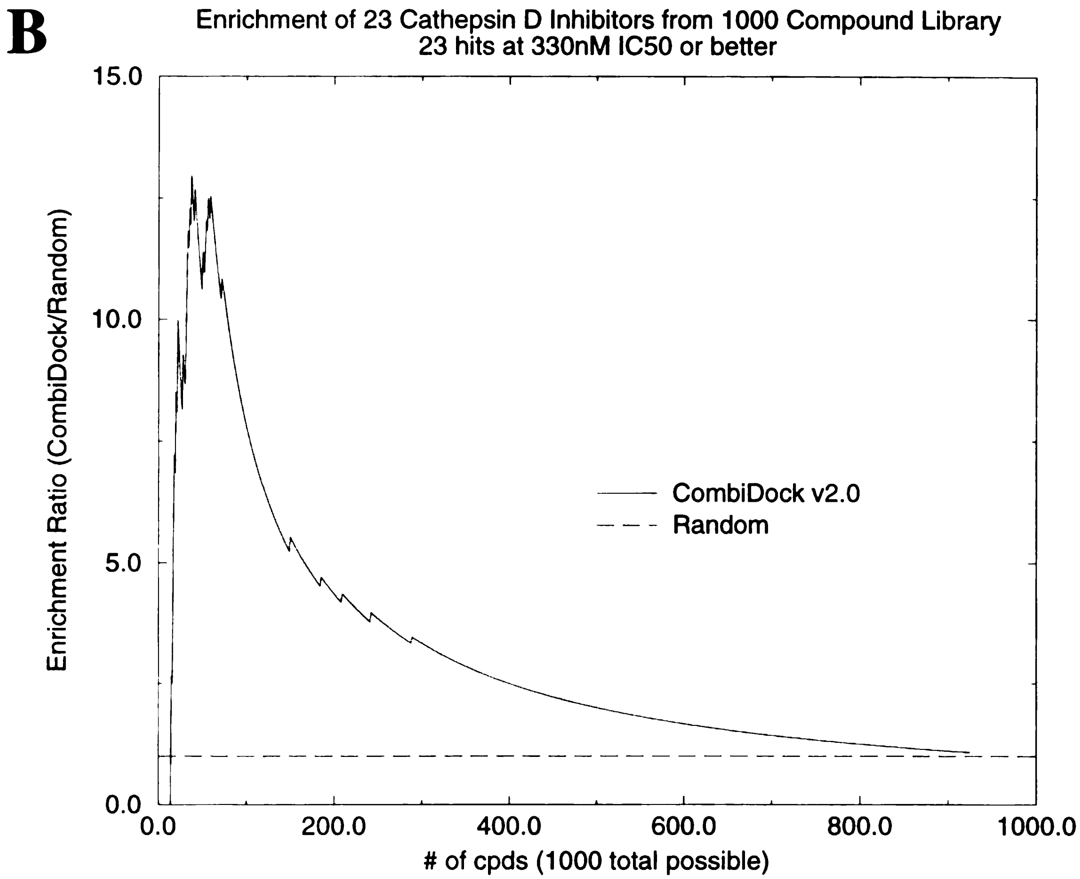


Figure 7: Enrichment of cathepsin D inhibitors from a 1000 member (hydroxyethyl)amine library. **A**) After using CombiDock to rank the library from 1-1000, the cumulative percent of inhibitors (out of 23) identified in each portion the database is shown (solid line). For comparison, the percent of compounds expected from a random rank of the library is displayed (dashed line). Nearly 80 percent of the inhibitors are identified in the top 10 percent of the database. **B**) The same data normalized by the number of compounds expected from the random ranking. When considering the top 5-10 percent of the database (as ranked by CombiDock) the number of inhibitors in increased by a factor of between 10 and 12.

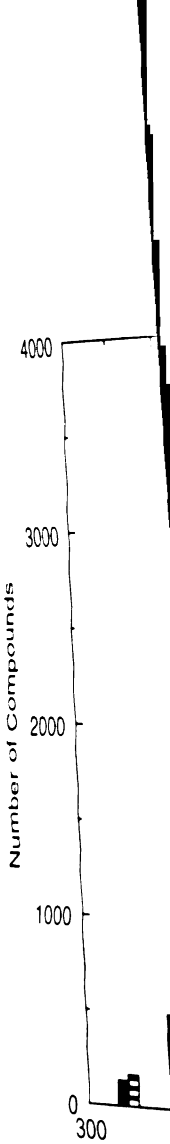


Figure 8: Molecula
the molecular weigh
database runs when
500. Ligand potency
a large number of c
value.

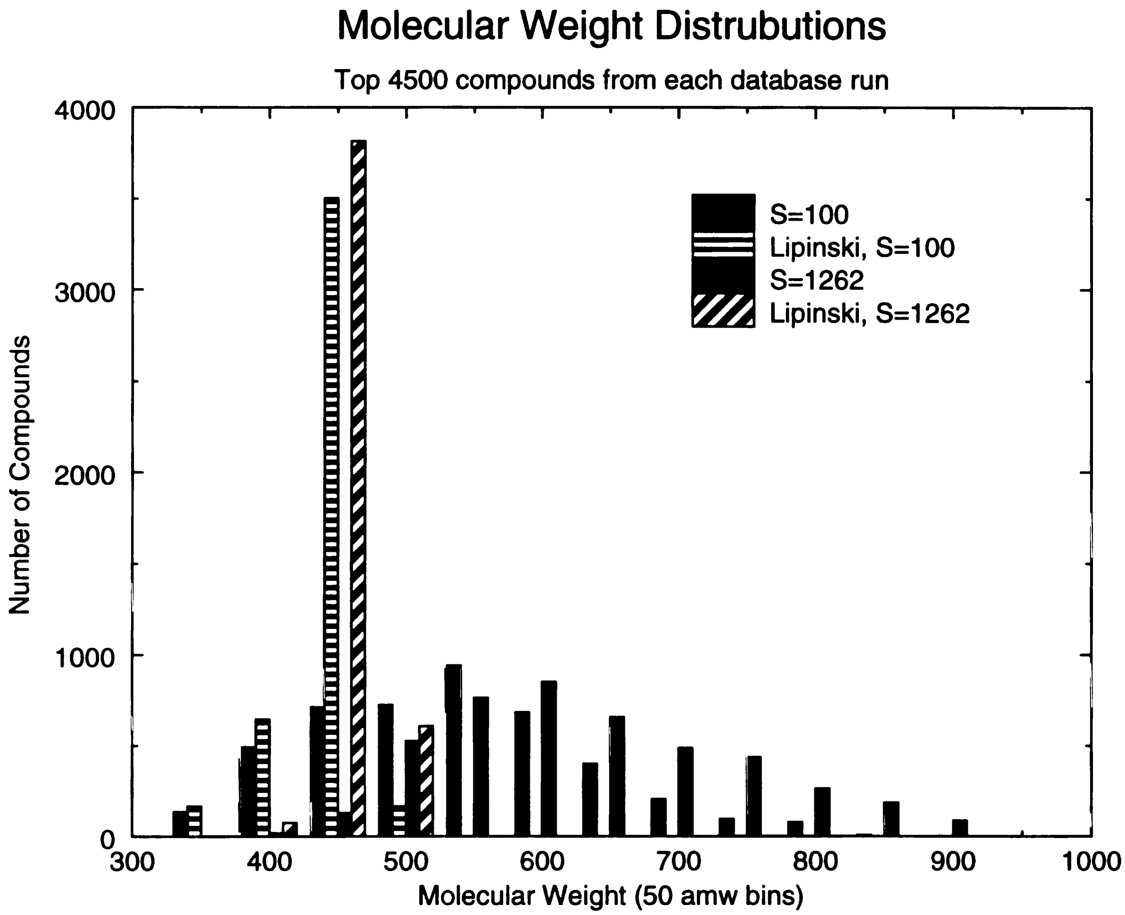


Figure 8: Molecular weight distribution. This histogram demonstrates the changes in the molecular weight distribution of the best-scoring molecules from each library in the database runs when the Lipinski filter is applied. The molecular weight cutoff was set at 500. Ligand potency, which is simultaneously being optimized, favors larger molecules so a large number of compounds appear at or slightly below the molecular weigh cutoff value.

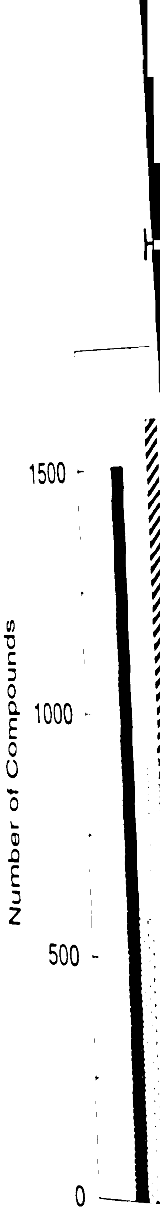


Figure 9: Hydrogen
in the hydrogen-bon
run when the Lipinsl
ter was set at 5.

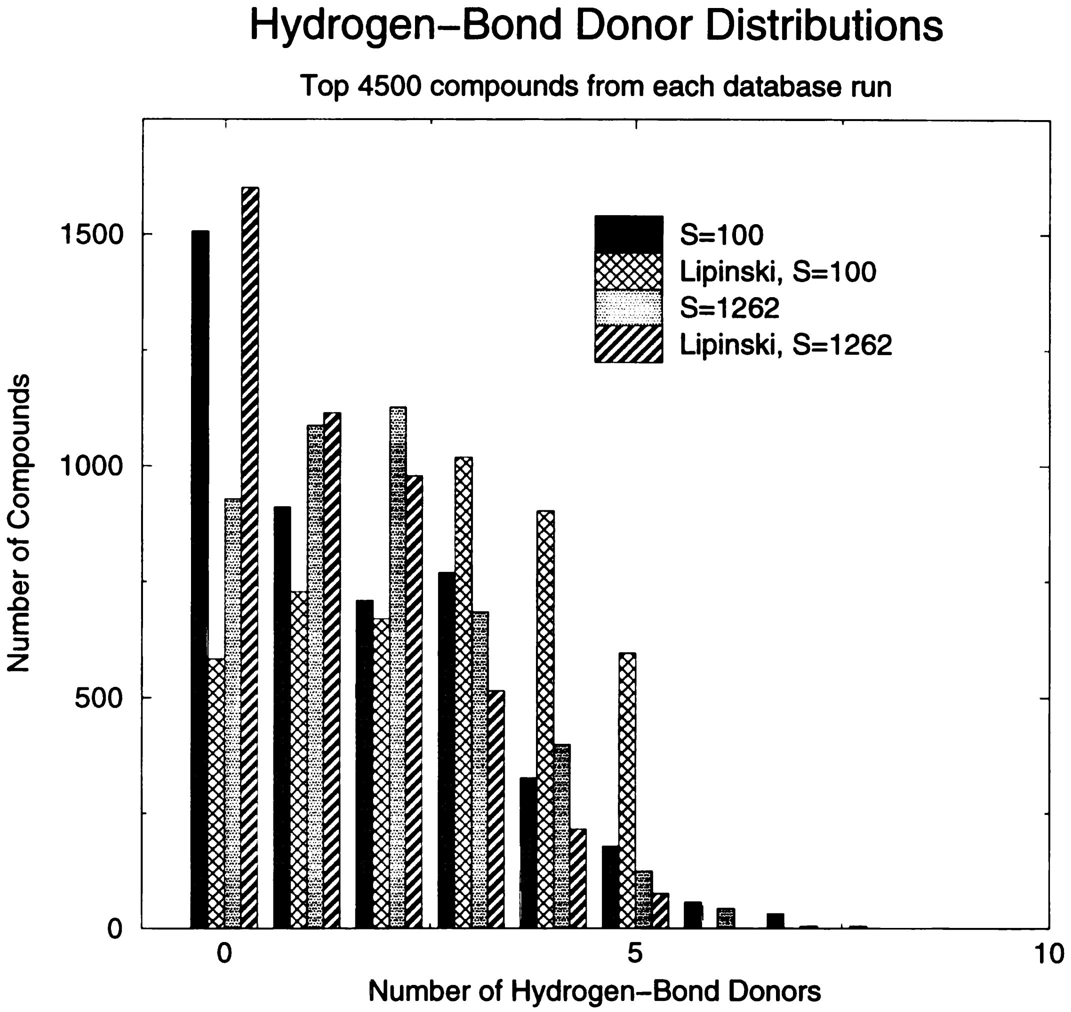


Figure 9: Hydrogen-bond donor distribution. This histogram demonstrates the changes in the hydrogen-bond donor distribution of the best-scoring molecules from each database run when the Lipinski filter is applied. The hydrogen-bond donor cutoff in the Lipinski filter was set at 5.

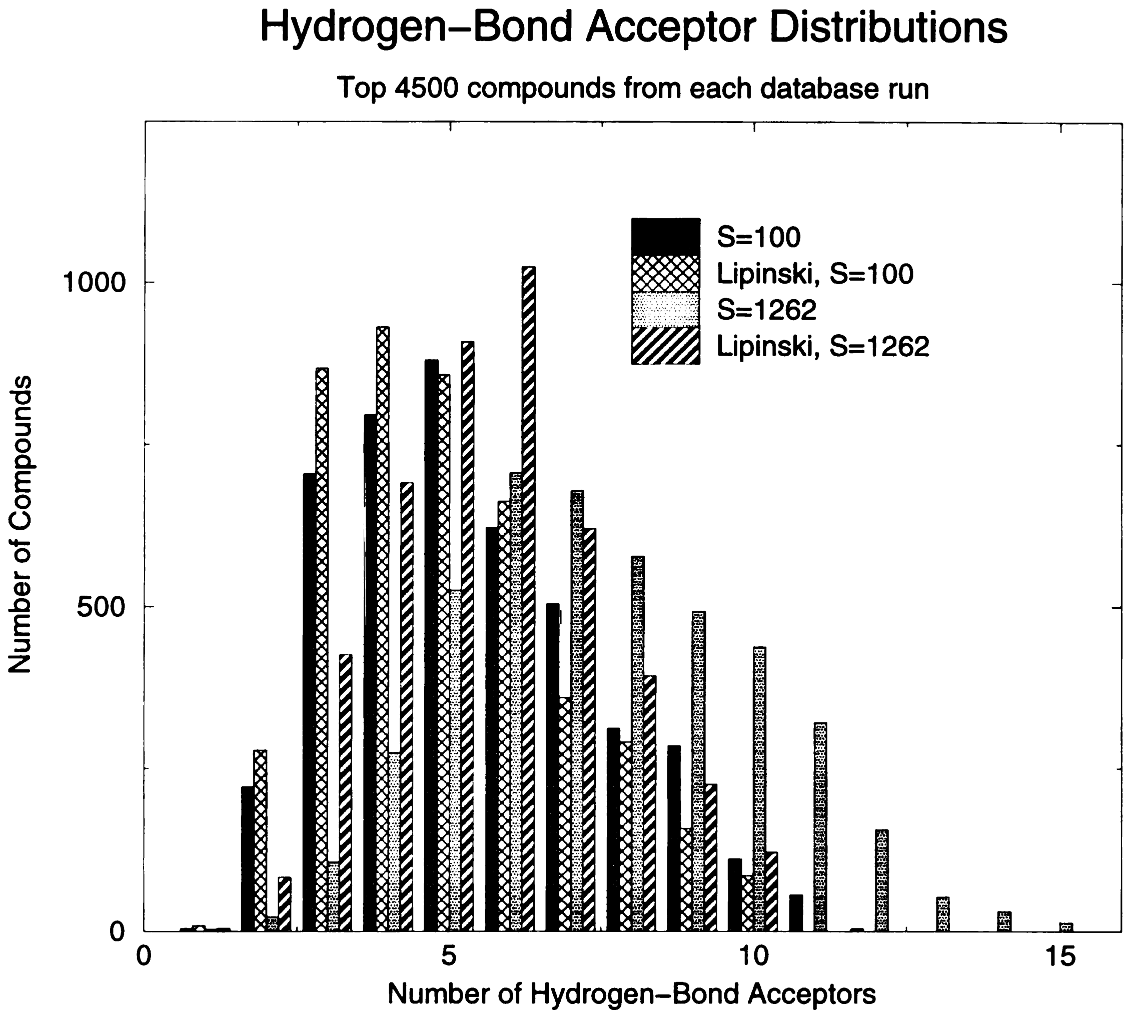


Figure 10 hydrogen-bond acceptor distribution. This histogram shows the changes in the hydrogen-bond acceptor distribution of the best-scoring molecules from each library in the database runs when the Lipinski filter is applied. The hydrogen-acceptor cutoff in the Lipinski filter was set at 10.

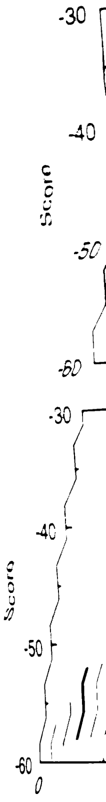


Figure 11: Small

top 100 compound

tical line represents

according to the bes

points are noted with

Lipinski filter while

ski filter.

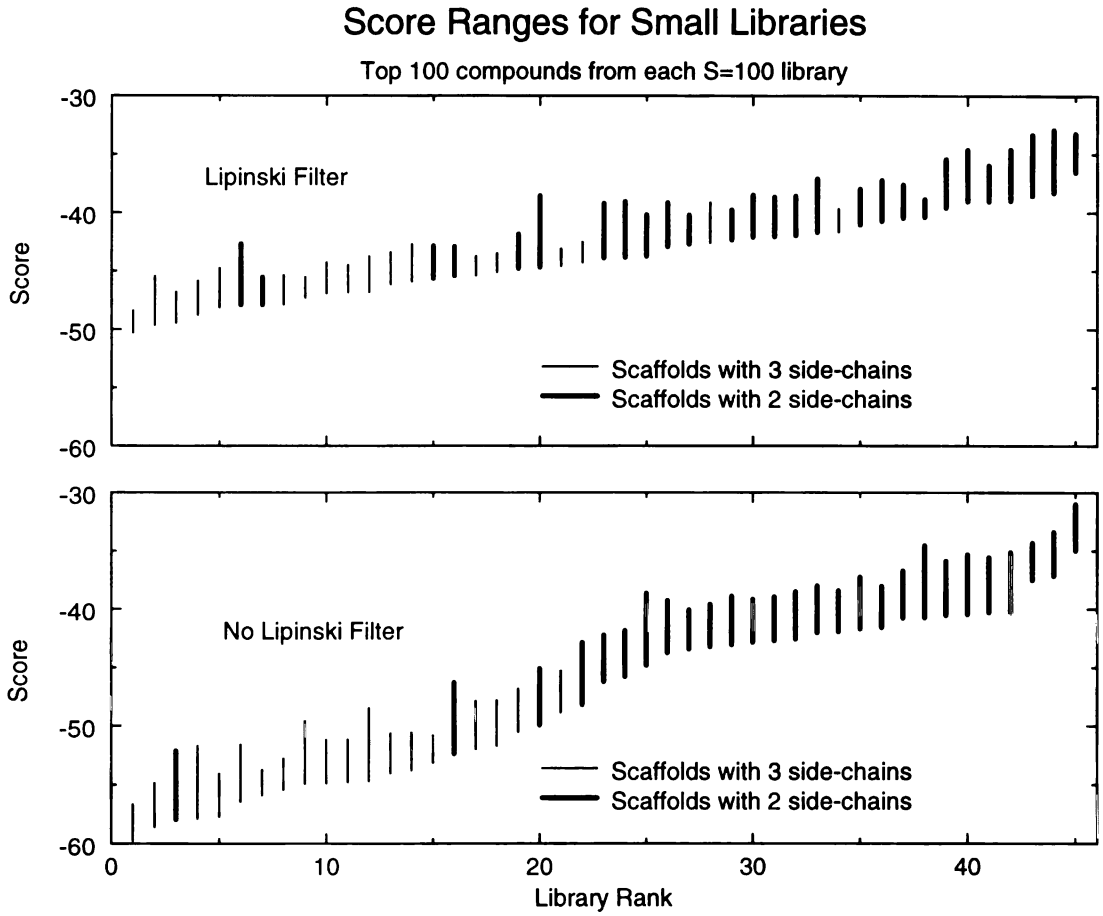


Figure 11: Small Library Ranking. These two graphs show the range of scores for the top 100 compounds from each library in the database runs with 100 side-chains. Each vertical line represents the range of scores for a single library. The libraries are ranked according to the best scoring compound in the library and libraries with two attachment points are noted with a bold line. The top window contains the database screening with the Lipinski filter while the lower window contains the database screening without the Lipinski filter.

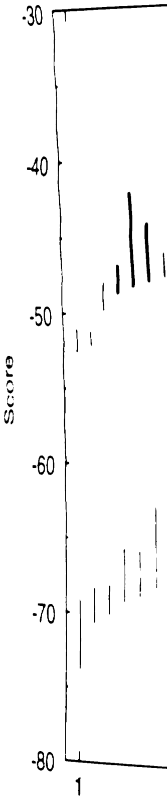


Figure 12: Large Li

compounds from eac

line represents the ran

in the library. The upp

lower set represent the

libraries with two attac

attachment points. The

any rank may not corre

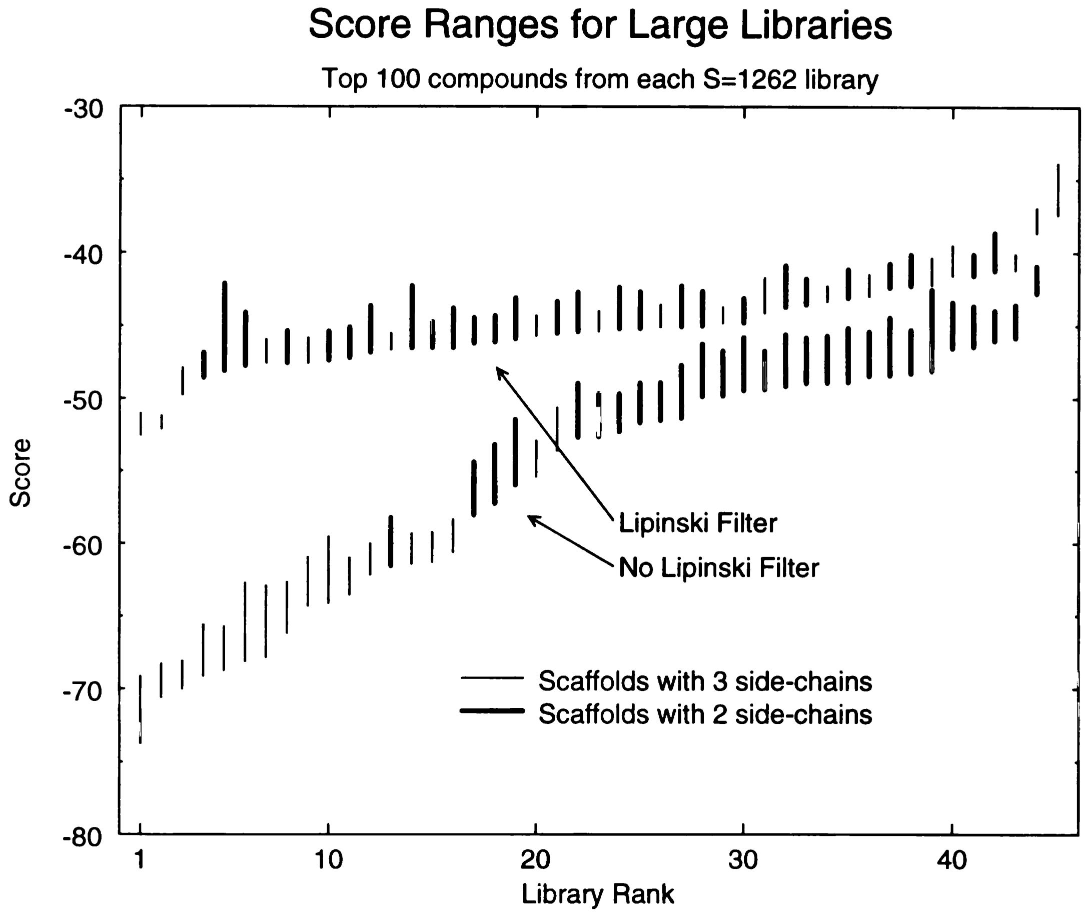


Figure 12: Large Library Ranking. This graph shows the range of scores for the top 100 compounds from each library in the database runs with 1262 side-chains. Each vertical line represents the range of scores for a single library, ranked by the best-scoring molecule in the library. The upper set of lines represent the run with the Lipinski filter while the lower set represent the run without the filter. As in figure 11, the heavy lines represent libraries with two attachment points while the lighter lines represent libraries with three attachment points. The libraries in each run are ranked independently, so the two lines in any rank may not correspond to the same scaffold.

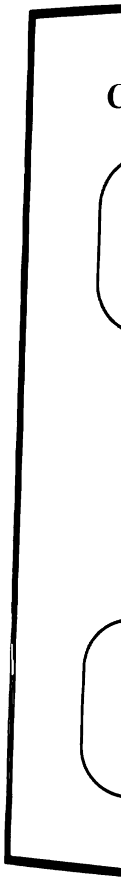


Figure 13:

the correlati

by one of th

ranking betw

Solid lines re

number asso

fit of library r

another datab

in the (N=100

run, the correl

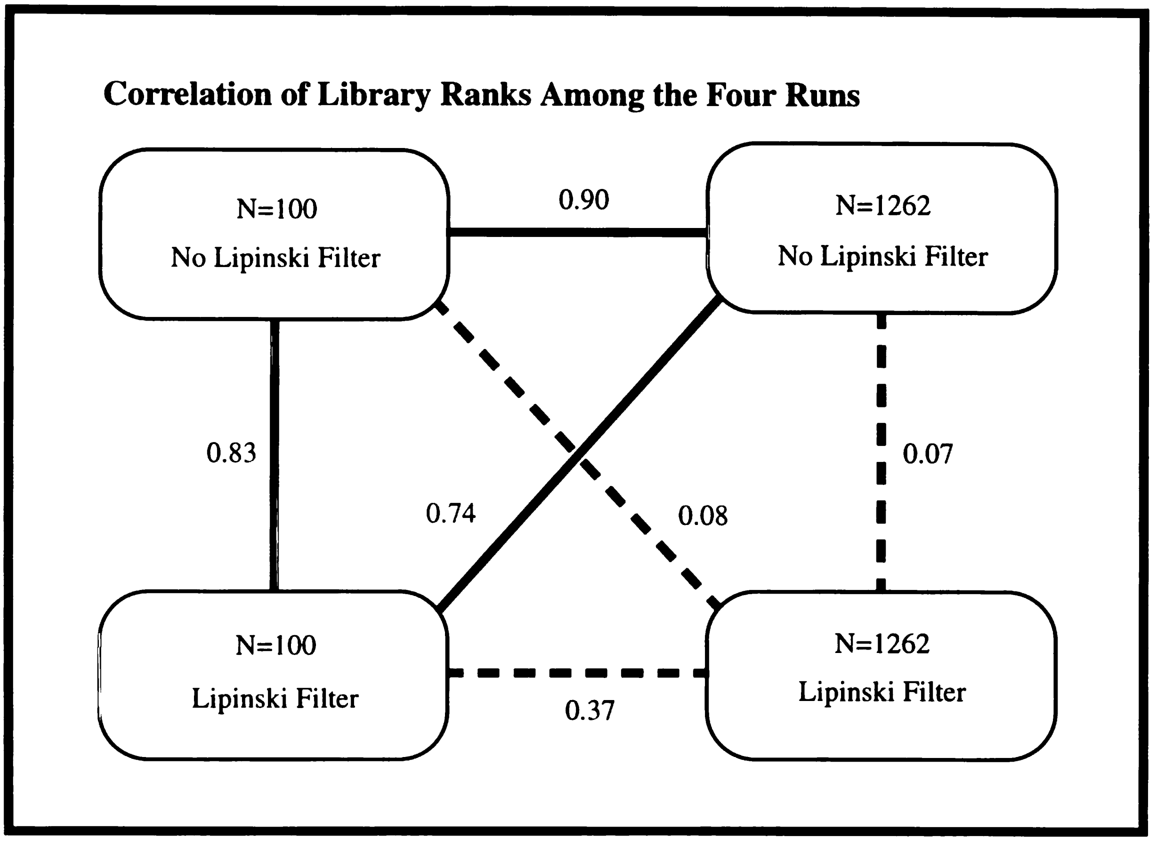


Figure 13: Library Ranking Correlation Diagram. This schematic diagram represents the correlation of library ranking between the four database runs. Each run is represented by one of the four rounded squares at each corner. The degree of correlation of the library ranking between each run are represented by lines between the corresponding squares. Solid lines represent high correlation, while dashed lines represent low correlation. The number associated with each line is the correlation coefficient of an unconstrained linear fit of library ranking from one database run plotted against the library ranking from another database run (*e.g.* when each library is plotted as a point (x,y), where x is its rank in the (N=100, No Lipinski filter) run and y is its rank in the (N=1262, No Lipinski filter) run, the correlation coefficient of a line fit through the points is 0.90).

Co

Chapter 4

Diversify: Computer-Assisted Construction of Molecular Libraries.

Abs

molecu

using r

ular rea

multi-st

engine v

ate prod

two exan

lize this

entropy c

ural prod

Introd

Co

synthesis

tage of lar

ceutically

relationshi

Small mole

cules with i

relationship

It is

space) which

Abstract

We describe a computational method to design and construct libraries of small molecules. *Diversify* is a computer programs which generates products from reactants using molecular transforms. The molecular transformations can be simple unimolecular reactions, complex multi-step enzymatic syntheses, combinatorial synthesis, or multi-step single compound syntheses. The core of this program is a versatile chemical engine which carries out "virtual" reactions on molecules and generates the appropriate products. We demonstrate the use of this transform-based "chemistry engine" in two examples. First, we generate and dock a virtual lead optimization library. We utilize this library as an opportunity to investigate a method to calculate conformational entropy of binding. Finally, we generate a virtual library of genetically engineered natural products.

Introduction

Combinatorial chemistry and combinatorial biochemistry have revolutionized the synthesis of medicinally relevant small molecules (*1*, *2*). These new syntheses take advantage of large sets of available starting reagents which can be combined to form pharmaceutically interesting products. One goal of medicinal chemistry is to understand the relationship between the structure of a molecule and its physical and biological properties. Small molecule combinatorial synthesis has been extremely successful in producing molecules with interesting biological properties (*3*, *4*) which should shed further light on the relationship between molecular structure and biological properties.

It is well known that the number of potential chemical compounds (chemical space) which may exist through acts of nature or human endeavor is astoundingly large

(5). Th
synthe
smaller
new me
be synth
of order
adapting
known c
cal synth
can be sy
use in oth

Co
pounds ex
chemistry.
libraries of
integrated
chemists h
their rapidl
ments invol
essence of t
number and
any compou

Comb

(*5*). The number of chemical compounds which can possibly be synthesized (potential synthetic space) by conventional organic or biological synthesis, is orders of magnitude smaller than chemical space, yet remains quite large, and is ever increasing in size with new methods and technologies. However, the numbers of compounds which have or can be synthesized in a reasonable amount of time today (practical synthetic space) is still tens of orders of magnitude smaller. Given the difficulty of developing synthetic methods and adapting them to new synthetic schemes, it is important to take as much advantage of known chemistries as possible. Not only is it important maximize understanding of practical synthetic space, but it is also important to categorize and catalog compounds which can be synthesized in a particular combinatorial library so that they are available for future use in other applications.

Combinatorial chemistry's increase in both the potential and real number of compounds explored by chemists has had a dramatic impact on the field of pharmaceutical chemistry. Combinatorial chemistry, including both solid and liquid phase synthesis of libraries of individual compounds as well as libraries of mixtures of compounds, has been integrated into many stages of drug development. As this integration has taken place, chemists have been challenged to develop methods to track chemical information about their rapidly expanding repertoire of combinatorial chemistries, the scaffolds and fragments involved, and the products formed. The work described here attempts to encode the essence of the chemical reactions in such a way that they can be used to maximize the number and diversity of potential products described (high sensitivity) without including any compounds which can't be practically synthesized (high specificity).

Combinatorial synthesis has had a particularly large impact on lead compound

optimizatio
lead compo
to identify
cially avail
Although I
it remains l
of the libra
evaluated.

The
molecules.
examine co
The concep
with the ret
chemical re
computatio
lysts, and r
mechanisti
class of co
environme
techniques
put an emp
ary regard f
application
dation (*11*).

optimization and has caused facile analog library generation to be a critical property for lead compounds. During the past decade, many computational approaches have been used to identify lead compounds. DOCK, one of these methods, screens libraries of commercially available compounds to find small molecules which bind to a macromolecule (*6, 7*). Although DOCK has successfully identified lead compounds with 1-10 μM IC_{50}s (*8-10*), it remains limited by the databases it searches. Here we will focus on expanding the scope of the libraries screened by DOCK and improving the scoring function by which they are evaluated.

There are many classes of computational methods for designing and manipulating molecules. Historically, the most prominent class are retrosynthesis programs, which examine complex molecules and help a chemist design a scheme for its synthesis (*11-13*). The concept of a "transform" as an abstraction of a chemical reaction was developed along with the retrosynthetic approach. Retrosynthesis programs use large knowledge bases of chemical reactions and complex strategies for their application (*14*). A second class of computational methods predicts reaction products when given reagents, solvents, catalysts, and reaction conditions (*13, 15*). These methods use molecular transforms based on mechanistic steps rather than complete reactions. *De novo* design programs are a third class of computational methods which modify small molecules inside a macromolecular environment in order to maximize a binding function. A wide variety of *de novo* design techniques have recently been applied to molecular design (*16*). *De novo* design methods put an emphasis on finding structures which optimize a scoring function with only secondary regard for the molecule's availability or synthetic accessibility. The first prosynthetic application of transforms was in CONGEN, a program developed to assist structure elucidation (*11*). The Diversify chemical engine uses ideas from each of these methods, applying the idea

The

cal engines

tion functio

engine com

of a genetic

a chemical

than model

My

The primar

forms) on t

organic che

thetically a

nomenclatu

vidually; ex

tions of the

including r

step necess

reagents fo

UC_Select

ment the nu

ing the num

application

encoding of

ing the ideas in a novel way to prosynthetic molecular design.

There are several other groups working on projects related to or involving chemical engines. In 1997, Daylight released a programming library with molecular-manipulation functions similar to some of the subroutines in my program (*17*). Daylight's chemical engine combined with a very large reaction library has recently been implemented as part of a genetic algorithm *de novo* design program (*18*). Alanex has independently developed a chemical engine (*19*). Their system is primarily being used for synthesis planning rather than modeling.

My chemical engine manipulates chemical structures according to a set of rules. The primary program, Diversify, takes reagents and carries out chemical reactions (transforms) on them to generate the products. By fashioning the rules after the principles of organic chemistry or biochemistry, molecules can be generated which are likely to be synthetically attainable. The chemical transforms are defined using common chemical nomenclature, a syntax familiar to all organic chemists. Reactions can be carried out individually; exhaustively; sequentially, following a specific synthetic route; or in combinations of these. Diversify quickly and reliably reproduces many types of chemistry, including ring closures, stereospecific and stereoselective transformations. An additional step necessary to assure synthetically accessible products is selection of available starting reagents for the chemical transformations. In chapter 2 (*vida supra*) we described UC_Select, a practical tool for this reagent selection process. These tools are used to augment the number of compounds being theoretically evaluated while at the same time limiting the number of compounds which are not synthetically practical. We will explore two applications where this selection of complete yet appropriate sets of starting materials, encoding of reactions, and generation of products can be utilized to investigate the biolog-

ical and physical prop
mization combinatoria
xanthine phosphoribos
isatin derivatives prese
mercially available isa
ating a structural libra
of *T. foetus* HGXPRT
thioester N-acetylcys
matically synthesize
diverse class of natu
astatin (figure 1). Re
synthesized *in vitro*
cal engine is used c
molecules are gene
library design.

To optimize
practically be carri
the field of medicin
able and which test
and demonstrate to
ing-mode hypothesi
associated with a pr
strong hypotheses fo

ical and physical properties of small molecules. The first example is design of a lead optimization combinatorial library around isatin inhibitors of *T. foetus* hypoxanthine-guanine-xanthine phosphoribosyltransferase (HGXPRTase) (see Chapter 6). Based on the nine isatin derivatives presented in Chapter 6, we chose a synthesis which combined the commercially available isatin hydrozone with a series of available benzaldehydes. After generating a structural library, we used DOCK to screen this library against the crystal structure of *T. foetus* HGXPRTase (*20*). The second example is analysis of a semi-synthetic thioester N-acetylcysteamine (SNAC) polyketide natural product library. Although enzymatically synthesized by repetition of simple chemistry, polyketides are an extremely diverse class of natural products which include therapeutics such as erythromycin and lovastatin (figure 1). Recently, both natural and genetically engineered polyketides have been synthesized *in vitro* (*21-23*), allowing the prospect of designing polyketides. If this chemical engine is used conservatively with chemically robust transforms, libraries of small molecules are generated which are useful for docking, lead optimization, and synthetic library design.

To optimize scientific efficiency, one must design experiments which both can practically be carried out and which yield the maximum possible pertinent information. In the field of medicinal chemistry, this means proposing compounds which are synthesizable and which test specific hypothesis about protein-ligand interactions. Here we describe and demonstrate tools to efficiently construct libraries of compounds and generate binding-mode hypothesis. These tools allow design of molecules which can be synthesized, are associated with a predicted binding mode of the ligand to the receptor, and thus provide strong hypotheses for experimental inquiry.

A. Condensation

B. Reduction

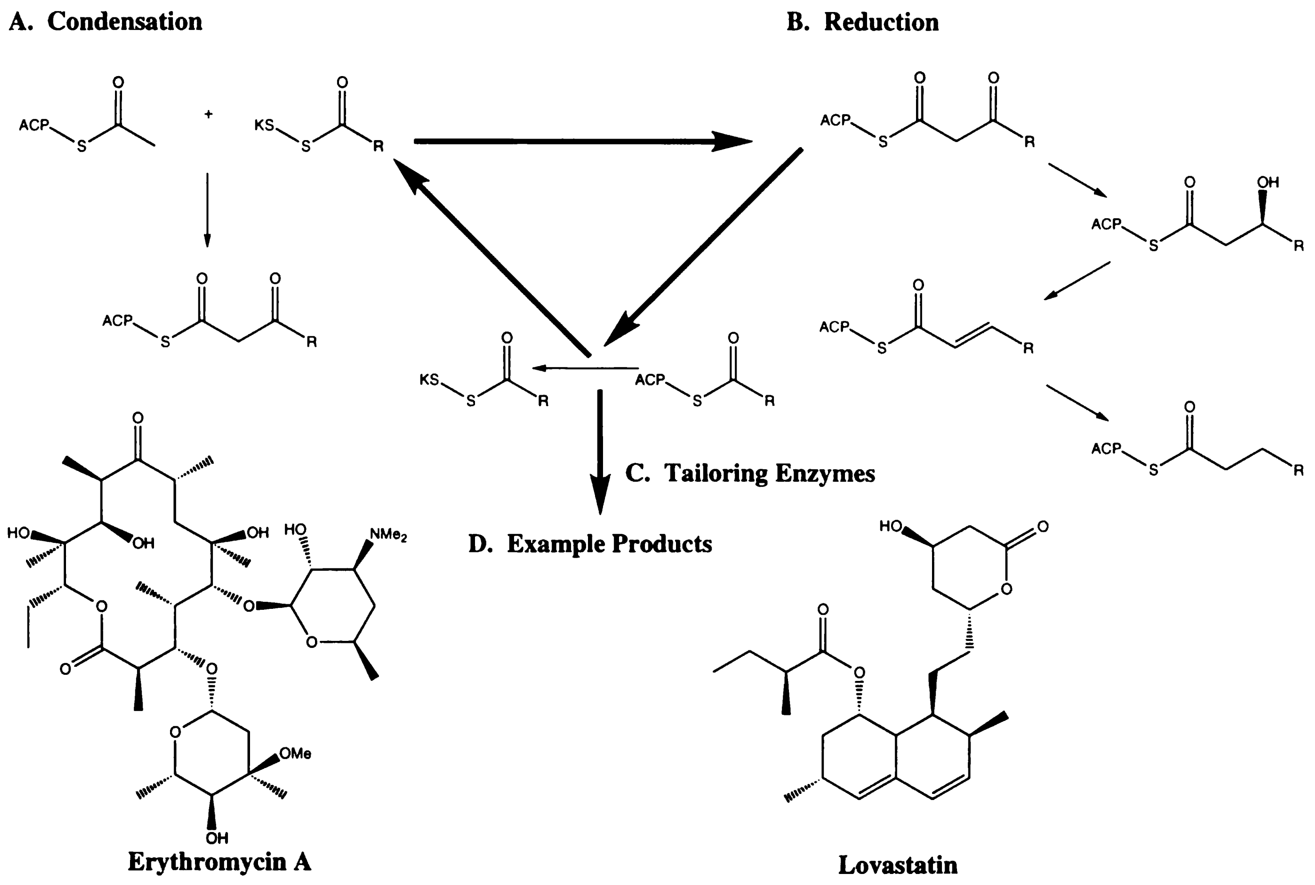


Figure 1: Polyketide Biosynthesis. Through repetative cycles of condensation (A) and beta-reduction (B) a core structure is built. Final tailoring enzymes (C) modify the core to give a variety of products (D). Diversity is generated by varying the feed stock, stereochemistry, degree of reduction, and tailoring enzymes. (ACP-acyl carrier protein, KS-keto-synthase)

Meth

(

rules. or

my prog

engine

tivity. c

formed

generic

catalys

substru

substru

classes

try eng

classe

replac

reacti

are th

gener

Repla

usual

transf

classe

Methods

General Considerations. My chemical engine uses generic reaction-like rules, or transforms, in a prosynthetic manner. Given a starting material and a transform, my program generates probable products. The most important commodities for a chemical engine are molecules and transforms. Molecules in my program contain: atoms, connectivity, charge, the reactant(s) and transform(s) used to create them, a record of products formed from them, and a unique identifier. The transforms for my chemical engine are generic and do not necessarily include solvent or specific conditions (e.g. temperature, pH, catalysts). Instead, their essential components are: a reactive-site substructure, a product substructure, an atom to atom mapping of the reactive-site substructure onto the product substructure, and a list of functional groups not compatible with the reaction. Some classes of transforms contain additional information as discussed below.

Transforms form the critical interface between synthetic chemistry and the chemistry engine. Although transforms have been previously defined (*12*), I developed four classes of transforms based on the relation of starting materials to products. These are the replacement, division, connection, and polymerization transforms (figure 2). All of the reactions explored to date can be accommodated by these classes. Connection transforms are the most important class of transform for library construction. This is because they generate a series of related products by joining a single reactant to a series of reagents. Replacement and separation transforms play a slightly lesser role in library construction, usually making minor changes in a molecules which allow it to undergo a connection transform. Polymerization transforms are much less common than the other transform classes.

A. Repl

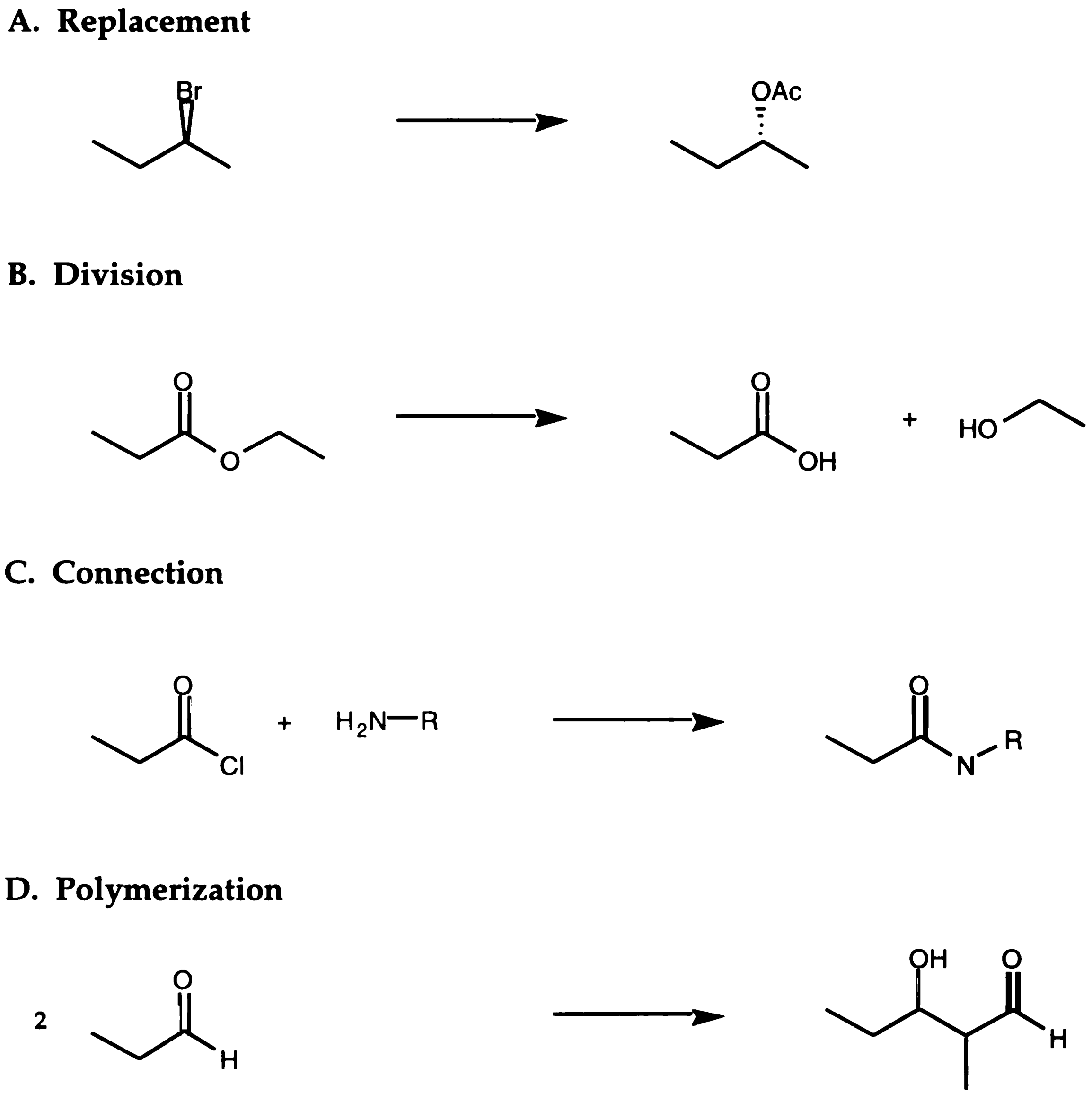
B. Divi

C. Cov

D. P

2

Figu
con
rep
be
for
fo
m


A. Replacement

B. Division

C. Connection

D. Polymerization

Figure 2: Transform Classes. The four classes of transforms are replacement, division, connection, and polymerization. Transforms are abstractions from reactions and do not represent a specific mechanism or technique. There are often several methods which could be used to carry out the chemistry represented by a transform. Further, a series of transforms does not necessarily represent an efficient synthetic scheme, rather, a series of transforms is a method for generating structures. These classes have been sufficient to incode much of synthetic chemistry including rearrangements and cycloadditions.

The series of reagents a connection transform joins to the single reactant are called the "connection reagents." Each connection transform has its own "connection reagents", and

con:

com

whic

nece:

try. Ii

limit

or noi

nectio

compo

gramm

tines (

and ex

and ea

binatio

produc

down i

istry, st

the mo

and selection of these reagents is a critical aspect of both transform generation and library construction. There are two stages to "connection reagent" selection: selection of reagents compatible with the transform's chemistry, and selection of a subset of those reagents which will actually be synthesized. In the first stage, careful investigation of the reaction is necessary to properly select side chains which are compatible with the connection chemistry. In the second stage, component supplier, cost, and molecular weight can be used to limit the initial number of "connection reagents." Structural calculations, such as docking, or non-structural calculations, such as diversity, can be used to select a subset of the "connection reagents" to be used in synthesis. An algorithm has been worked out to aid library component selection.

Technical Details. The chemical engine is written in C using Daylight's programming libraries and takes advantage of Daylight's efficient structural searching routines (*17*). Molecules in the chemical engine are represented by the chemically intuitive and extremely compact SMILES format with implicit connectivity (*24*).

Transforms need to be in a format that makes it both easy to write new transforms and easy to read what change a transform will make to a molecule. Transforms use a combination of SMILES and common chemical nomenclature, to represent the reactant and product substructures (figure 3). Specifying the reaction site in transforms has been broken down into primary effects (selectivity and specificity) and secondary effects (stereochemistry, steric influence, electronic influence). Whenever possible, transforms are written in the most general form to broaden their applicability.

A. Ch

B. Re

C. A

D. I

Figure
reactio
ten us
SMIL
produ

integr

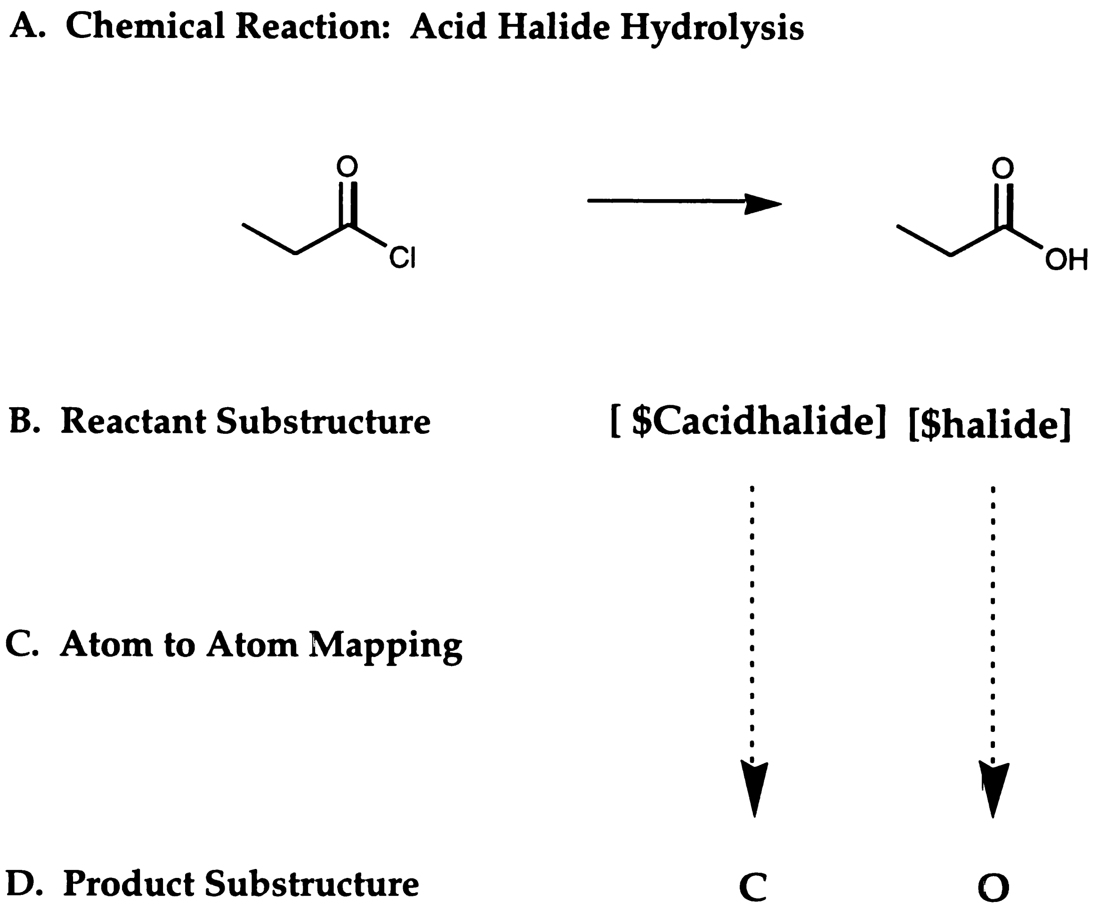
chem

can b

(ACD

Data

from



A. Chemical Reaction: Acid Halide Hydrolysis

B. Reactant Substructure

C. Atom to Atom Mapping

D. Product Substructure

Figure 3: Transform data structure. Each transform is designed to mimic a chemical reaction (A). The essential elements of a transform are: the reactive-site substructure, written using common chemical nomenclature (B); the product substructure, written using SMILES (D); and an implicit atom to atom mapping of the reactant substructure onto the product substructure (C).

The utility of Diversify is enhanced by its computational environment. Diversify is integrated into Daylight's database system, which is specifically designed to manage chemical information (*17*). The output from UC_Select reagent searches (see Chapter 2) can be directly imported into Diversify. Furthermore, the Available Chemicals Database (ACD), Comprehensive Medicinal Chemistry Database (CMC), and the Medicinal Drug Data Report Database (MDDR) (*25*) have all been converted to Daylight format. Output from the chemical engine can be read back into the database system for similarity searching and

tural dat

Exan

example

the cons

upon bi

polyket

biosynt

als for

Exar

fied in

relatec

substr

our m

devel

desig

deve

cont

pote

ing and clustering. Further, the SMILES products of Diversify can be converted into structural databases for docking using a variety of methods (*26, 27*).

Examples

As mentioned above, we will now discuss two applications of Diversify. The first example is a combinatorial library focused on optimizing a lead compound. We describing the construction and docking of this library, including ligand conformational entropy loss upon binding. The second example is a description of using Diversify to model a polyketide synthase multienzyme system. To demonstrate the integration of synthetic and biosynthetic transforms in Diversify, we model the syntetic preparation of starting materials for encorporation into the polyketide synthase.

Example 1: Optimization Library

Problem Formulation. Based on the promising HGXPRTase inhibitors identified in chapter 6, we desired to design an optimization library. Having already explored related compounds which were available for purchase through extensive similarity and substructure searching, we sought methods to synthesize libraries of compounds based on our most potent inhibitors. We considered both the phthalanhydride and isatin libraries and developed proposed binding modes for each library to direct our initial efforts in library design (see figure 1 b and c in Chapter 6).

For each library, we explored potential synthetic schemes which would allow development of libraries to explore our structure-based binding models. The isatin library contains a hydrazone, so condensation of the available hydrazine with aldehydes was one potential scheme. In addition, we considered synthetic schemes using a Wittig reaction (with the

condensa

The phth

ment.

V

phthalar

acylate a

is revers

acylate

6 table

solubil

medici

ment.

medic

hyde

I

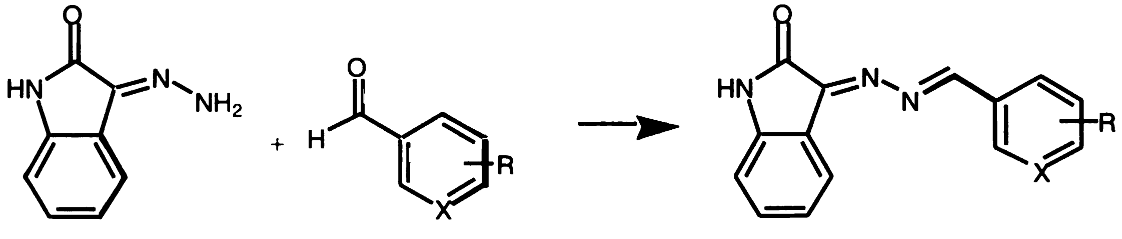
H

(with the isatin as either the Wittig reagent or the electrophile), reductive amination, aldol condensation, thioether formation, Grignard reaction, or Palladium coupling reactions. The phthalanhydride compound's simple amide linkage make it ideal for library development.

We also considered the medicinal properties of both lead series. Although the phthalanhydride is perhaps the most stable anhydride available, it retains the potential to acylate a variety of biological nucleophiles. In cell culture, TF1 inhibition of HGXPRTase is reversible (Chapter 6), yet this does not completely eliminate the potential that TF1 may acylate other cellular targets. The isatin lead compounds (compound **16** and **17** in Chapter 6 table 1) contain a moderately unstable hydrazone, and the isatins have only moderate solubility in water. Although neither class of compounds is ideal, both have potential for medicinal development.

Both the isatin and the phthalanhydride classes appear ripe for further development. We considered the isatin library first because of its familiarity to the collaborating medicinal chemists. The synthetic scheme consisted of a condensation of substituted aldehydes with the 3 hydrazine derivative of isatin.

Isatin Derivative Synthetic Scheme



X=5,6 member aromatic or heteroaromatic ring

We sought to integrate the chemical information and the structure-based model to pred

atter

con

stru

po

po

tr

w

v

v

r

u

predict which subset of the potential isatin library should be given the most synthetic attention. Our goal was to enrich the number of HGXPRTase inhibitors in the synthesized compounds by prioritizing the synthesis of compounds according to their score in the structure based-design model. The ease of this exercise is a direct result of selecting compounds in the initial screening which were amenable to simple synthesis.

Reagent Selection. The program UC_Select (Chapter 2) was used to identify potential reagents for this optimization library from the ACD v97.2. The primary functional group necessary in the reagents was an aldehyde, and for the purpose of this library, we limited the search to aldehydes alpha to a ring. The search was limited to compounds with molecular weight of 100-450 daltons, calculated logP of -2 to 5, 0 to 5 hydrogen bond donors, 0 to 10 hydrogen bond acceptors, 0 to 9 rotatable bonds, and 0 to 2 formal charges (*28*). Compounds which contained phosphor based acids or esters, primary or secondary amines, hydrazones, hydroxylamines, hydrazines, acid halides, sulfonic acid or esters, anhydrides, peroxides, azides, azos, long (>7) unbranched chains, four or more halides, metal atoms, two nitro groups, dipeptides, or macrocycles (n>=8) were eliminated. Finally, the search was limited to compounds available from Salor Chemical, TCI U.S., Sigma, Fluka, Aldrich, Maybridge, CalBio, ICN, Indofine, Acros, Lancaster, Transworld, or Pfaltz & Bauer.

Library Construction. We used Diversify to create an imine by the combination of each aldehyde selected with UC_Select and the available hydrazine isatin derivative (*vida supra*). The transform used for this *in silico* reaction (table 1) first identified the terminal nitrogen in the hydrazine moiety of the isatin and prepared it to form an intermolecular double bond. In each aldehyde derivative, the aldehyde group was identified and the

Transform name	Reactive atom set	Product atom set
hydrazine	N[N;D1]	NN&=1
aldehyde	[$Caldehyde]=[$Ocarbonyl]	*&=1X

Table 1: Isatin Library Transforms. Two half transforms were used to generate the imine library from the hydrazine and aldehydes. The hydrazine transform identifies the terminal nitrogen in the isatin hydrazine and creates a double bond to an "external" fragment (identified by the numeral 1). The aldehyde transform identifies the carbonyl carbon and carbonyl oxygen of the aldehyde, copies the carbon unchanged, removes the oxygen, and completes the imine product by forming an "external" double bond to the isatin hydrazine fragment prepared in the first half-transform (identified with the matching numeral 1).

carbonyl oxygen was removed. The carbonyl carbon was then bonded to the terminal nitrogen of the isatin molecule with a double bond. This resulted in the appropriate product molecule. After completion of the transform on the heavy atoms, the implicit hydrogens were reassessed by Diversify to assure the appropriate charge and valence states (assuming no change in tautomer).

The three dimensional structure of each product was generated using CONCORD (*26*) as implemented in Sybyl version 6.4 (Tripos, St Louis, MO). The regular angles and bond distances from CONCORD are more appropriate for the subsequent conformational sampling in DOCK's incremental growth algorithm (*vida infra*) than methods which generate bond angles and distances optimized for a specific conformation (e.g. distance geometry). Partial atomic charges were calculated for each product molecule using the method of Gasteiger-Marsilli (*29*) as implemented in Sybyl 6.4.

Library Screening. The library was docked into the active site of *T. foetus* HGXPRTase based on the structure of the complex of this enzyme to guanosine-mono-phosphate (GM

Skillman.

entire libr

and orien

tical anch

each proc

pled at ea

tional mi

After eac

mationa

Figure 4
ented by
carbon o

phate (GMP) (*20*). The incremental growth algorithm of Ewing et.al. (Ewing, T.J.A, Skillman, A.G, Kuntz, I.D., submitted) was used with a fixed anchor to flexibly dock the entire library to the rigid HGXPRTase receptor. Chemical matching (*8*) was used to match and orient the isatin fragment on top of the crystallographic guanine (figure 4). This identical anchor placement was used for each library member. The remaining side chains of each product were built in one rigid fragment at a time. All torsional minima were sampled at each rotatable bond, and torsional minimization as well as rotational and translational minimization of the growing molecular fragment were carried out at each step. After each step of growth, the search was narrowed to a set of 25 good scoring and conformationally

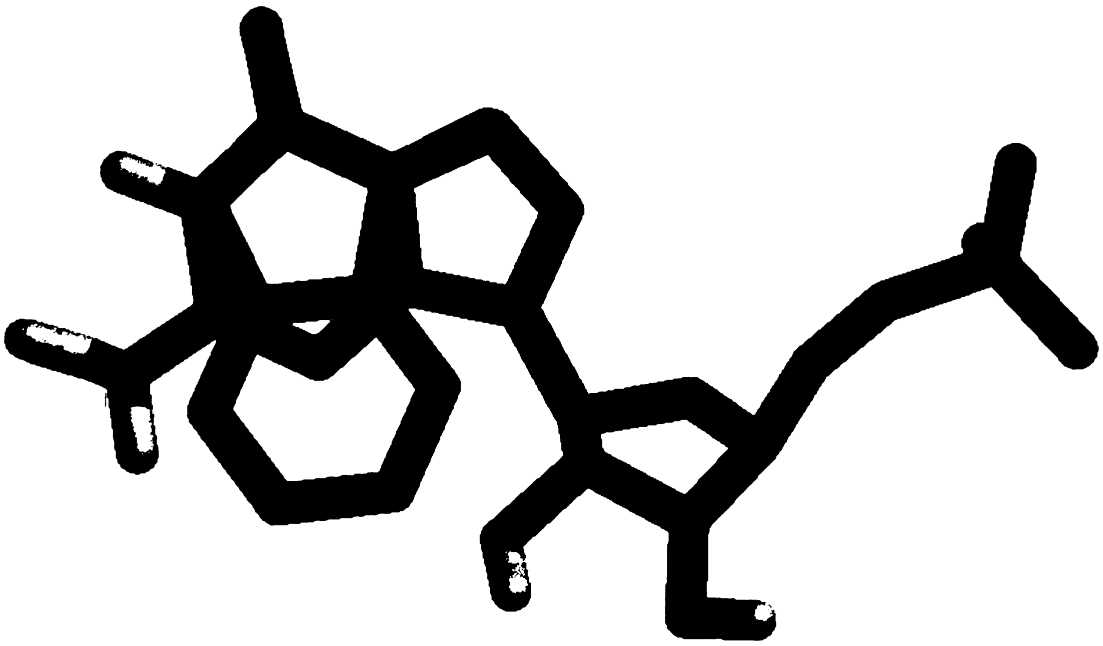


Figure 4: Placement of the Isatin Anchor. The isatin anchor fragment (green) was oriented by overlaying the donor amide nitrogen, the acceptor amide oxygen, and the amide carbon of the isatin with those from the crystallographic GMP from the HGXPRTase

active site.

diverse fragments. The calculation was repeated three times with different random seeds for both chemical and force-field scoring. A single best conformation for each molecule from each scoring function was saved for each run and was used in subsequent solvation rescoring (*vida infra*). All of the compounds in the library were ranked according to their best score among the three runs for both scoring functions. Neither the molecular mechanics scoring function nor the empirical scoring function in DOCK take account of the loss of conformational entropy upon binding of the ligand to the receptor. Because the size of the library is limited to about 700 compounds by the availability of compatible aldehyde reagents, more detailed score evaluation was feasible. The standard methods of scoring were corrected with a conformational entropy term (see appendix to this chapter).

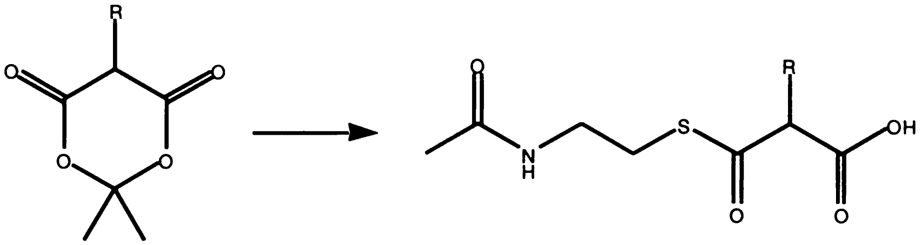
Solvation Scoring. We also assessed the use of a new Generalized-Born/Surface Area (GBSA) scoring function (*30*). The GBSA method is not a pairwise calculation and thus is CPU limited in its current implementation, so the best scoring conformations from force-field and chemical scoring were reminimized and scored under the GBSA method. Six conformations of each molecule in the library (three from chemical scoring, three from force-field scoring) were reminimized and rescored with the GBSA method. The library compounds were reranked based on the best of the six solvation scores for each compound and the top 4% of compounds were assessed visually in their proposed binding mode in the active site.

Despite being known inhibitors, compounds **10**, **12**, **16**, and **17** from Chapter 6, which are members of this library, were not among the top scoring compounds. Since these are known inhibitors, they were used to explore the implicit assumption in rescoring,

that driving conformation construction with one scoring function will not inappropriately bias the second scoring function. Compound **17**, the most potent compound in the series, was used as a test case. The standard DOCK 4.0 torsion sampling parameters were replaced with sampling of every flexible bond at 30 degree increments. All conformations of **17** in the HGXPRTase active site with a favorable score under force-field and chemical scoring were generated and saved. All of these conformations were reminimized and rescored with GBSA solvation scoring and compared to the scores obtained by only rescoring the single best pose from each scoring function.

Example 2: Natural Product Library

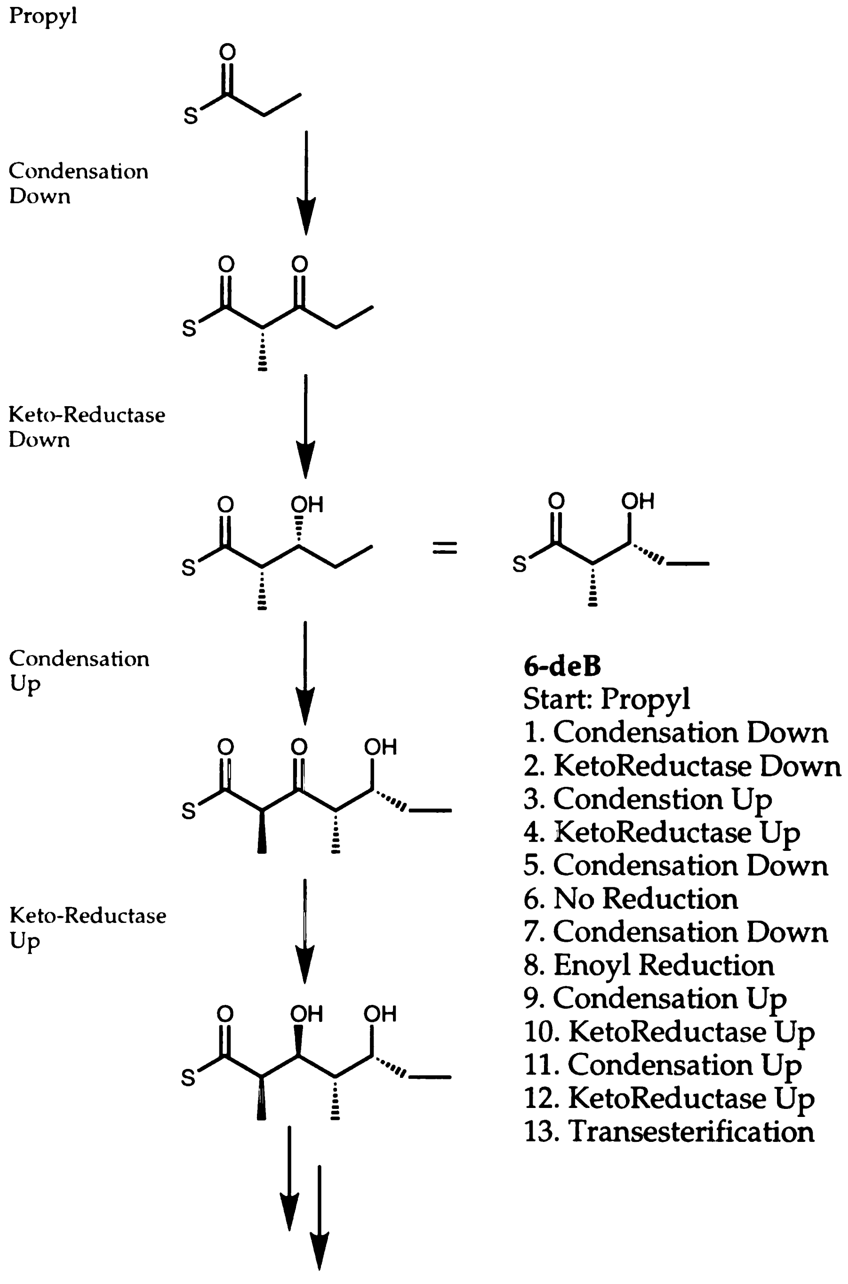
We use the polyketide example to demonstrate 1) Diversify can be used to model biologically significant systems of reactions 2) Diversify can be used to model synthetic schemes using multiple reactions in sequence, and 3) Diversify can be used to build large combinatorial libraries by combining libraries of reactions in a synthetic scheme in addition to combining libraries of reagents. Two recent experimental developments form the basis for this example. First, the multienzyme reaction complex which synthesizes the erythromycin precursor polyketide natural product has been cloned and the synthetic path elucidated (*21-23*). Second, it has been shown that a variety of thiaminoacetyl (SNAC) compounds can be fed to the bacteria and thus incorporated as starting materials into the polyketide products (*31*).



Synthesis of N-acetylcysteamine precursors from analogs of Meldrum's acid

We used UC_Select (Chapter 2) to identify commercially available meldrum acid SNAC precursors. Transforms for Diversify were written which mimic each of the possible condensation and reduction reactions seen the they enzymatic synthesis of macrolactones. For condensations, three reactions were considered; malonyl-CoA condensation, and methylmalonyl-CoA condensation (branched methyl with R or S stereochemistry). For reduction, five reactions were considered; no reduction (ketone preserved), ketoreduction (hydroxyl with R or S stereochemistry), dehydration (trans double bond), and enoyl-reduction (yields a methylene). For a 14-member macrolactone, such as erythromycin, the synthesis consists of six cycles of condensation and reduction (figure 5)

In order to model the genetic modification of the polyketide synthase enzyme system (PKS), control of reaction sequences was developed. The basic Diversify method was extended to include reaction sequence input (table 2). Reaction sequence input allows the user to control which reactions can be carried out at each step of the synthesis. In the degenerate form, the input simply specifies the synthetic scheme (e.g. - the series of reactions used to generate the desired product). Thus to generate a single compound, such as 6-deB, the erythromycin precursor, a reaction sequence with only one reaction in each step of the sequence can be used. In the more complex form, the sequence input can specify



6-deB
Start: Propyl
1. Condensation Down
2. KetoReductase Down
3. Condenstion Up
4. KetoReductase Up
5. Condensation Down
6. No Reduction
7. Condensation Down
8. Enoyl Reduction
9. Condensation Up
10. KetoReductase Up
11. Condensation Up
12. KetoReductase Up
13. Transesterification

Figure 5: 6-deB Biosynthesis. Polymeric biosynthesis of the 6deB erythromycin core is shown. 6-deB is generated by 6 cycles of condensation, each followed by a variable degree of beta-keto reduction. Chain grown is terminated by a transesterification ring closure which forms the macrolactone intermediate.

Level	Transforms Applied At Each Level				
0	pconup[a]	pcondn[b]			
1	krup[c]	krdn[d]			
2	pconup	pcondn			
3	nored[e]	krup	krdn	krdhtrans[f]	krdher[g]
4	pconup	pcondn			
5	nored				
6	pconup	pcondn			
7	nored	krup	krdn	krdhtrans	krdher
8	pconup				
9	krup	krdn			
10	pconup	pcondn			
11	krup	krdn			
12	te[h]				

Table 2: Reaction Sequence Control. The transforms to be exhaustively applied at each step in the synthetic sequence are listed by their step in the sequence and the transform name. There is no limit on either the number of transforms in any single step or the total number of synthetic steps. This reaction sequence was applied to the SNAC precursors in order to generate the complete library. Footnotes: [a,b]methylmalonyl condensation (two stereochemistries), [c,d]ketoreduction (two stereochemistries), [e]no reduction (ketone preserved), [f]ketoreduction followed by dehydration (trans isomer), [g]ketoreduction and dehydration followed by enoyl reduction (leaving a methylene), [h]transesterification.

multiple possible reactions at each step which can all be applied to generate a library of products. For instance, in order to close the macrolactone ring, the initial reduction must leave a hydroxyl group to form the lactone, so in the first round of reduction, only the two ketoreduction transforms were applied while at other steps in the sequence all five possible

reduction transforms were applied. The reaction sequence can contain as many reactions at each step as necessary, and the reactions at each step are applied exhaustively to the products from the previous reaction step to generate a "reaction-based" combinatorial library.

Results

Optimization Library

Selection of synthetically and medicinally favorable aldehyde reagents for the isatin derivative library (vida supra) from the ACD 97.2 yielded 761 potential reagents. Eleven of these compounds were eliminated because of unanticipated unusual functional groups or difficultly generating three dimensional structures with CONCORD (*26*). The library was segregated into 721 neutral compounds and 29 charged compounds. We chose to focus on the neutral compounds because of the difficulty of properly representing atomic charges in this highly charged active site.

Although each compound started with the isatin anchor fragment overlaid on the crystallographic position of the guanine, as the variable portion of each molecule grew, significant shifting of the isatin group could take place. A variety of acceptable binding modes were seen, however, very few formed the exceptional hydrogen bonds which could be anticipated by the topological similarity between the hydrogen bonding patterns of isatin and the crystallographic guanine. The approximation made in treating the receptor as rigid may play a large part in this failure to observe excellent hydrogen bonding properties since it fails to allow the protein to adjust to optimize these potential bonding interactions. Furthermore, it is known that the distant dependent dielectric approximation under-

17

ciso

ciso

emphasized hydrogen-bonds (Keith Burdick, personal communication). Despite the common anchor throughout the library, the molecules explored much of the remaining binding pocket, particularly in the region of the phosphate binding pocket.

# Rotatable Bonds	Number of Examples	Mean $T\Delta S_{conf}$ (Kcal/M)	standard deviation
2	290	1.21	0.37
3	132	1.61	0.60
4	135	1.81	0.67
5	79	2.42	0.79
6	35	2.54	0.80
7	14	2.42	0.67
8	7	2.84	1.03
Total	692	1.65	0.74

Table 3: Conformational Entropy Correction. Accumulated statistical data for calculation of the conformational entropy penalty by the state counting method. This data is molecules from the isatin optimization library.

Compounds in the library contained from two to twelve rotatable bonds resulting in calculated conformational entropy penalties ranged from 0.42 Kcal/M to about 5 Kcal/M. The calculated entropy penalties did not grow linearly with the number of rotatable bonds as is presumed by many models (table 3). The penalty grew approximately linear over a range of 2-5 rotatable bonds, but for 5 or more rotatable bonds, the mean conformational entropy penalties were approximately constant. However, there was a very high variance of entropy penalties for each set of compounds with the same number of rotatable bonds. The difference in entropy penalties for the molecules was large compared to the total scores of the compounds, and resulted in significant changes in the library rankings,

particularly amoung the top 30 compounds. For additional details about the conformational entropy correction, please refer to the appendix.

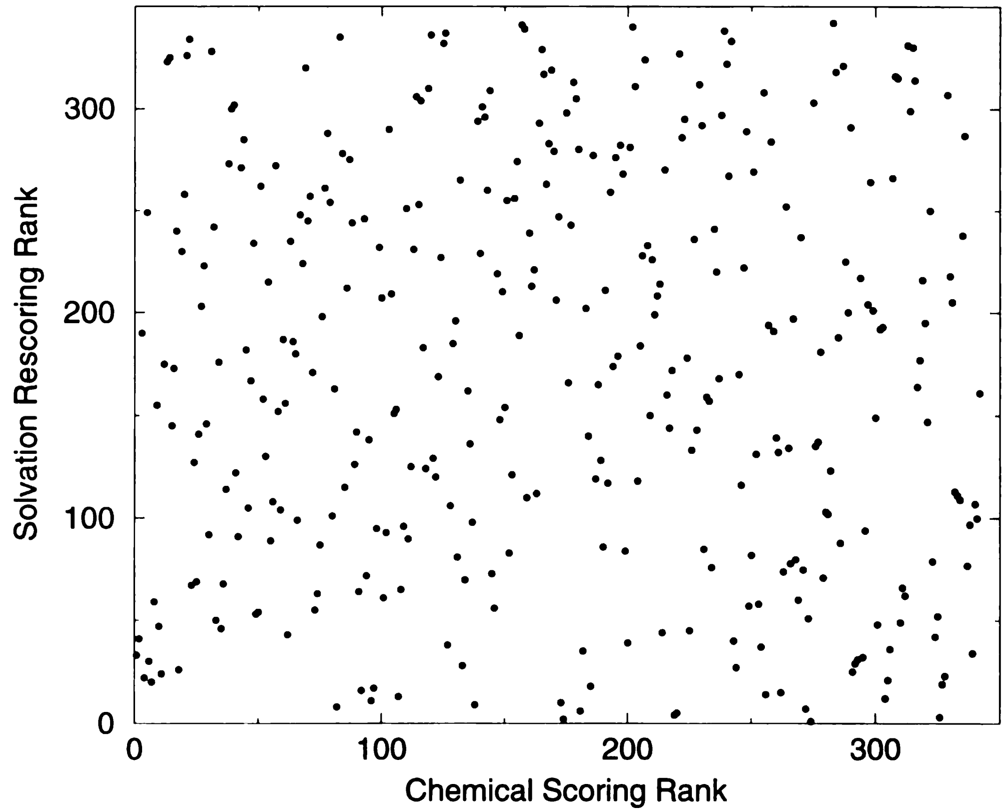


Figure 6: Solvation Rescore. Rank of 342 poses of the inhibitor **17** (chapter 6) docked to HGXPRTase with chemical scoring compared to the rank when the poses are rescored and minimized with GBSA scoring.

The GBSA method is too computationally expensive to use as a primary scoring function. Instead, single point rescore calculations of receptor-ligand poses generated with surrogate scoring functions are used. To determine whether chemical scoring or force-field scoring provided a better surrogate for solvation scoring, and thus produced better structures for solvation rescoring, we examined the known HGXPRTase inhibitor, compound **17**, in detail. Figures 6 and 7 show a comparison of the chemical (figure 6) and force-field (figure 7) scores versus GBSA rescores for all of the conformations generated by the

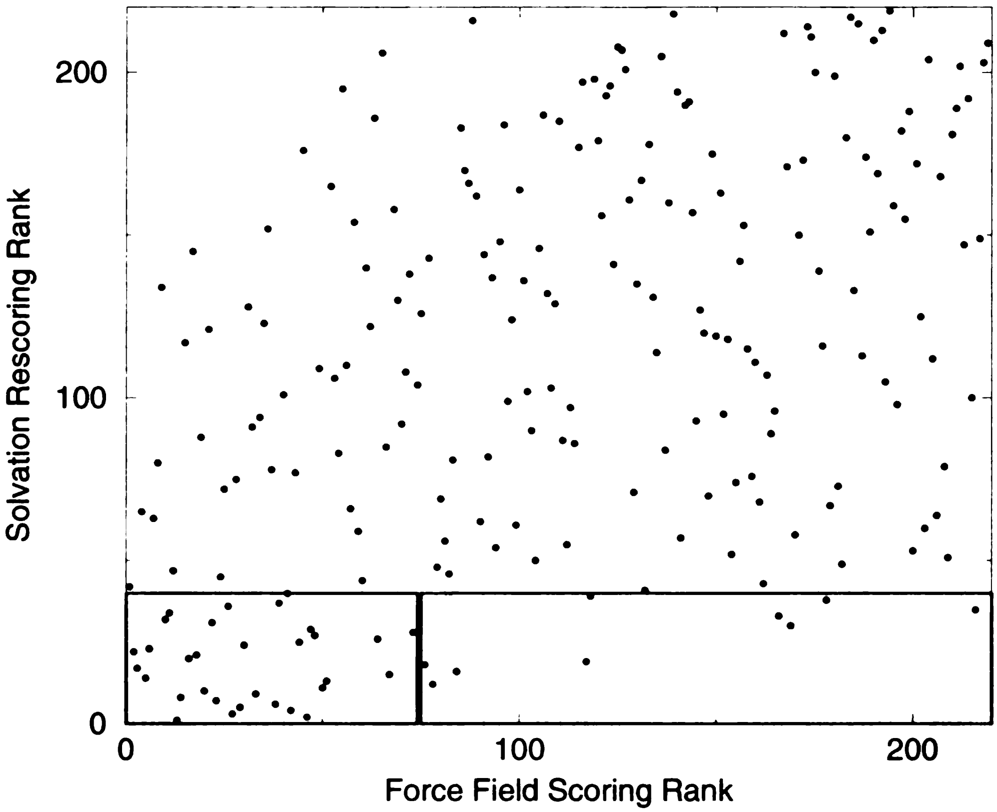


Figure 7: Solvation Rescore. Rank of 219 poses of the inhibitor **17** (chapter 6) docked to HGXPRTase with force-field scoring compared to the rank when the poses are rescored and minimized with GBSA scoring. Note the significant difference in density of points in the red box versus the blue box. This indicates that, in contrast to chemical scoring, rescoring and minimizing the 75 best force-field poses with GBSA identifies nearly all of the best GBSA poses.

incremental growth docking with very high sampling. Figure 6 clearly shows there is no correlation at all between chemical scoring and GBSA scoring in this system. In contrast, rescoring and reminimizing the seventy-five best poses from force-field scoring generates nearly all of the best scoring GBSA poses (figure 7). Of the four known inhibitors which were in the library, only one, compound **10** from chapter 6, was found in the top ten percent of GBSA ranked compounds in the initial search. However, when all possible poses of the ligands were rescored with GBSA rather than simply the single best force-field

pose, the best GBSA pose identified had a much better score (they make up four of the top six compounds). Unfortunately, it is not computationally feasible to test if the rest of the library would benefit an equal amount from a similar detailed computation.

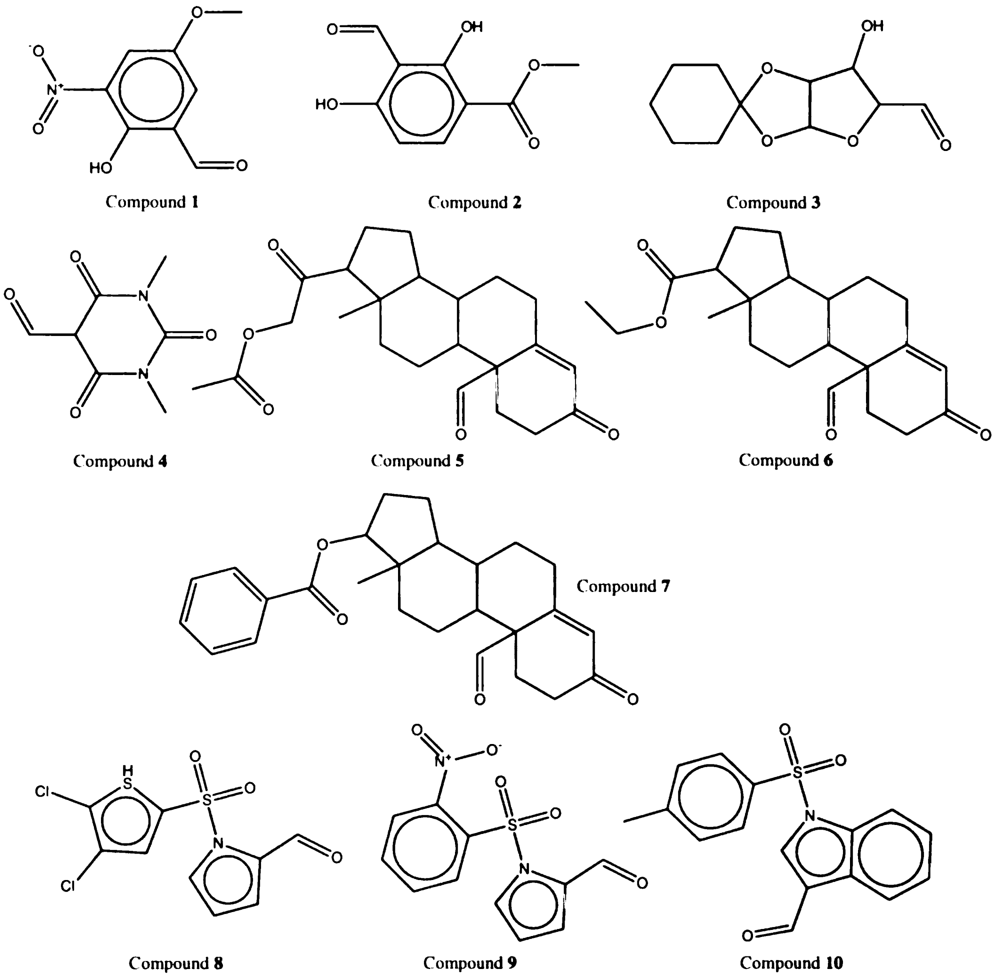


Figure 8: Optimization Library Hits. Ten aldehyde reagents identified by screening the isatin-imine product library against HGXPRTase. Compounds **1** through **7** score well under chemical and force-field scoring. In addition, Compound **5**, **6**, and **7** have good GBSA scores. Compounds **8**, **9**, and **10** are chemically interesting, and could lead to a second optimization library.

Several potential classes of inhibitors were identified with force-field, chemical, and GBSA scoring functions, each corrected for conformational entropy. A variety of 2,5

case

case

disubstituted benzaldehydes, similar to compound **10** from chapter 6 scored well. These are promising candidates because in addition to having good scores, they have a similar topology to a known inhibitor. However, a similar result might have been obtained by a computationally less intensive topological similarity search. Seven compounds appear among the top 30 compounds of both the chemical scoring and force-field scoring lists (figure 8). Two of these compounds are 2,3,5 tri-substituted benzaldehydes, conservative analogs of the lead compounds (**1** and **2** of figure 8). An unusual bicyclic compound with a spiro-cyclohexane moiety is also on the hitlist (**3** of figure 8). One biolobigically interesting compound is an 1,3-dimethyl-2,4,6-trioxohexahydropyrimidine-5-dimethylcarbaldehyde (**4** of figure 8). The three remaining compounds are four-cycle non-aromatic hormone derivatives (**5**, **6**, and **7** of figure 8). The molecular weights of these compounds are between 500 and 550, which is slightly larger than optimal, but still acceptable for further exploration at this early stage of the development cycle. Not only do these final three compounds occur on both the force-field and chemical scoring lists, but they are also in the top 30 compounds reranked by GBSA solvation rescoring. The most chemically interesting compounds are a set of sulfonamide-linked two ring aldehyde derivatives (**8**, **9**, and **10** of figure 8). These may pick up additional hydrophobic interactions, and they are also favorable because the sulfonamide linkage offers a facile route to exploration of a library in this additional pocket. In any future synthesis, these compounds should receive first priority.

Polyketide Results. Ten transforms were generated including the transesterification ring closure to encode the basic macrolactone polyketide synthesis (table 4). Six additional transforms were created to mimic polyketide auxiliary enzymes (e.g. methylations

Transform Name[a]	Reactive Atom Set	Product Atom Set[b]
pconup	NSC(=O)	XXC1(=O).NSC(=O)[C@@H]1C
pcondn	NSC(=O)	XXC1(=O).NSC(=O)[C@H]1C
pconshort	NSC(=O)	XXC1(=O).NSC(=O)C1
nored	NSC(=O)CC(=O)C	NSC(=O)**(=*)*
krup	NSC(=O)CC(=O)C	NSC(=O)*[C@H](O)*
krdn	NSC(=O)CC(=O)C	NSC(=O)*[C@@H](O)*
krdhtrans	NSC(=O)CC(=O)C	NSC(=O)/C=C(X)/*
krdher	NSC(=O)CC(=O)C	NSC(=O)*C(X)*
te	NSC(=O)************[O;D1]	XXC1(=O)************O1
6-hydroxyl	OC(=O)****[C@@H]([C;D1])*	OC(=O)****[C@@]1(*)*.O1
12,10-hydroxyl	O=CO*[C@H]([C;D1])*	O=CO*[C@]1(*)*.O1

Table 4: Polyketide Synthase Transforms. The three condensation transforms, five reduction transforms, and three accessory transforms necessary to model basic macrolactone biosynthesis are presented. Each of these replacement transforms identifies the inclusion (reactive)atoms with a SMARTS string. These reactive atoms are mapped 1:1 with the product atoms indicated in the product set SMILES strings. Footnotes: [a]see figure 8 for description of transform names, [b]X is a non-standard SMILES lexicon indicating destruction of the associated reactive atom.

Transform name	Reactive atom set	Product atom set
generate snac	O=C1OC(C)(C)OC(=O)C1	O=C1OX(X)(X)S2C(=O)C1.C2CNC(=O)C
load enzyme	O=C(O)CC(=O)SCCNC(=O)C	X=X(X)*C(=O)SNXXX(=X)X

Table 5: SNAC Synthesis Transforms. Following the synthesis of used by Pohl and coworkers, an initial transform was created which generated the thiaminoacetyl precursor (snac transform). A second transform which modeled loading of the SNAC precursor onto the PKS via a thioester bond, with N representing the "eNzyme" was also generated. Together these transforms prime the PKS for library generation with the meldrum acid analogs.

and glycosylations) and metabolite formation (e.g. enolether formation, spiroketal formation) which are important for biological activity (*32*). Two final transforms were generated

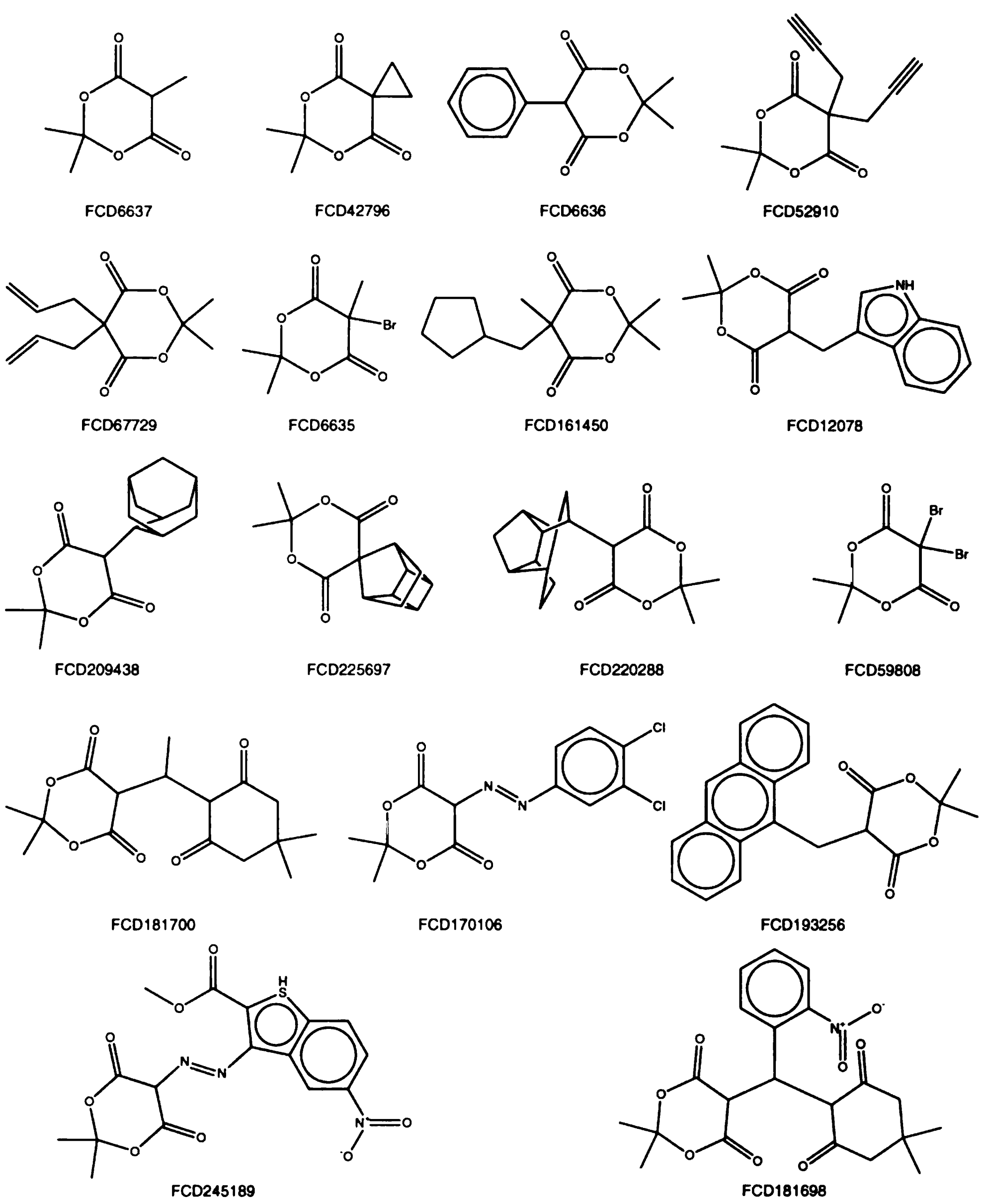
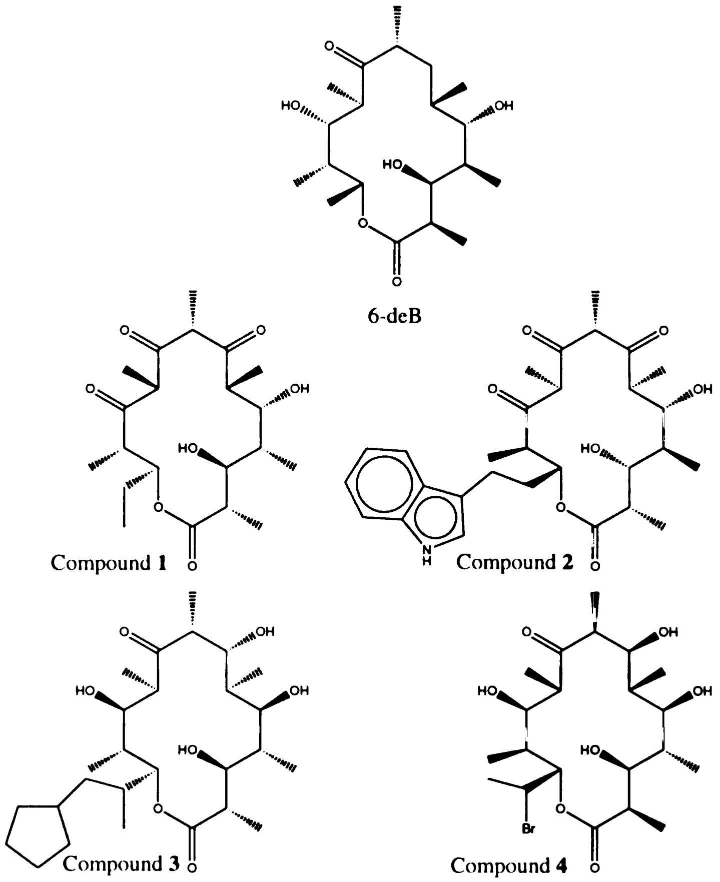


Figure 9: Meldrum's Acid Analogs. These seventeen reagents are commercially available and can be converted to the corresponding SNAC derivatives for incorporation into polyketides. The extent of variability which can be tolerated by the PKS machinery is broad, but has not yet been completely explored. Some of these compounds, particularly the larger derivatives, may not be properly processed by the PKS.

to prepare the SNAC precursors from available reagents and load them on the enzyme for macrolactone generation (table 5) (*31*).



6-deB

Compound 1

Compound 2

Compound 3

Compound 4

Figure 10: Macrolactone Products. Four example products selected at random from the macrolactone library are shown along with 6-deB, the related erythromycin precursor. Compound **1** is a derivative or FCD6637, compound **2** is a derivative of FCD12078, compound **3** is a derivative of FCD161450, and compound **4** is a derivative of FCD6635.

The UC_Select search for appropriate SNAC starting materials from the ACD v98.2 identified seventeen Meldrum acid analogs (figure 9)(*31*). These were sucessfully converted to macrolactone precursors using a two step linear synthetic and enzyme loading scheme which follows the method of Pohl et. al. These precursors were taken forward as a group to generate one large SNAC library containing 293,607 semi-synthetic virtual

polyketides. This library contained 69,632 macrolactone erythromycin analogs (figure 10) while the remaining 223,975 were linear derivatives.

Discussion

There are several points which warrant further discussion. We organize the discussion into segments pertaining to the optimization library and the polyketide library. First, we demonstrate that when using the GBSA method to rescore molecular poses generated with force field scoring, tens of poses are necessary to assure that the best GBSA scores are found. Next, we briefly consider the effects of the entropy correction on library design. Finally we discuss the versatility of Diversify, exemplified by modeling of the semi-synthetic polyketide synthesis.

Optimization Library. Figures 6 and 7 show conclusively that one should not use chemical scoring in an attempt to enrich good solvation scoring conformations in this system. By contrast, there appears to be some weak correlation between force-field scoring and solvation scoring. The best fifty conformations from force-field have much better solvation scores than any other set of force-field conformations. This demonstrates that, at least for this system, using the top conformations from force-field scoring will generate some of the best solvation scores.

Application of the conformational entropy penalty resulted in significant shifting of the ranking of compounds. In particular, with torsional minimization, more flexible compounds can be found disproportionately represented among the best scoring compounds. Although application of an entropy correction term moved many of these compounds out of the top 4 percent, some highly flexible compounds remained among the very be

formyl

with a

pound

correc

with

note

libr

ber

des

hy

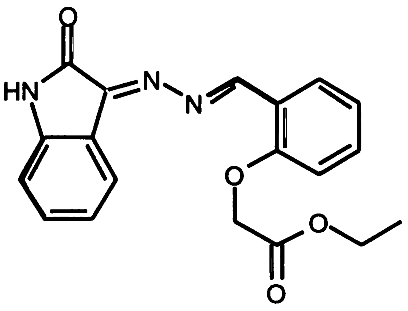
en

th

be

very best compounds. For instance, even after the entropy correction, ethyl 2-(2-formylphenoxy)acetate (MFCD00052131, *vida infra*) was the third ranked compound, with a *corrected* force-field score of -38.21 Kcal/M. The entropy penalty for this compound was calculated to be 2.86 Kcal/M. For a more detailed discussion of the entropy correction method, please refer to the chapter appendix.

MFCD00052131



UC_Select was used to generate a library which had physical properties consistent with the "rule of five" criteria for good pharmacokinetic profiles. Despite this, it must be noted that the hydrazone functional group is not very medicinally desirable. An alternative library, which can be explored in a similar manner are the products of a Wittig reaction between the isatin and cinnamoyl derivatives. This library would have the much more desirable olefin moieties to replace the hydrazone. However, we chose to explore the hydrazone library first because no olefin derivatives have been synthesized or assayed.

Polyketide Library. We first sought to address the question of whether complex enzymatic biosynthesis machinery such as the PKS could be described in the context of the Diversify chemical engine. Although there are fragment based methods which could be used to generate a macrolactone library (*33*), the elegance of the Diversify solution is ı7

ıng

cıgı

satisfying. By encoding only ten transforms (table 4), we were able to capture all of macrolactone synthetic machinery in a manner which can easily incorporate expanding knowledge about the selectivity of individual enzymes as well as allow incorporation of novel staring materials of final decorations by accessory enzymes.

Further, the reaction scheme control structure allows streamlined expression of the genetic variability in a format which parallels the organization of the biological system and allows specification and creation of either individual molecules (figure 5) or libraries of macrolactones (table 2). Diversify is a general chemical engine which can be applied to complex enzymatic systems as well as organic synthesis.

The ability to generate libraries of macrolactones raises the question of the nature and content of these libraries. Perhaps the most obvious result is that if all partial products are considered, the vast majority of polyketides in the virtual library are not the cyclized macrolactones generally considered, but acyclic polyketide polymers more reminiscent of other biopolymers such as peptides and fatty acids or synthetic polymers such as peptoids (*34*). Whether these agents can be developed into drug-like molecules will depend largely on whether PKSs can be identified which allow sufficiently large derivatives of malonyl-CoA to be incorporated.

The reaction sequence method mimics not only the natural synthesis, but each transform can be associated with individual genes in the polyketide synthase gene, so when genes are shuffled, it can easily be modeled by shuffling the corresponding transforms in the reaction sequence. This parallel between the molecular-biology, chemistry, and the library generation, makes sharing data between the fields simple and allows easier interpretation of the results.

This SNAC virtual library demonstrates the power of genetically designed macrolactone libraries. Although the SNAC library itself is quite large, there were only seventeen SNAC reagent precursors available in the ACD. One solution is to synthesize a single SNAC which contains a functional group that can be derivatized in a combinatorial reaction either as a solid or liquid phase library synthesis. Although this chemical modification would have the potential to greatly increase the numbers of compounds and diversity at this single position, one should not overlook the variety of functionality which is modeled and can be displayed all around the macrolactone ring using the genetic engineering, particularly when glycosylation or secondary cyclizations are considered.

General Discussion. Generation of virtual libraries from available reagents is one of the important bottlenecks in the computational design of combinatorial libraries. For an individual reaction, it is feasible to generate a virtual library using simple scripts, however, for frequent construction of libraries, a more general, less error-prone method is beneficial. The method presented here follows the logical sequence of the chemist's synthetic scheme to generate virtual libraries. This approach has several advantages: first, the method can be understood and used by the synthetic chemist; second, the method leads to a natural transfer of synthetic information and constraints from a description of the synthesis to the virtual library construction; third, because Diversify is designed to model any chemical reaction, it can handle chemical subtleties, such as stereochemistry, that pose difficult special cases for library construction scripts; and finally, since Diversify is a complied C program which uses the Daylight substructure searching utilities, it is extremely efficient at constructing virtual libraries.

I

compon

may be

ular syn

amines

reagent

explore

of reag

out bef

can he

has alr

protei

know

genet

ing a

comp

tions

enzy

lies

Co

co

In combinatorial synthesis, it sometimes becomes advantageous to synthesize the component pieces from related precursors. In later stages of compound optimization, this may be easier than preforming modifications on the final product. For instance, in a particular synthesis, a side-chain may be attached as an amine reagent. However, if available amines do not provide the desired side-chain, one may wonder if synthesis of the amine reagent from a primary halide may be beneficial. Diversify and UC_Select can be used to explore these reagent synthesis problems, and to help the chemist determine which route of reagent synthesis may yield the most interesting products. This analysis can be carried out before the chemist develops any reagent synthesis protocols. In this manner, Diversify can help assess the best ways to expand the potential of a combinatorial synthesis which has already been worked out.

Many proteins carry out chemical reactions. One method to compare and catalog proteins is by examining the chemical reactions which they can carry out. The universe of known proteins has been cataloged and compared along many lines including, underlying genetic sequence, primary amino-acid sequence, structural motifs, and function. By creating a system which can systematically describe chemical reactions, we can encode and compare chemical reactions carried out by families of enzymes and use these same reactions as a metric to catalog proteins. Using Diversify's reaction descriptions to categorize enzyme families may lead to new insights in the relations between protein structural families and their related protein functional families.

Conclusions

A chemical engine has been developed which simulates synthetic chemistry by computationally manipulating molecules. The critical resources of a chemical engine are

molecules and transforms. A simple method to identify appropriate, available reagents for this chemical engine has been developed. Transform generation is flexible enough to allow extension of the transforms to enzymatic reactions as well as general organic reactions. The combination of appropriate library construction and structure-based screening allows the selection of binding hypothesis which lead to strong inference(*35*) and thus rapid advancement of molecular design projects.

Acknowledgments

We are grateful to Mike Siani, Connie Oshiro, and Donna Hendrix for helpful discussions. We thank Meg Stauber, Ellen Kick, and Karl Maurer for discussions of synthetic chemistry. We would like to thank Jeremy Yang and Jack Delany of Daylight Chemical Informations Systems Incorporated for help with the Daylight Toolkits. This work was supported by the Daiichi Corporation, the Pfizer Corporation, and an NIH training grant to AGS.

Appendix: Conformational Entropy

As with any structure-based design method, the results from DOCK are generally limited by the scoring function used to evaluate ligand-macromolecule binding. The field of binding free energy evaluation has many unanswered questions and many approximate solutions exist (*36*). These solutions, or scoring functions, fall into two categories, empirical estimates fitted to structural data and fundamental estimates (*36*). The empiric functions usually contain all of the terms pertinent to the Gibbs free energy, but are limited by their training data, while fundamental functions are more general, but may wholly neglect important terms in the free energy, such as desolvation or entropy. Recently, application of generalized Born theory as well as Poisson-Boltzman theory to molecular docking has incorporated many of the solvation and entropy terms formally into a fundamental scoring function(*30*)(*37*). One prominent term still missing from this scoring function is conformational entropy. We briefly explore a simplistic state counting model for calculating conformational entropy and compare it to experiment and to empirical scoring functions before applying it to the isatin lead optimization library (see main text).

Entropy Correction Method. The standard methods of scoring were corrected with a conformational entropy term. Because the size of the library is limited to about 700 compounds by the availability of compatible aldehyde reagents, more detailed score evaluation is feasible. Neither the molecular mechanics scoring function nor the empirical scoring function in DOCK take account of the loss of conformational entropy upon binding of the ligand to the receptor. This conformational entropy term is often approximated as a linear function of the number of rotatable bonds. Here we chose to calculate conformational entropy by simplistically enumerating and counting energetically accessible conformations in vacuum. We used DOCK to enumerate all available torsion positions of all

flexible bonds and generate an intramolecular score for each. The minimum scoring conformation was found for each molecule, and all conformations which lay within 10Kcal/M internal energy were counted. This total number, W, was assumed to be an approximation of the number of states the molecule could assume in solution. From well known thermodynamic equations, one can derive a simple relationship between the conformational entropy and W, the number of conformations in the free state when we assume that there is only one bound conformation and that $T=310.14^{o}$ K.

$$\Delta G = \Delta H - T\Delta S$$

$$S = R\ln(W)$$

$$T\Delta S = RT\ln\left(\frac{W}{1}\right) = 1.98cal/(MK) \bullet 310.15K \bullet \ln(W)$$

$$T\Delta S = 0.616Kcal/M \bullet \ln(W)$$

Using this relationship, we calculated a conformational entropy loss upon binding for each ligand. This conformational entropy penalty was added to the force-field and chemical score of each molecule and the compounds were ranked by their new scores.

Entropy Correction Results. The conformational entropy calculated for ten n-alkane molecules is plotted against the number of rotatable bonds in figure 11. A linear fit of the data points yielded

$$T\Delta S_{\mathrm{conf}}\,(\mathrm{Kcal/M}) = 0.48 * N_{\mathrm{rot}} + 0.27$$

with a correlation coefficient of 0.999. The slope of linear fits to three sets of experimental

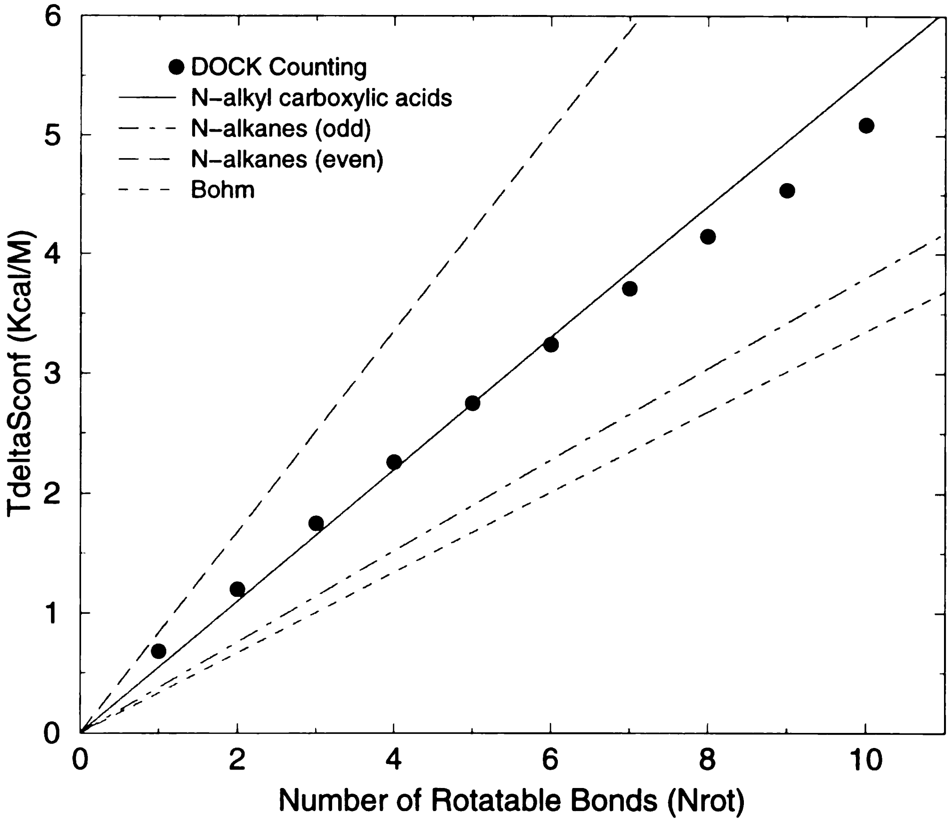


Figure 11: State Counting Validation. Conformational entropy of N-alkanes calculated with the state counting method (black circles) versus number of rotatable bonds. These data are consistent with experimentally derived (entropy of fusion) slopes for odd N-alkanes (dot-dash), even N-alkanes (long dash), and N-alkyl carboxylic acids (solid). The conformational entropy method of Bohm is shown for comparison (short dashes).

entropy of fusion experiments are plotted for comparison (*38*). The data from this study is intermediate between the experimental data for odd and eve n-alkanes and is very close to the data for n-alkyl carboxylic acids. The method of Bohm, which also approximates this n-alkyl data well is shown for comparison.

Conformational entropy of ligand binding calculated by counting states for the library of isatins is presented in table 3 and figure 12. The mean and standard deviation of the conformational entropies are plotted against the number of rotatable bonds. Although

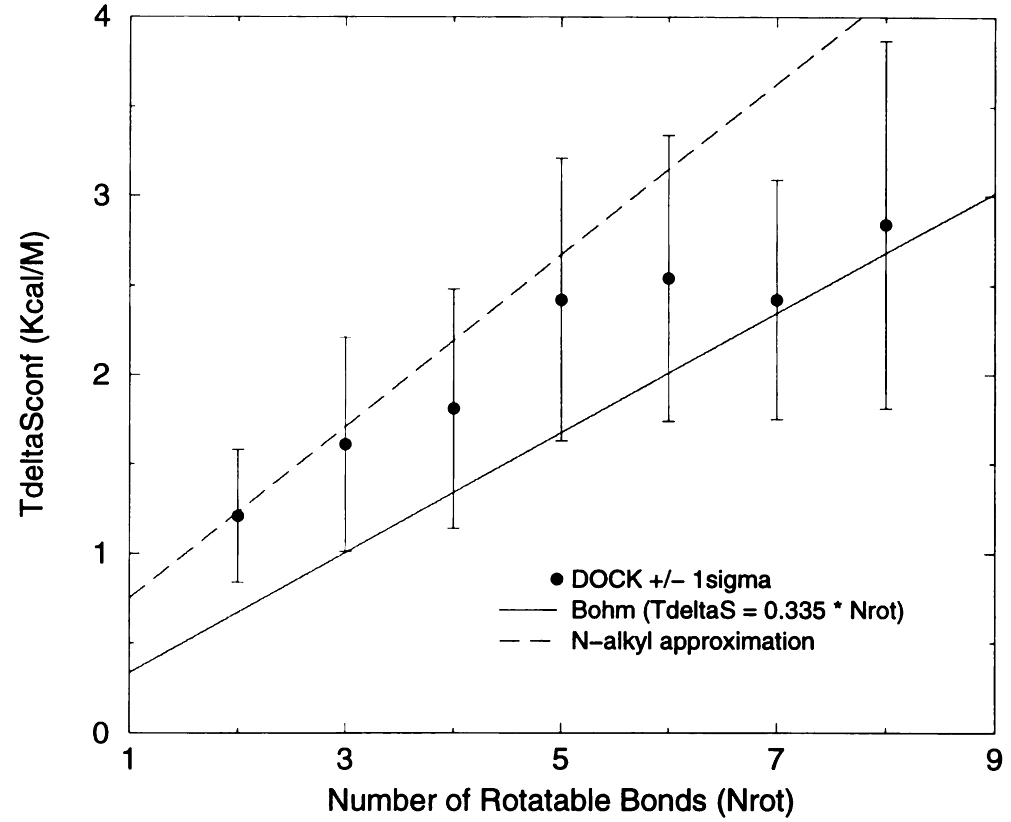


Figure 12: Small Molecule Conformational Entropy. Means and standard deviations of conformational entropy for the isatin library members are plotted against number of rotatable bonds. Note first that the mean entropies fall below the N-alkyl entropy in a non-linear fashion, particularly as the number of rotatable bonds grows. Second, note that the standard deviations are large relative to the differences between the means, indicating that the data cannot be sufficiently described by the single independent variable, N_{rot}, as many approximations attempt. The linear approximation based on the N-alkyl compounds (long dash) and from Bohm's method (solid) are presented for comparison.

the mean values are approximately linear for N_{rot} of 2 to 5, they fall significantly below linear for higher numbers of rotatable bonds. Note, this library's scaffold contained two rotatable bonds, so no examples of one rotatable bond are shown. Two linear models of conformational entropy are shown for contrast. First, the linear fit of the state counting data for n-alkyl carboxylic acids is shown. Second, the linear approximation from the scoring function of Bohm (*39*) is shown. The former fits the relatively rigid molecules

well, but greatly overestimates the more flexible molecules. The latter avoids over estimating the most flexible molecules, but at a cost of greatly underestimating nearly all of the compounds.

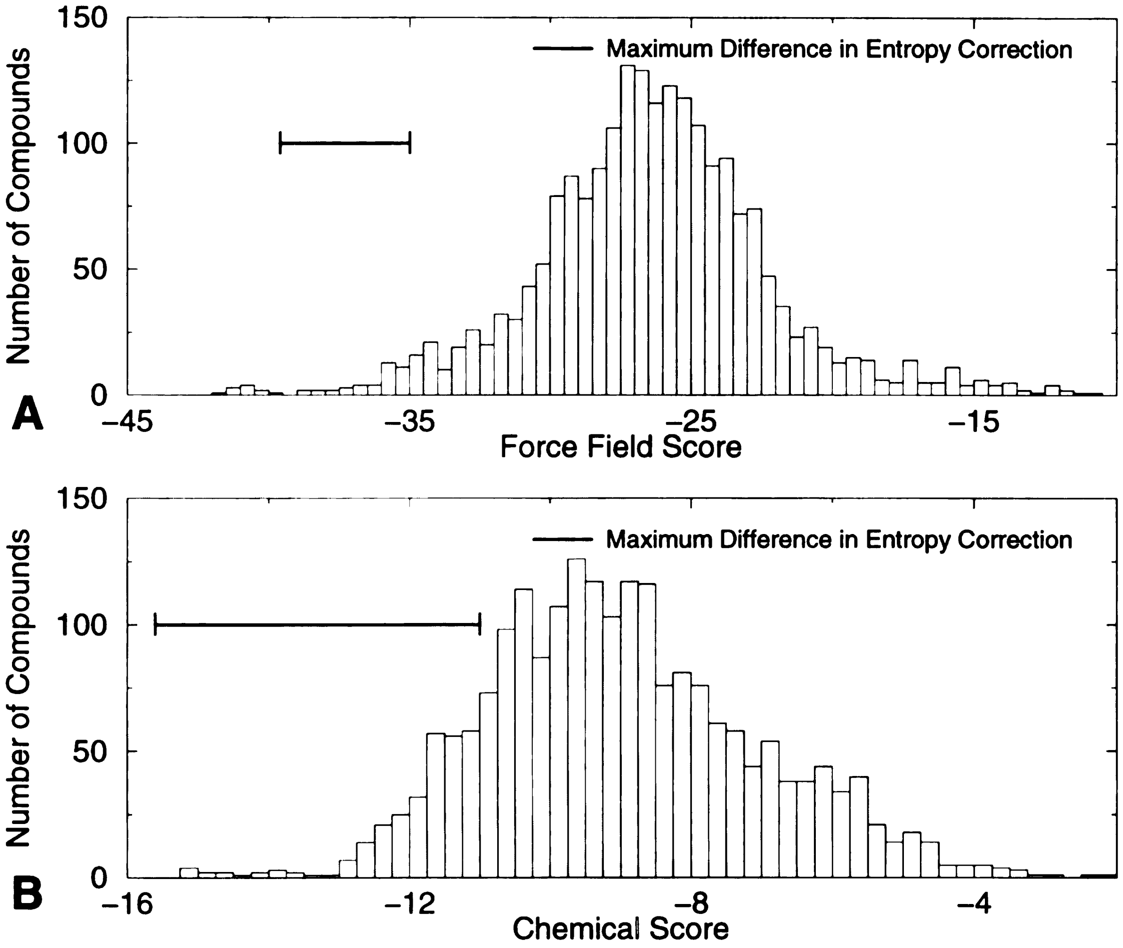


Figure 13: Significant Entropy Correction. The maximum difference in entropy corrections between two compounds is shown (bar) along with the force field (**A**) and chemical (**B**) scoring histograms for the entire isatin library. Entropy corrections have the potential to significantly effect the ranking of compounds in the library using either scoring function.

The relative size of the entropy correction term compared to the force-field and chemical scores is shown in figure 13A and 13B respectively. The range of entropy correc-

tions among the library compounds is 0.42 to 5.02 Kcal/M, so the maximum difference in correction between two molecules is 4.59 Kcal/M. Each figure contains a histogram of the uncorrected scores from the library docking. In addition, a bar indicating the relative size of the maximum difference in entropy correction is illustrated, assuming the units are the same. This bar indicates the maximum possible shift in positions of one molecule in the histogram relative to another. Even in the case of force-field scoring, which greatly overestimated binding energies, the entropy correction factor make a significant contribution to the overall score.

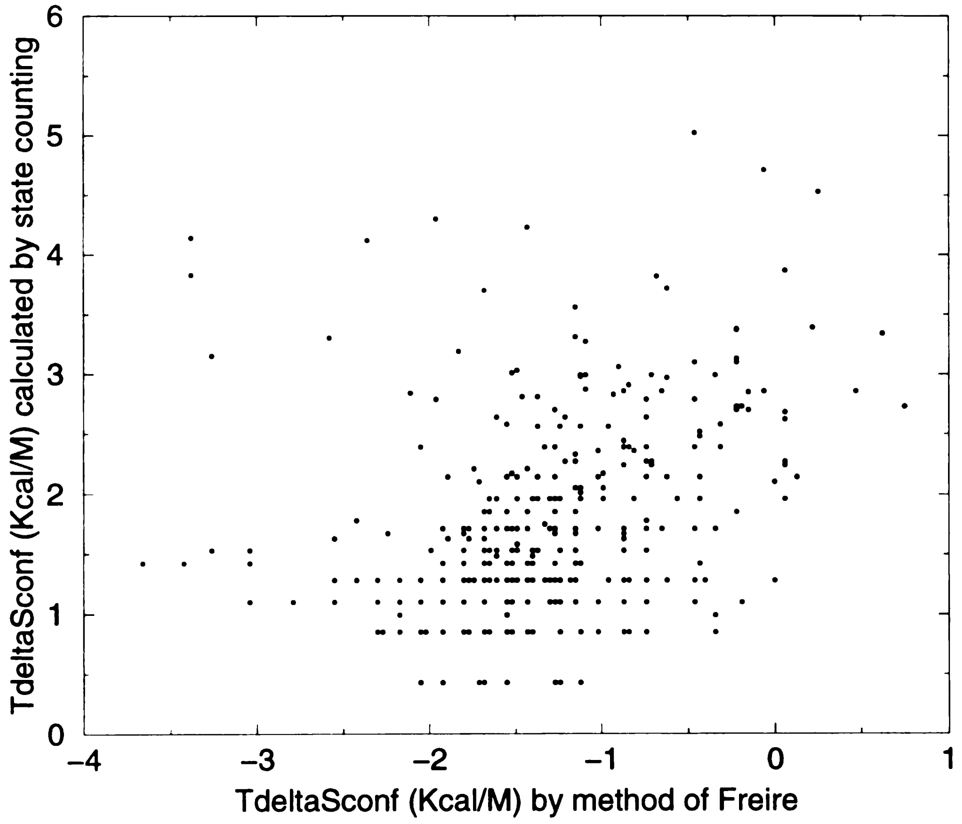


Figure 14: Comparison to Heuristic Method. Entropy correction factors calculated by the state counting method presented here versus the method of Freire. The method of Freire uses the heuristic of "number of heavy atoms" as an approximation of filled volume to make a linear correction to the entropy penalty ($T\Delta S_{conf}$ (Kcal/M) = $0.53 * N_{rot} - 0.124 * N_{heavy}$). There is little correlation between the methods.

In a study of the binding properties of a series of HIV protease inhibitors, Bardi et al. developed a method to estimate conformational entropy which included a term to correct for the occupied volume (*40*). the method subtracted the empiric term 0.124 Kcal/M for each heavy atom in the molecule from the conformational entropy term of 0.528 Kcal/M per rotatable bond. To compare estimates, the conformational entropy for the library of compounds generated by Bardi's method was plotted against the state counting method presented here (figure 14). Little linear correlation can be seen between the two methods as reflected in a correlation coefficient of <0.4 for a linear fit of the data.

Entropy Correction Discussion. Despite a very approximate estimate of the solution phase conformational ensemble of the ligand and the assumption of only one bound state, this model reproduces experimental entropy of n-alkyl systems very well. Both the experimental and the calculated values show the well known linear relationship between conformational entropy and number of rotatable bonds for n-alkanes (figure 11). This result is surprisingly good and serves to validate our extension of the method to more drug-like small molecules. However, when the state counting method is applied to the small molecule library, intramolecular clashes (excluded volume) limit the number of low energy conformations in molecules with many rotatable bonds (figure 12). As is shown by the comparison to the linear approximation from applying the state counting to the n-alkanes or Bohm's approximation, the data is inadequately described by a linear proportionality to the number of rotatable bonds. Furthermore, although the mean values can be well approximated by a quadratic fit of the number of rotatable bonds, the large variance within each set relative to the difference between consecutive means indicates that another independent variable is needed to describe the data. Good estimates of the conformational entropy of ligand binding requires one to take account of the intramolecular conforma-

tional restrictions particular to each molecule.

The free energy of binding is formally $\Delta G = \Delta H - T\Delta S$. Although not formally separable, it is a useful approximation to separate ΔS into multiple terms, one of which is ΔS_{conf}, the entropy cost of restricting the conformational entropy of the ligand upon binding. Since it is generally understood that a bound ligand accesses fewer conformations than when in solution, ΔS_{conf} is always negative and the conformational entropy contributes a "penalty" to the binding energy. Because it contains a natural logarithm of the number of state and because the sign is the same for all compounds it is easy to assume this term will not have a large effect on the ranking of ligands, and indeed, the error in neglecting this term may be small compared to other terms neglected in force-field scoring such as the entropy or enthalpy of solvation. However, recent work on application of GB/SA to the docking problem (*30*) has led us to consider conformation entropy. Figure 13 shows not only that the conformational entropy "penalty" is significant relative to total docking score, but that the difference in conformational penalties between ligands has the potential to dramatically effect the relative ranking of compounds.

In the work of Bardi on HIV protease inhibitors (*40*), the conformational entropy is estimated by a linear combination of the number of rotatable bonds (similar to the method of Bohm) and the number of heavy atoms. The second term, which carries a negative coefficient, is meant to correct for the reduction in conformational space due intramolecular clashes (occupied volume). Although it includes a surrogate term to estimate the constraint on conformational space, the method does not correlate well with the direct state counting method described here (Figure 14).

Both of these methods are a step forward from the estimates linear with the number of rotatable bonds. The Bardi method is much faster, however it is limited by its heu-

ristic for occupied volume, which appears to be dramatically effected by the case under consideration. This is best illustrated by examining the sign of the entropy correction calculated with the Bardi method, which predicts that conformational entropy will favor binding for nearly all of these molecules. While the state counting method is slower, the entropy correction term need only be counted once for a database of compounds since in as much as the approximation of one bound conformation holds, the entropy correction is independent of the target macromolecule.

The state counting method as presented here is limited by its rudimentary generation of a conformational ensemble, however, there are a number of more rigorous intramolecular scoring schemes, particularly those which take account of ligand solvation, which could be used to extend the state counting approach discussed here. Furthermore, the choice of a 10 Kcal/Mol cutoff for counting states is arbitrary, and is made in the context of a very simplistic internal energy. Despite these approximations, this method appears more accurate than those which simply calculate a linear relationship between the conformational entropy and the number of rotatable bonds.

A rigorous method of ensemble construction and state counting has been successfully applied to protein side chain conformations(*41*). However, like our method, attempts to extend the method of Leach to small molecules has been limited by the accuracy of the intramolecular energy and ground state (Brian Shoichet, personal communication). The method we present is fast enough to be practical for databases of compounds, yet reliable enough to reproduce experimental n-alkyl conformational entropy values. A more elegant and thorough approach to constructing the conformational ensemble can only improve on the results obtained here.

Bibl

1.

2.

3.

4.

235-

5.

6.

Che

7.

Jou

8.

Sci

9.

Co

Sta

10

Ku

11

12

13

Ac

14

Na

Bibliography

1. Thompson, L. A., and Ellman, J. A. (1996) *Chemical Reviews 96*, 555-600.

2. Ellman, J. A. (1996) *Accounts Of Chemical Research 29*, 132-143.

3. Dolle, R. E. (1998) *Molecular Diversity 3*, 199-233.

4. Dolle, R. E., and Nelson, K. H. (1999) *Journal Of Combinatorial Chemistry 1*, 235-282.

5. Czarnik, A. W. (1996) *Accounts of Chemical Research 29*, 112-113.

6. Meng, E. C., Shoichet, B. K., and Kuntz, I. D. (1992) *Journal Of Computational Chemistry 13*, 505-524.

7. Kuntz, I. D., Blaney, J. M., Oatley, S. J., Langridge, R., and Ferrin, T. E. (1982) *Journal of Molecular Biology 161*, 269-288.

8. Shoichet, B. K., Stroud, R. M., Santi, D. V., Kuntz, I. D., and Perry, K. M. (1993) *Science 259*, 1445-1450.

9. Ring, C. S., Sun, E., McKerrow, J. H., Lee, G. K., Rosenthal, P. J., Kuntz, I. D., and Cohen, F. E. (1993) *Proceedings Of the National Academy Of Sciences Of the United States Of America 90*, 3583-3587.

10. Bodian, D. L., Yamasaki, R. B., Buswell, R. L., Stearns, J. F., White, J. M., and Kuntz, I. D. (1993) *Biochemistry 32*, 2967-2978.

11. Wipke, W. T., and Howe, W. J. (1977) , ACS Publications, Washington, D. C.

12. Corey, E. J., and Wipke, W. T. (1969) *Science 166*, 178-192.

13. Gasteiger, J., Ihlenfeldt, W. D., Rose, P., and Wanke, R. (1990) *Analytica Chimica Acta 235*, 65-75.

14. Corey, E. J., and Cheng, X. (1989) *The Logic of Chemical Synthesis*, John Wiley, New York, N. Y.

15. Jorgensen, W. L., Laird, E. R., Gushurst, A. J., Fleischer, J. M., Gothe, S. A., Helson, H. E., Paderes, G. D., and Sinclair, S. (1990) *Pure and Applied Chemistry 62*, 1921-1932.

16. Lewis, R. A., and Leach, A. R. (1994) *Journal of Computer-Aided Molecular Design 8*, 467-475.

17. Daylight Chemical Information Systems, I. (1996) , Santa Fe, N. M.

18. Delany, J. (1999) in *Merlin Users Group (MUG99)* (Weininger, D., Ed.), Daylight Chemical Information Systems, Inc., Santa Fe, New Mexico.

19. Polinsky, A., Feinstein, R. D., Shi, S., and Kuki, A. (1996) in *American Chemical Society Conference* (Chaiken, I. D., and Janda, K. D., Eds.) pp 219-232, ACS Symposium Series, American Chemical Society.

20. Somoza, J. R., Chin, M. S., Focia, P. J., Wang, C. C., and Fletterick, R. J. (1996) *Biochemistry 35*, 7032-7040.

21. McDaniel, R., Ebertkhosla, S., Hopwood, D. A., and Khosla, C. (1995) *Nature 375*, 549-554.

22. Alvarez, M. A., Fu, H., Khosla, C., Hopwood, D. A., and Bailey, J. E. (1996) *Nature Biotechnology 14*, 335-338.

23. Pieper, R., Luo, G. L., Cane, D. E., and Khosla, C. (1995) *Nature 378*, 263-266.

24. Weininger, D., Weininger, A., and Weininger, J. L. (1989) *Journal Of Chemical Information and Computer Sciences 29*, 97-101.

25. MDL Information Systems, I. (1995) , 14600 Catalina Street
San Leandro, CA 94577.

26. Rusinko, A., Skell, J. M., Balducci, R., McGarity, C. M., and Pearlman, R. S. (1986) in *192nd American Chemical Society National Meeting*, Anaheim, California.

27.

ıal

28.

Dr

29

30

Sc

3

A

3

(

3

(

27. Cohen, N. C., Blaney, J. M., Humblet, C., Gund, P., and Barry, D. C. (1990) *Journal Of Medicinal Chemistry 33*, 883-894.

28. Lipinski, C. A., Lombardo, F., Dominy, B. W., and Feeney, P. J. (1997) *Advanced Drug Delivery Reviews 23*, 3-25.

29. Gasteiger, J., and Marsili, M. (1980) *Tetrahedron 36*, 3219-3288.

30. Zou, X. Q., Sun, Y. X., and Kuntz, I. D. (1999) *Journal Of the American Chemical Society 121*, 8033-8043.

31. Pohl, N. L., Gokhale, R. S., Cane, D. E., and Khosla, C. (1998) *Journal Of the American Chemical Society 120*, 11206-11207.

32. Lartey, P. A., Faghih, R., Pagano, T., Nellans, H. N., Petersen, A., and Plattner, J. J. (1995) *Journal Of Antibiotics 48*, 730-732.

33. Siani, M. A., Weininger, D., James, C. A., and Blaney, J. M. (1995) *Journal Of Chemical Information and Computer Sciences 35*, 1026-1033.

34. Simon, R. J., Kania, R. S., Zuckermann, R. N., Huebner, V. D., Jewell, D. A., Banville, S., Ng, S., Wang, L., Rosenberg, S., Marlowe, C. K., Spellmeyer, D. C., Tan, R. Y., Frankel, A. D., Santi, D. V., Cohen, F. E., and Bartlett, P. A. (1992) *Proceedings Of the National Academy Of Sciences Of the United States Of America 89*, 9367-9371.

35. Platt, J. R. (1964) *Science 146*, 347-353.

36. Tame, J. R. H. (1999) *Journal Of Computer-Aided Molecular Design 13*, 99-108.

37. Shoichet, B. K., Leach, A. R., and Kuntz, I. D. (1999) *Proteins: Structure, Function, and Genetics 34*, 4-16.

38. Searle, M. S., and Williams, D. H. (1992) *Journal Of the American Chemical Society 114*, 10690-10697.

39. Bohm, H. J. (1994) *Journal Of Computer-Aided Molecular Design 8*, 243-256.

40. Bardi, J. S., Luque, I., and Freire, E. (1997) *Biochemistry 36*, 6588-6596.

41. Leach, A. R., and Lemon, A. P. (1998) *Proteins: Structure, Function, and Genetics 33*, 227-239.

Prologue to Chapter 5

Some projects are bigger than any individual investigator, and the "RT project" (HIV-1 Reverse Transcriptase) presented in this chapter is one of them. This project's inception dates to when I first began working in the Kuntz lab, nearly two years before I started graduate school. Diana Roe did a large amount of initial work examining the RT structure and identifying the binding sites described in this chapter. Todd Ewing and I did many of the screening runs, using very early versions of Dock 4.0 (years before flexible docking was implemented). After the initial inhibitor was discovered, Meg Stauber and Karl Maurer, two post-docs from the Kenyon lab, spent virtually their entire post-doc careers discovering just how hard Carbonyl-J was to work with. Eventually, Karl and I discovered the potent analog, Calcamine Orange, and Karl was able to design and synthesize many additional analogs. Despite identifying many very potent compounds, this project has been frustrated for over four years by a lack of profitable structural data for any RT-inhibitor complex. As an alternative, we turned to a variety of physical and biological methods to investigate the mechanism of this class of RT inhibitors. We have used a combination of RNaseH inhibition, RT mutation studies, and Biacore DNA/RT binding to show these compounds inhibit RT by a novel, but unknown, mechanism. All of these studies were carried by out by our collaborators in the laboratories of Steve Hughes and Robert Fisher. Finally, I want to mention that the compounds identified as inhibitors here are reflective of the era in which they were identified. There has been a significant emphasis in the past five years of the importance of identifying "drug-like" molecules early in the design process. Although "drug-like" is a very elusive adjective, the compounds identified here probably do not fit into that category (however amorphous). This highlights a funda-

mental dilemma. On one hand, by not selecting drug-like molecules early, we have discovered a series of potent HIV-1 RT inhibitors which act by a novel mechanism but probably can't be developed into medicinally useful analogs. On the other hand, if we had taken a chance at being more selective early on, we might have not identified any inhibitors. The optimal position for this "selectivity pendulum" remains unclear.

A

J

N

Chapter 5

A Novel Mechanism for Inhibition of HIV-1 Reverse Transcriptase

by

A. Geoffrey Skillman, Karl W. Maurer, Diana C. Roe, Margaret J. Stauber, Dolan Eargle, Todd J. A. Ewing, Angelika Muscate, Maxine V. Medaglia, Robert J. Fisher, Edward Arnold, Hong-Qiang Gao, Robert Buckheit, Paul L. Boyer, Stephen H. Hughes, Irwin D. Kuntz, and George L. Kenyon

Submitted in October 1999 for publication in Biochemistry.

Abstract

The Human Immunodeficiency Virus (HIV) epidemic is an important medical problem. Although combination drug regimens have produced dramatic decreases in viral load, current therapies do not provide a cure for HIV infection. We have used structure-based design and combinatorial medicinal chemistry to identify potent and selective HIV-1 reverse transcriptase (RT) inhibitors that work by a mechanism distinct from that of current HIV drugs. The best of these compounds (compound **4**, 2 naphthalenesulfonic acid, 4-hydroxy-7-[[[[5-hydroxy-6-[(4-cinnamylphenyl)azo]-7-sulfo-2-naphthalenyl]amino]carbonyl]amino]-3-[(4-cinnamylphenyl)azo], disodium salt) has an IC_{50} of 90nM for inhibition of polymerase chain extension, an K_d of 40nM for inhibition of DNA-RT binding, and an IC_{50} of 25-100nM for inhibition of RNaseH cleavage. The parent compound (**1**) was as effective against ten nucleoside and non-nucleoside resistant HIV-1 RT mutants as it was against the wild-type enzyme. Compound **4** inhibited HIV-1 RT and Murine Leukemia Virus (MLV) RT, but it did not inhibit T_4 DNA polymerase, T_7 DNA polymerase, or the Klenow fragment at concentrations up to 200nM. Finally, compound **4** protected cells from HIV-1 infection at a concentration more than 40 times lower than the concentration at which it caused cellular toxicity.

Introduction

Despite a worldwide effort, Human Immunodeficiency Virus (HIV) infection and the subsequent Acquired Immunodeficiency Syndrome (AIDS) remain important medical problems. Although the recent dramatic drop in the AIDS mortality rate and opportunistic infections in the United States is a tribute to new drug therapies(1), the number of people infected by HIV continues to increase(2). Current treatment of HIV infection in the United States commonly involves three drugs: two nucleoside reverse transcriptase inhibitors (NRTIs) and a protease inhibitor (PI)(3). These combination regimens can dramatically decrease viral load(3); however, despite these improvements, replication-competent HIV can still be recovered from patients after 2 years of successful therapy(4).

Resistance has limited the prolonged efficacy of all HIV drugs so far developed. HIV replication generates on average approximately one mutation each time the genome is copied(5). This genomic diversity makes development of resistant strains almost certain under conditions of selective (drug) pressure, high viral load, and rapid replication. Hecht and co-workers recently reported the transmission of an HIV strain which was resistant to four protease inhibitors as well as two reverse transcriptase inhibitors(6). The emergence of multi-drug-resistant strains of HIV demonstrates the need for the development of new drugs whose mechanism of action is distinct from those of known drugs.

Reverse transcriptase, an essential enzyme for HIV replication(7), has two enzymatic functions, the polymerase (Pol) and the ribonuclease H (RNaseH). In the normal viral replication cycle, RT converts viral genomic RNA into a double stranded linear DNA. The success of both NRTIs and non-nucleoside reverse transcriptase inhibitors (NNRTIs) in the treatment of HIV infection demonstrates that HIV-1 RT is a valid drug

17

ase

ase

target.

All HIV-1 RT inhibitors which are either approved or in clinical trials bind to either the NRTI binding site or the NNRTI binding site(8, 9). Several other classes of RT inhibitors have been identified including dyes, natural products, sulfated polysaccharides, and small naphthalene sulfonic acids(10-12). Some of these compounds specifically inhibit HIV-1 RT, while others inhibit a wide variety of polymerases(13-16). The mechanism of inhibition has not been established for most of these compounds. However, in gel-shift studies by Hizi(15, 17), a series of marine natural products appears to block RT-nucleic acid binding and RT Pol activity, but paradoxically, not RNaseH activity. In this paper we use structure-based design, molecular similarity, and combinatorial medicinal chemistry to identify and develop a chemically distinct class of compounds that inhibits the nucleic acid binding, RNaseH activity, and polymerase activity of HIV-1 RT with low-nanomolar potency.

Materials and Methods

Structure-Based Design (Figure 1, Step 1). The DOCK algorithm has been described in detail elsewhere(18-21). As a brief overview, concavities in the molecular surface of a macromolecule are filled with atom-sized spheres to generate a negative image of each potential binding site. Distances between sphere centers are matched to distances between ligand atoms to produce thousands of potential ligand orientations within the site. Each ligand orientation is evaluated based on steric and chemical complementarity to the target macromolecule in order to create a list of the molecules predicted to bind to the target.

To identify potential binding sites, we calculated a molecular surface(22) of the RT-DNA complex(23) and generated 29 clusters of spheres. Three clusters (e.g., binding sites) were selected based on their distance from the NNRTI binding site, functional significance, sequence conservation of nearby residues, crystallographic reliability, and geometric quality of the pocket.

A database of commercially available small molecules was prepared from the ACD v93.2 (Available Chemicals Directory, Molecular Design Limited, San Leandro, CA). Three-dimensional structures of each compound were obtained using CONCORD(24), and partial charges were generated using the Gasteiger-Marseili(25) method. Compounds were evaluated using the intermolecular van der Waals and electrostatic terms from the AMBER force-field(26) as well as an empirical scoring function (Ewing T.J.A., and Kuntz, I.D., unpublished results). Compounds were selected for biological screening based on: (i) force-field and empiric score (ii) dissimilarity to known non-nucleoside inhibitors, (iii) visual inspection of their proposed binding mode, (iv) solubility and reactivity, (v) chemical diversity, (vi) toxicity, (vii) and ease of analog synthesis. Dissimilarity was assessed using the connectivity metric described by Bemis(27).

Similarity Searches (Figure 1, Step 2)**.** The ACD v95.1 (Available Chemicals Directory, Molecular Design Limited, San Leandro, CA) was converted to Daylight format(28). Searches were carried out with Daylight's connectivity measure of similarity and the Tanimoto similarity metric(29). Daylight's Merlin search engine(28) was used to probe the ACD for related structures.

Reagent Selection. A new program, UC_Select (Skillman, A., Kuntz, I., UCSF, San Francisco, CA), was used to identify reagents from the ACD that were both chemi-

17

case

ease

cally compatible with analog synthesis (*vida infra*) and had appropriate medicinal properties (i.e., solubility, molecular weight, non-toxicity, non-reactivity).

Chemicals. Compound **1** was purchased from Pfaltz & Bauer (Waterbury, CT). Compound **2** was purchased from Sigma. While the ACD structure for **2** is asymmetric, we found the correct structure is the symmetric compound shown in Table 1 (based on mass spectrum, NMR, and HPLC characterization, data not shown).

General. L-SIMS and electrospray mass spectral analyses were performed by the UCSF mass spectrometry facility, A. L. Burlingame, Director. MALDI spectra were performed on a PerSeptive Biosystems Voyager-DE instrument and were internally calibrated by close proximity spotting. NMR spectra (^{1}H, and ^{13}C) were taken on a GE 300 MHz instrument. Centrifugation was performed at top speed on an International Clinical Centrifuge.

Synthesis of compounds 3, 4, and 5 (Figure 1, Step 3)**.** A sample of the aryl amines, compound **3**, 4-amino-benzoic acid (263.6 mg, 1.92 mmole); compound **4**, 4-aminocinnamic acid hydrochloride salt (376.2 mg, 1.88 mmole); compound **5**, 4-amino-3-methylbenzoic acid (283.4 mg, 1.87 mmole) was slurried in 2 mL of water, and 1.25 mL of 20% H_2SO_4 was added followed by a solution of $NaNO_2$ (1 eqv. 122-125 mg) in 1 mL water. The reaction was mixed with a pipette until 95+% of the precipitate dissolved (2-5 minutes) and was then added to 15 mL water. A saturated solution of Na_2CO_3 was added until a pH of about 9 was reached (6 mL). A solution of compound **1** (0.5 eqv, approx 510 mg) fully dissolved in 10 mL warm water was added, followed by 3 mL sat Na_2CO_3 to assure basicity, and the resulting solution was allowed to stir at room temperature for 2.5 hours after which time it was acidified with 20% H_2SO_4 to a pH of approximately 0 and

diluted to 140 mL volume with water. The resulting precipitate was isolated by centrifugation, washed with water, methanol or ethanol and then dried under vacuum overnight to yield the desired products. Analytical data for Compound **3**, - ion LRESMS found (MH-) 799, (MNa-) 821, ^{1}H NMR (300 MHz, DMSO) δ 15.81 (s, 2H) 12.5-13.0 (broad s, 2 H) 9.53 (s, 2H) 8.21 (d, J= 8.7 Hz, 2H) 7.99 (d, J=8.1 Hz 4H) 7.82 (d, J= 8.1 Hz, 4H) 7.76 (s, 2H) 7.50 (s, 2H), ^{13}C NMR (75 MHz, DMSO) δ 177.81, 166.88, 151.85, 146.14, 144.82, 143.50, 137.00, 130.84, 129.40, 128.65, 126.90, 125.04, 122.29, 117.91, 116.62, 116.35, Compound **4**, - ion LRESMS found (MH-) 851, (MNa-) 873, ^{1}H NMR (300 MHz, DMSO) δ 15.94 (s, 2H) 12.08 (broad s, 2H) 9.58 (s, 2H) 8.20 (d, J= 8.4 Hz, 2H) 7.77 (m, 12H) 7.62 (d, J= 15.9 Hz, 2H) 7.51 (s, 2H) 6.56 (d, J=15.9 Hz, 2H), ^{13}C NMR (75 MHz, DMSO) δ 177.00, 167.34, 151.55, 144.25, 143.87, 143.71, 143.12, 136.72, 131.08, 129.37, 128.76, 128.16, 124.70, 121.05, 117.89, 117.37, 117.11, 115.89, Compound **5** - ion LRESMS found (MH-) 827, (MNa-) 849, ^{1}H NMR (300 MHz, DMSO) δ 16.17 (s, 2H) 9.64 (s, 2H) 8.19 (d, J= 7.8 Hz, 2H) 8.09 (d, J= 7.5 Hz, 2H) 7.85 (s, 6H) 7.62 (d, J= 7.8 Hz, 2H) 7.57 (s, 2H) 2.45 (s, 6H), ^{13}C NMR (75 MHz, DMSO) δ 177.77, 167.06, 151.77, 144.78, 144.07, 143.33, 136.92, 132.09, 130.14, 128.71, 128.55, 126.64, 125.17, 124.93, 122.39, 117.91, 116.20, 115.33, 56.09 (EtOH) 18.60 (EtOH) 16.72.

Amersham Polymerase Assay. Enzyme activity was measured with a scintillation proximity assay(30), which uses a biotin/streptavidin bead capture system. RNA-dependent Pol activity was measured using purified HIV-1 RT (Worthington Biochemical, Lakewood, NJ) and a synthetic 17mer/50mer RNA-DNA template/primer containing biotin at the 5‘ end of the DNA. Inhibitors were dissolved in DMSO. The template-primer, buffer,

dNTP, and inhibitor were incubated for 10 minutes at 37° C before the RT was added. The final 100µL reaction mixture contained: 39nM of template-primer; buffer of 40mM Tris HCl pH8.0 / 10mM $MgCl_2$ / 60mM KCl / 10mM dithiothreitol; 75µM of each dNTP except dTTP; 25µM dTTP and 35µM tritium labeled d*TTP; 10µM inhibitor; and 2.0µg of RT. After three minutes incubation at 37° C, the reaction was quenched with 40µL of 0.56M EDTA. Then 10µL of streptavidin SPA beads (in suspension) was added and incubated at 37° C for 10 minutes. Finally, 850 µL of 10mM Tris Cl, pH7.4, 0.15M NaCl was added. The signal was produced by biotinylated polymers that contained d*TTP binding to the streptavidin beads and stimulating the scintillant. The amount of d*TTP polymerized was measured using a scintillation counter. Results for each concentration of inhibitor were recorded as counts per minute relative to an uninhibited standard.

Resistant Mutants. The mutants were constructed using BspMI cassette mutagenesis, as described previously (31, 32). The assay has also been previously described(33). Briefly, RNA dependent DNA Pol activity was assayed using purified RT protein and a poly(rC)•oligo(dG), 0.02mM dGTP, 0.002mCi $[\alpha\text{-}^{32}P]$dGTP, and a buffer of 25mM Tris Cl, pH 8.0, 75mM KCl / 8.0mM $MgCl_2$ / 2.0mM dithiothreitol / 10mM 3-[(3-cholamidopropyl)dimethylammonio-1-propane-sulfonate (CHAPS) containing acetylated bovine serum albumin at 100µg/mL. The assay mixture was incubated for 30 minutes, then the reaction was stopped by the addition of 50 µL of a 10-mg /mL solution of sheared and denatured salmon sperm DNA followed by 3.0mL 10% trichloroacetic acid (TCA). The labeled polymer was collected by suction filtration on Watchman glass GF/C and counted.

NCI polymerase assay. The RNA dependent DNA polymerase activity was measured with the method described above for resistant mutant assays.

17

cise

ecisee

RT-DNA binding. The oligonucleotides used for attachment to the surface plasmon resonance surface (SPR) were synthesized with biotin at their 3' ends using TEG CPG (Glen Research, Sterling, VA). The biotinylated plus strand oligonucleotide (5'-A GCA GTG GCG CCC GAA CAG GGA CCT GAA AGC-3' biotin) was mixed with equimolar minus strand (3'-GGGCTTGTCCCTGGACTTTCG-5'), heated to 95° C, and allowed to cool to room temperature for annealing. This forms a DNA version of a primer/ template that contains sequences from the LTR of HIV-1.

SPR was performed with a BIAcore instrument manufactured by Biosensor AB (Uppsala, Sweden) using methods described previously(34). The buffer used in the SPR experiments was 0.15M NaCl - 10mM HEPES (pH 7.5) 5mM DTT - 0.05% Tween 20. Binding experiments were initiated by passing buffer across an SPR sensor chip containing a known amount of oligonucleotide for approximately 100 s at 5 µL/min, followed by a 10 µL injection of buffer containing RT solution. Injection of the RT sample was followed by buffer for an additional 200 s. The chip surface was then regenerated with two successive 5 µL pulses of 0.1% sodium dodecyl sulfate (SDS) - 3mM EDTA.

A standard curve relating the initial binding slope(35) of RT binding to 225 Resonance Units of primer/template was constructed by injecting solutions containing 0.5, 1, 2.5, 5, 10, 25, 50, 100, 250, and 500nM HIV-1 RT. Solutions of 500nM HIV-1 RT were incubated with different concentrations of inhibitor, and free HIV-1 RT was measured by injection of the solution and determination of the resultant initial slope. The free HIV-1 RT was plotted against inhibitor concentration, and the curve was fit using the solution affinity model contained in the BIA software version 3.01 to obtain K_d's.

RNaseH assay. The assay for RNaseH activity has been described elsewhere(36,

37).

Cell Culture. The viral replication assay has been described elsewhere(38). This assay generates both an effective concentration for cell rescue from viral infection (EC_{50}) and a toxic concentration for cell death independent of viral infection (TC_{50}). From these EC_{50}'s and TC_{50}'s, a therapeutic index (T.I.) was calculated (TC_{50}/EC_{50}).

Results

We screened the ACD with DOCK at three binding sites. These three sites are located (i) adjacent to the Pol active site, (ii) at the base of the thumb, and (iii) in the area between the thumb and the minor groove of DNA. Ninety-two compounds were selected for the three binding sites and screened for their ability to inhibit HIV-1 RT Pol activity. Twenty-seven of the ninety-two compounds (29%) showed at least 5% inhibition in a 10μM assay and corresponding increases in inhibition at 30μM and 100μM. The best compound from our initial screening was **1** with an IC_{50} of 5μM (Table 1, Figure 1), while 9-Chloro-Thiobenzimidazolone (9-Cl-TIBO) in the same assay conditions had an IC_{50} of 20μM.

To demonstrate that **1** does not bind to the nucleoside or the non-nucleoside binding site, we assayed it against a variety of HIV-1 RT drug resistant mutants including seven NNRTI resistant single mutants (Y181L, Y188L, L100I, K103N, V106A, E138K and P236L), two NRTI resistant mutants (L74V and M184V), and the AZT-21 mutant (M41L, D67N, K70R, T215Y, and K219Q). Compound **1** was as effective against all ten resistant mutants tested as against wild-type HIV-1 RT (data not shown).

Based on these results, we pursued similar available compounds and identified **2**, a

superstructure of **1**. Compound **2** had an IC_{50} of 1.5μM versus HIV-1 RT Pol activity (Table 1, Figure 1). In addition, the results obtained with compound **2** inspired the synthesis of a wide range of aryl diazo derivatives (Table 2, Figure 1). We also replaced the central urea with a variety of linkers including thiourea, oxalyl, squarate, chelidonate, chelidamate, 2,6-pyridine dicarboxylate, and terephthalate groups. Finally, we modified the linkers and aryl diazo groups together in a combinatorial fashion. The most potent compounds contained urea linkers and carboxylic acid groups (**3-5**, Table 2). Inhibition of HIV-1 RT Pol activity was measured by two independent methods. The Amersham assay and the NCI assay agreed to within a factor of four for each molecule. Compound **4** was the most potent compound with IC50s of 90nM and 24nM, respectively, in the two assays (Table 2).

We explored the ability of each of the compounds to inhibit binding of HIV-1 RT to DNA using a BIAcore assay. A prerequisite for this study is that the compounds do not bind to the CM5 chip. Unfortunately, **5** bound to the surface of the chip so its K_d could not be determined. We were, however, able to measure dissociation constants for **1**, **2**, **3**, **4**, and 8-Cl-TIBO. 8-Cl-TIBO, the negative control, did not inhibit RT binding to DNA at any concentration measured. Our initial compounds, **1** and **2**, inhibited RT binding to DNA with K_d's of 2μM and 0.54μM, respectively (Table 1). The more potent compounds, **3** and **4**, inhibited RT binding to DNA with K_d's of 105nM and 40nM, respectively (Table 2).

Compounds **4**, **5** and 8-Cl-TIBO were tested for their ability to inhibit HIV-1 RT RNaseH activity. Each compound was tested at concentrations of 25nM, 100nM, 500nM, and 2000nM at a four minute time interval (Figure 2). 8-Cl-TIBO showed no RNaseH

inhibition at concentrations up to 50mM (data not shown). Compounds **4** and **5** each demonstrated an IC_{50} between 25nM and 100nM (Figure 2,Table 2) showing that the two most potent Pol inhibitors are also potent RNaseH inhibitors.

Having identified several potent inhibitors, we investigated their selectivity. Compounds **4** and **5** were tested for inhibition of two viral reverse transcriptases (HIV-1 RT and MLV RT) and three prokaryotic DNA polymerases (T4 DNA Pol, T7 DNA Pol, and the Klenow fragment). Compounds **4** and **5** showed significant inhibition of HIV-1 RT and MLV RT at 200nM (Figure 3). Neither of the compounds showed inhibition of T7, T4,or the Klenow DNA Pol activities at concentrations up to 200nM. Compounds **4** and **5** are potent and selective inhibitors of retroviral RTs.

Finally, we tested our most promising compounds in a cell culture assay. The EC_{50}'s of compounds **3**, **4**, and **5** are 8.8μM, 2.5μM, and 2.5μM, respectively (Table 3). The corresponding toxicity measurements for compounds **3**, **4**, and **5** are >100μM, 112μM, and 120μM, respectively, with therapeutic indices of 10-50, indicating that these compounds are not toxic to the cells at concentrations an order of magnitude higher than their EC_{50}'s (Table 3).

Discussion

HIV infection and AIDS remain a major medical problem(2). The rapid development of resistance to current drug therapies serves to highlight the need for new therapies which have independent mechanisms of action. We have used structure-based design principles to identify novel, potent, and selective inhibitors of HIV-1 RT. These inhibitors act by a unique mechanism and are not susceptible to the resistance mutations which elimi-

nate the activity of NNRTIs or NRTIs. Three specific topics merit further discussion. First, we consider the mechanism of action of these naphthylurea derivatives. Second, we examine the utility of structure-based design in a system as challenging as HIV-1 RT. Finally, we discuss the structural and medicinal properties of these compounds.

Mechanism of Action. Let us summarize the evidence that these compounds inhibit HIV-1 RT by a novel mechanism. The BIAcore data (Tables 1 and 2) strongly support the hypothesis that this series of compounds prevents HIV-1 RT from binding to DNA duplexes. In sharp contrast, neither NNRTIs nor NRTIs inhibit such binding. Further, the BIAcore data make the decisive prediction that the RNaseH activity of HIV-1 RT should be inhibited by these compounds in a similar concentration range. The RNaseH data (Figure 2) clearly show that compounds **4** and **5** inhibit HIV-1 RT RNaseH. Although NNRTIs like Nevirapine can alter the specificity of RNaseH cleavage, they do not block RNaseH activity(39). Furthermore, in our control, 8-Cl-TIBO had no effect on RNaseH activity (data not shown). Thus, although the binding site and exact binding modes remain unknown, we have shown that these compounds interact with HIV-1 RT and prevent its binding to DNA.

The data presented here imply that these compounds prevent HIV-1 RT's normal interaction with nucleic acid duplexes. The three most plausible mechanisms for this inhibition are 1) binding to RT in a mode competitive with the binding of the nucleic acid duplexes, 2) binding to a secondary site and thereby preventing an essential conformational change required for duplex binding, or 3) binding to DNA and interfering with its interaction with HIV-1 RT.

The specificity data in Figure 3 show conclusively, not only that compounds **4** and

5 are specific RT inhibitors, but also that they do not act by binding to DNA. If these compounds inhibited RT by binding to nucleic acids, they would inhibit the three DNA Pols in addition to the two viral RTs. The enzymatic inhibition, DNA binding inhibition, and selectivity of this class of compounds set them apart from all previously identified HIV-1 RT inhibitors.

Aspects of the structure of these compounds appears intriguingly similar to DNA and they may bind to HIV-1 RT as DNA mimics. For example, in a planar conformation, the inter-sulfate distance is within the same range as an inter-phosphate distance across a base-pair. If this series bind to HIV-1 RT as DNA mimics, this would favor the hypothesis that they are competitive inhibitors of nucleic acid binding. Further, if these compounds prove to be examples of a broader class of DNA mimics, new analogs may bind specifically to other DNA binding proteins such as retroviral integrases or host transcription factors.

Although these compounds are all potent HIV-1 RT inhibitors *in vitro,* our data do not exclude alternate modes of action in cell-based assays. For instance, unrelated aromatic poly-sulfonic acids have been shown to inhibit GP-120 binding to CD-4(41). Compounds **3**, **4**, and **5** all inhibit HIV-1 replication in cell culture without causing cellular toxicity. The EC_{50}'s of **3**, **4**, and **5** in culture are one to two orders of magnitude higher than the corresponding *in vitro* measurements. This may be due to poor cell permeability, cellular efflux pumps, or cellular modification of the compounds. Compounds **4** and **5** may be present in concentrations more than 40 times their EC_{50} without generating cytotoxicity, and thus are suitable for further cell-culture study. We hope to investigate the effects of this series of compounds on the replication of other viruses.

The compounds with inhibitory properties closest to the series presented here are the marine natural products Toxiusol and Peyssonol A and B(15, 17). Our compounds potently inhibit HIV-1 RT Pol activity, RNaseH activity, and DNA binding. In contrast, Peyssonols A and B do not inhibit HIV-1 RNaseH activity and Toxiusol has not been reported to inhibit HIV-1 RNaseH activity. Furthermore, our compounds are selective for retroviral RTs, whereas Toxiusol inhibits the Klenow fragment. It is clear that the naphthylureas presented here represent a new direction in RT inhibition.

Structure-Based Design. HIV-1 RT has presented a difficult crystallographic challenge. However, the original RT-DNA structure(23) contained sufficient information to allow DOCK to identify 27 diverse inhibitors of HIV-RT Pol activity including **1**, which was sufficiently potent (IC_{50} 5μM) to merit further development. These results demonstrate that structure-based screening can be useful even with target structures whose resolution is >2.5 Å.

The NNRTIs encompass a wide array of chemical classes which all bind to the same site and are associated with a characteristic sets of resistance mutants(8). In the design phase, we biased binding site selection away from the non-nucleoside binding site and biased compound selection away from known inhibitors. This biased design strategy successfully identified **1**, which was equally effective against wild-type HIV-1 RT and HIV-1 RT mutants resistant to either NNRTI's or NRTI's. This evidence indicates that **1** does not bind to the non-nucleoside binding site and that structure-based design can be used to specifically target or avoid a particular site on a large macromolecule such as HIV-1 RT.

Although our attempts to grow co-crystals have not yet been successful, the struc-

ture of a complex of any of these compounds bound to HIV-1 RT would provide a useful starting place for design of potential drugs. Until such data are available, we are exploring several hypothetical binding modes based on the structure of the HIV-1 RT/DNA complex(40). One promising binding site is a deep groove at the interface of the p66 and p51 subunits near residues W406, Q507, and A508 of p66, and N418 of p51. However, the binding site and exact binding modes of our compounds remain unknown (except that the compounds do not appear to bind to the NNRT or NNRTI binding sites).

Medicinal Chemistry. Compound **2** and its analogs **3-5** are potent superstructures of compound **1**. Although the high degree of symmetry in the best inhibitors is somewhat surprising, the larger compounds presumably are more potent because the additional groups are able to explore interactions in pockets adjacent to the original binding mode. On the other hand, it is not uncommon for analogs of low potency inhibitors (such as **1**) to explore alternative binding modes. Although we synthesized compounds with a wide variety of central linkers and aryl diazonium side-chains (Figure 1), the compounds that are most active against HIV-1 RT were the urea-linked compounds with acidic aryl diazo side-chains (**3-5**). All of these compounds are quite potent *in vitro*. In their current form, however, this class of inhibitors has some drawbacks, including multiple formal charges, diazonium groups, and high molecular weights. When we replace the distal acids with an isoelectric (and medicinally favored) tetrazole, it retains most of its activity (HIV-1 RT Pol IC_{50} = 220nM, data not shown). However, we were not able to identify potent smaller or non-sulfated molecules. In order for these compounds to be suitable for study in animal models, further medicinal chemistry development will be necessary. Nevertheless, these compounds are the first in a novel class of HIV-1 RT inhibitors.

Conclusions

We have designed and developed a new class HIV-1 RT inhibitors which act by a novel mechanism of action. Using a structure-based design strategy biased away from previously known drugs and binding sites, we identified compounds that prevent HIV-1 RT from binding to nucleic acids. This class of compounds are both potent and selective inhibitors of viral reverse transcriptases and are not effected by any of the major HIV-1 RT resistance mutants which were tested. These compounds and their mechanism of action are distinct from any previously described class of HIV-1 RT inhibitors. These compounds demonstrate that HIV-1 RT has a previously undescribed binding site which is capable of supporting potent small molecule binding and inhibition. The naphthylurea compounds identified here represent an exciting new direction in HIV-1 RT inhibitors.

Acknowledgments

We thank PerSeptive Biosystems and the Bradford Gibson group for use of the Voyager MS. We thank Daylight Chemical Information Systems Inc., Molecular Design Limited Inc., and Tripos Associates for the use of their software. We thank Andrew Leavitt, Donna Hendrix, Malin Young, Piotr Cieplak, and Peter Kollman for their helpful discussions. We thank Leo Lee of NCI-Frederick Cancer Research and Development Center for the synthesis of oligonucleotides. This research was sponsored in part by the National Cancer Institute, DHHS, under contract with ABL. The contents of this publication do not necessarily reflect the views or policies of the Department of Health and Human Services, nor does mention of trade names, commercial products, or organizations imply endorsement by the U.S. Government. This work was also sponsored in part by the NIH under

GM39552.

Figures

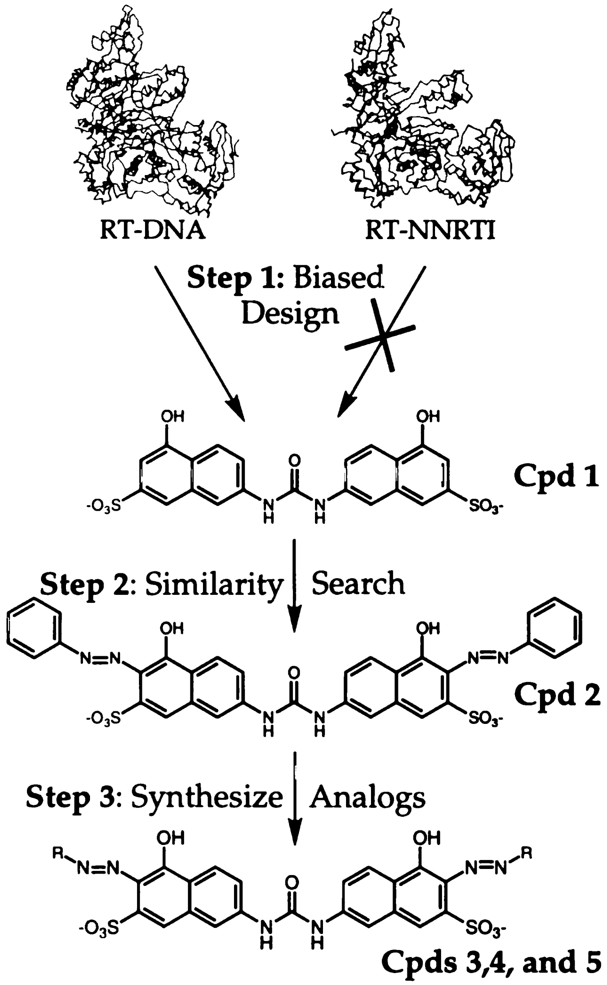
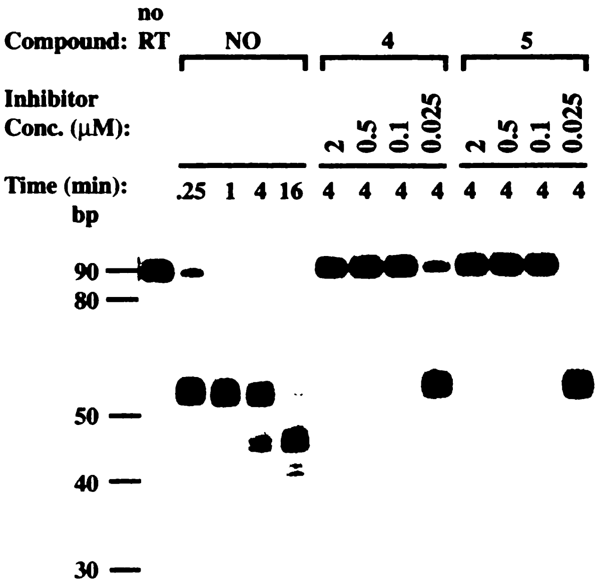


Figure 1: Ligand Design Scheme. A biased structure-based design scheme (step 1) was used to identify initial inhibitors. Similar compounds which were commercially available were found (step 2). A series of analogs which were potent and selective HIV-1 RT inhibitors were synthesized.

Figure 2: RNaseH Inhibition. HIV-1 RT cleavage of the RNA strand of an RNA-DNA duplex is completely inhibited at concentrations of 2, 0.5, and 0.1 µM and partially inhibited at 0.025 µM by compounds **4** and **5** (A).



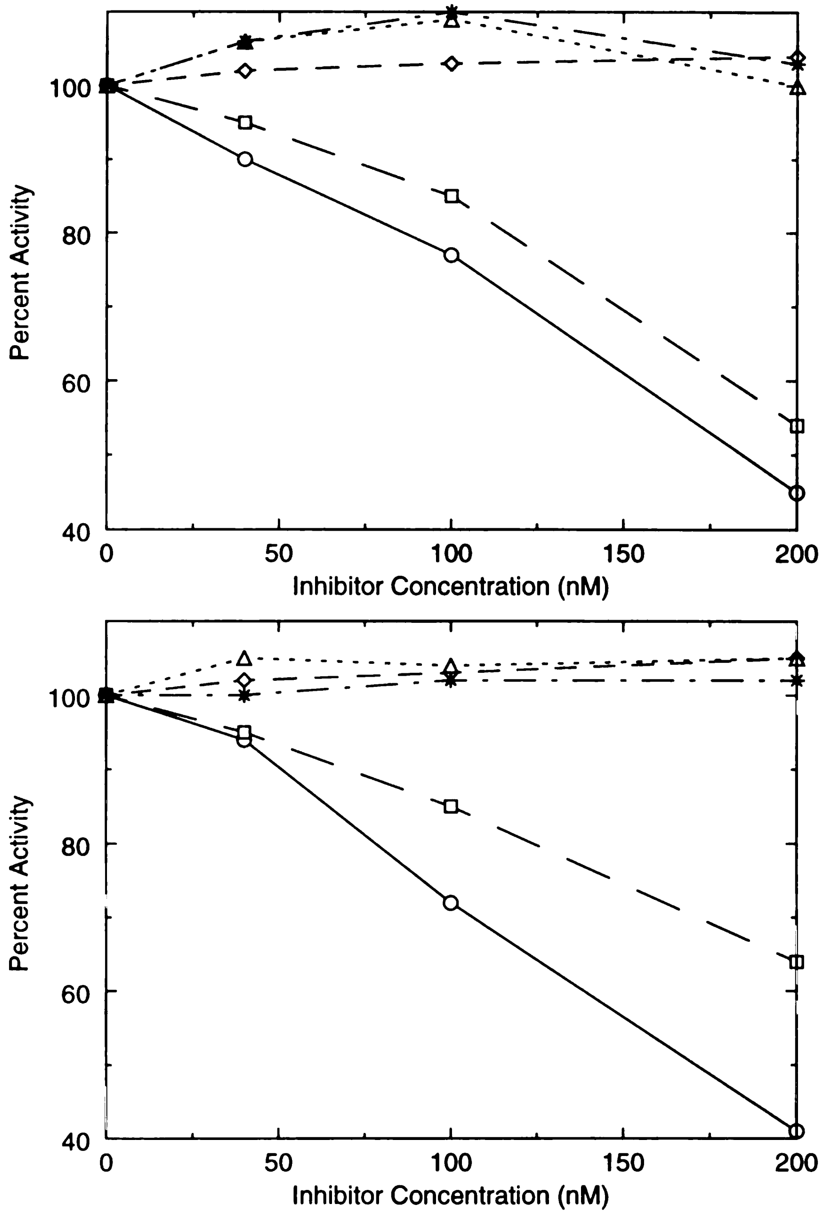


Figure 3: Selective inhibition of retroviral RTs. Inhibition of HIV-1 RT——⊖——, MLV RT – ⊟ –, T4 DNA Pol · — ◇ — ·, T7 DNA Pol ···△···, and Klenow fragment - - ✳ - - by compound **4** (**A**) and compound **5** (**B**).

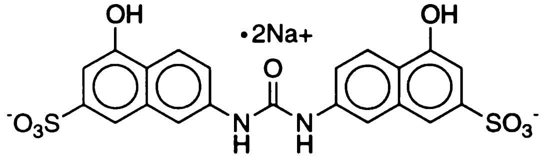
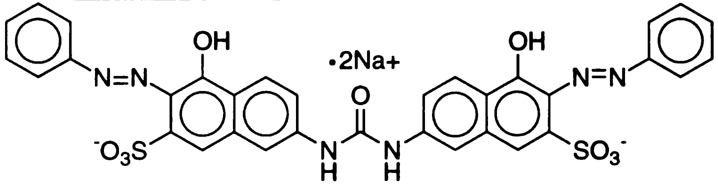
#	Compound Structure	HIV-1 RT polymerase IC_{50} (nM)	DNA binding Kd (nM)
1		5000	2000
2		1500	540

Table 1: HIV-1 RT polymerase and DNA binding inhibition data for two compounds identified by Structure-based design. a=Amersham polymerase assay.

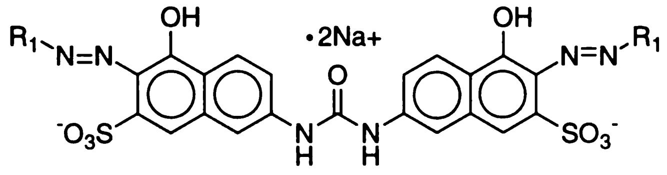
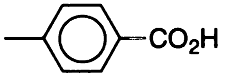
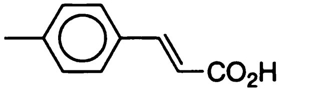
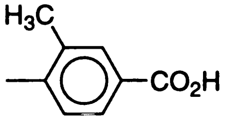
		HIV-1 RT polymerase IC_{50} (nM)[a]	HIV-1 RT polymerase IC_{50} (nM)[b]	RNaseH IC_{50} (nM)	DNA binding Kd (nM)	EC_{50} (μM)	TC_{50} (μM)	T.I. (TC_{50}/ EC_{50})
#	R1							
3		390	316	ND	105	8.8	>100	>11.4
4		90	24	25-100	40	2.5	112	44.8
5		150	42	25-100	INC	2.5	120	48

Table 2: HIV-1 RT Inhibition data. Inhibition of HIV-1 RT by compounds **3-5** in *in vitro* polymerase (two independent assays), RNaseH, and DNA binding assays. Effects of compounds **3-5** on cell culture HIV-1 infection (EC_{50}), cellular toxicity (TC_{50}), and Therapeutic Index (TC_{50}/EC_{50}) are shown in the final three columns. ND = Not Determined, INC = Incompatible, T.I. = Therapeutic Index, a = Amersham Assay, b = National Cancer Institute Assay.

Bibliography

1. Palella, F., Delaney, K., Moorman, A., Loveless, M., Fuhrer, J., Satten, G., Aschman, D. & Holmberg, S. (1998) *N. Engl. J. Med.* **338,** 853-860.

2. Piot, P. (1998) *Science* **280,** 1844-1845.

3. Carpenter, C., Fischl, M., Hammer, S., Hirsch, M., Jacobsen, D., Katzenstein, D., Montaner, J., Richman, D., Saag, M., Schooley, R., Thompson, M., Vella, S., Yeni, P. & Volberding, P. (1998) *Jama* **280,** 78-86.

4. Wong, J. K., Hezareh, M., Gunthard, H. F., Havlir, D. V., Ignacio, C. C., Spina, C. A. & Richman, D. D. (1997) *Science* **278,** 1291-1295.

5. Telesnitsky, A. & Goff, S. (1997) in *Retroviruses*, eds. Coffin, J., Hughes, S. & Varmus, H. (Cold Spring Harbor Laboratory Press, Plainview, New York), pp. 141-146.

6. Hecht, F., Grant, R., Petropoulos, C., Dillon, B., Chesney, M., Tian, H., Hellmann, N., Bandrapalli, N., Digilio, L., Branson, B. & Kahn, J. (1998) *N. Engl. J. Med.* **339,** 307-311.

7. Declercq, E. (1992) *Aids Research and Human Retroviruses* **8,** 119-137.

8. Emini, E. & Fan, H. (1997) in *Retroviruses*, eds. Coffin, J., Hughes, S. & Varmus, H. (Cold Spring Harbor Laboratory Press, Plainview, New York), pp. 666-677.

9. Huang, H., Chopra, R., Verdine, G. & Harrison, S. (1998) *Science* **282,** 1669-1675.

10. Tan, G., Wickramasinghe, A., Verma, S., Singh, R., Hughes, S., Pezzuto, J., Baba, M. & Mohan, P. (1992) *J. Med. Chem.* **35,** 4846-4853.

17

case

17

case

25

11. Mohan, P., Loya, S., Avidan, O., Verma, S., Dhindsa, G., Wong, M., Huang, P., Yashiro, M., Baba, M. & Hizi, A. (1994) *J. Med. Chem.* **37,** 2513-2519.

12. Moelling, K., Schulze, T. & Diringer, H. (1989) *J. Virol.* **63,** 5489-5491.

13. Sun, H., Qiu, S., Lin, L., Wang, Z., Lin, Z., Pengsuparp, T., Pezzuto, J., Fong, H., Cordell, G. & Farnsworth, N. (1996) *J. Nat. Prod.* **59,** 525-527.

14. Loya, S., Rudi, A., Kashman, Y. & Hizi, A. (1997) *Biochem. J.* **324,** 721-727.

15. Loya, S., Bakhanashvili, M., Kashman, Y. & Hizi, A. (1995) *Biochemistry* **34,** 2260-2266.

16. Ohta, K., Mizushina, Y., Hirata, N., Takemura, M., Sugawara, F., Matsukage, A., Yoshida, S. & Sakaguchi, K. (1998) *Chem. Pharm. Bull. (Tokyo)* **46,** 684-686.

17. Loya, S., Bakhanashvili, M., Kashman, Y. & Hizi, A. (1995) *Arch. Biochem. Biophys.* **316,** 789-96.

18. Meng, E., Shoichet, B. & Kuntz, I. (1992) *J. Comp. Chem.* **13,** 505-524.

19. Meng, E., Gschwend, D., Blaney, J. & Kuntz, I. (1993) *Protein-Struct. Funct. Genet.* **17,** 266-278.

20. Kuntz, I., Blaney, J., Oatley, S., Langridge, R. & Ferrin, T. (1982) *J. Mol. Biol.* **161,** 269-288.

21. Ewing, T. & Kuntz, I. (1997) *J. Comput. Chem.* **18,** 1175-1189.

22. Connolly, M. L. (1983) *Science* **221,** 709-713.

23. Jacobo-Molina, A., Ding, J., Nanni, R., Clark, A., Lu, X., Tantillo, C., Williams, R., Kamer, G., Ferris, A., Clark, P., Hizi, A., Hughes, S. & Arnold, E. (1993) *Proc. Natl. Acad. Sci. USA* **90,** 6320-6324.

24. Rusinko, A., Sheridan, R., Nilakatan, R., Haraki, K., Bauman, N. & Venkataghavan, R. (1989) *J. Chem. Inform. Comput. Sci.* **29,** 251-255.

25. Gasteiger, J. & Marsili, M. (1980) *Tetrahedron Lett.* **36,** 3219.

26. Weiner, S., Kollman, P., Case, D., Singh, U., Ghio, C., Alagona, G., Profeta Jr, S. & Weiner, P. (1984) *J. Amer. Chem. Soc.* **106,** 765-784.

27. Bemis, G. & Kuntz, I. (1992) *J. Comput. Aid. Molec. Design* **6,** 607-628.

28. Thor, Merlin, and Daylight Toolkits, version 4.51, Daylight Chemical Information Systems, Inc., Santa Fe, NM.

29. Willett, P. (1987) *Similarity and Clustering in Chemical Information Systems* (John Wiley & Sons, New York, NY, USA).

30. RT SPA enzyme assay kit NK8972, Amershan Life Science, Inc., Arlington Heights, IL 60005.

31. Boyer, P., Ferris, A. & Hughes, S. (1992) *J. Virol.* **66,** 7533-7537.

32. Boyer, P., Ferris, A. & Hughes, S. (1992) *J. Virol.* **66,** 1031-1039.

33. Boyer, P., Tantillo, C., Jacobo-Molina, A., Nanni, R., Ding, J., Arnold, E. & Hughes, S. (1994) *Proc. Natl. Acad. Sci. USA* **91,** 4882-4886.

34. Fisher, R., Fivash, M., Casasfinet, J., Erickson, J., Kondoh, A., Bladen, S., Fisher, C., Watson, D. & Papas, T. (1994) *Prot. Sci.* **3,** 257-266.

35. Christensen, L. (1997) *Anal. Biochem.* **249,** 153-164.

36. Loya, S., Gao, H., Avidan, O., Boyer, P., Hughes, S. & Hizi, A. (1997) *J. Virol.* **71,** 5668-5672.

37. Gao, H., Boyer, P., Arnold, E. & Hughes, S. (1998) *J. Mol. Biol.* **277,** 559-572.

38. Weislow, O., Kiser, R., Fine, D., Bader, J., Showmaker, R. & Boyd, M. (1989) *J. Nat. Cancer Inst.* **81,** 577-586.

39. Palaniappan, C., Fay, F. & Bambara, R. (1995) *J. Biol. Chem.* **270**, 4861-4869.

40. personal communication, Jacobo-Molina, A.

41. Baba, M., Schols, D., Mohan, P., DeClercq, E. & Shigeta, S. (1993) *Antivir. Chem. Chemother.* **4,** 229-234.

Prologue to Chapter 6

Chapter 6 is a classic example of rigid molecular docking. We screened the Available Chemicals Database (ACD) and identified weak inhibitors of the essential *T. foetus* enzyme hypoxanthine-guanine-hypoxanthine-phosphoribosyltransferase (HGXPRTase). Two significant advances were made during this docking exercise. First, we used ease of analog synthesis as one of the criteria for compound selection. The impact of this bias was seen in chapter 3 where one of the lead compounds was optimized with a simple synthetic library. Second, we improved the inhibition of our lead compounds by an order of magnitude using searches of the ACD for compounds similar to our initial hits. We show clearly that simple topological similarity can be used to recover some of the compounds missed (false negatives) by the original structure based design method. These two methods show that we can better utilize the information from structure-based design, first by improving our data-mining of the ACD and second, by integrating synthetic information into our structure-based models, we can design compounds which are both easy to synthesize and are potentially potent inhibitors.

While working in a scientific field where a significant portion of the highest quality work is done in industry, it is easy to become influenced by industry's appropriate fixation on fiscal considerations. To balance this position, we in academia have the luxury (even responsibility) to explore disease processes and treatments which are significant in many aspects, yet which have either to large a risk or too small a monetary reward for a commercial interest. The study of parasitic diseases are an outstanding opportunity for academic application of drug development methodology. This project satisfies some of these goals, as it serves as a proof of concept for this nucleic acid scavenger protein as a relevant target in s

pla

len

tio

pr

ni

ta

fo

h

c

a

V

in some protozoan parasites. Although *T. foetus* is not the highest profile parasite, it does play an important role in Bovine health in Africa. More importantly, *T. foetus* is an excellent model system, and some of the inhibitors developed here also show selective inhibition of the analogous enzymes in *Giardia lamblia* and *Schistosoma mansoni*, two protozoan parasites which cause significant human morbidity and mortality. Before beginning my graduate training, I spent three months in India working in rural missions hospitals. It was there that I first saw the devastation parasitic diseases can wreak on those not fortunate enough to afford sanitary living conditions. Although at the time I was struck by how powerless I was, as a relatively uneducated cultural outsider, to affect positive change, nearly a decade of medical and scientific education have done little to increase my ability to effect change in rural India. Unfortunately, financial and cultural obstacles often prevent appropriate prevention or treatment of parasitic diseases even when effective therapeutics exist.

Chapter 6

Rational Design of Novel Antimicrobials: Blocking Purine Salvage in a Parasitic Protozoan

by

John R. Somoza, A. Geoffrey Skillman Jr., Narsimha R. Munagala, C. M. Oshiro, Ronald M. A. Knegtel, Solomon Mpoke, Irwin D. Kuntz, and Ching C. Wang

Reproduced with permission from
Biochemistry, April 1998, 37(16):5344-5348.

ABSTRACT

All parasitic protozoa obtain purine nucleotides solely by salvaging purine bases and/or nucleosides from their host. This observation suggests that inhibiting purine salvage may be a good way of killing these organisms. To explore this idea, we attempted to block the purine salvage pathway of the parasitic protozoan *Tritrichomonas foetus*. *T. foetus* is a good organism to study because its purine salvage depends primarily on a single enzyme, hypoxanthine-guanine-xanthine phosphoribosyltransferase (HGXPRTase) and could provide a good model for rational drug design through specific enzyme inhibition. Guided by the crystal structure of *T. foetus* HGXPRTase, we used structure-based drug design to identify several nonpurine compounds that inhibited this enzyme without any detectable effect on human HGPRTase. One of these compounds, 4-[N-(3,4-dichlorophenyl)carbamoyl]phthalic anhydride (referred to as TF1), was selected for further characterization. TF1 was shown to be a competitive inhibitor of *T. foetus* HGXPRTase with respect to both guanine (in the forward reaction; K_i=13 μM) and GMP (in the reverse reaction; K_i=10 μM), but showed no effect on the homologous human enzyme at concentrations up to 1 mM. TF1 inhibited the *in vitro* growth of *T. foetus* with an EC_{50} of approximately 40 μM. This inhibitory effect was associated with a decrease in the incorporation of exogenous guanine into nucleic acids, and could be reversed by supplementing the growth medium with excess exogenous hypoxanthine or guanine. Thus, rationally targeting an essential enzyme in a parasitic organism has yielded specific enzyme inhibitors capable of suppressing that parasite's growth.

Introduction

Many pathogenic microorganisms are dependent on their hosts for key metabolites,

and it should be possible to exploit this dependency for the design of novel chemotherapeutic agents. For example, all of the parasitic protozoa (e.g. Plasmodium, Toxoplasma, Leishmania, Trypanosoma, etc.) lack the ability to synthesize purine nucleotides de novo (1). Instead, these organisms rely on salvage enzymes to obtain purine bases and nucleosides from their host and convert them to the corresponding nucleotides. Thus, interfering with purine salvage could be an effective way of killing these organisms.

We sought to demonstrate the feasibility of this approach by blocking purine salvage in the parasitic protozoan *Tritrichomonas foetus*, an organism that can cause embryonic death and infertility in cows (2). *T. foetus* is a good model system because its purine salvage pathway is well understood, and relies primarily on a single enzyme, hypoxanthine-guanine-xanthine phosphoribosyltransferase (HGXPRTase), to replenish its purine nucleotide pool (3). HGXPRTase catalyzes the transfer of the ribose-5-phosphate moiety of α-D-5-phosphoribosyl-1-pyrophosphate (PRPP) to the N9 position of hypoxanthine, guanine, or xanthine to form the corresponding ribonucleotide.

Although mammals can produce purine nucleotides *de novo*, they also make use of purine salvage pathways. This is an important caveat to the idea of targeting purine salvage for chemotherapeutic purposes, because mammals recycle hypoxanthine and guanine using hypoxanthine-guanine phosphoribosyltransferase (HGPRTase), an enzyme that shares 27% sequence identity with tritrichomonal HGXPRTase (4). Since a decrease in human HGPRTase activity can lead to hyperuricemia (5), it is important to block the pathogen's purine salvage pathway while leaving the corresponding mammalian pathway untouched.

Our goal was to selectively inhibit the *T. foetus* HGXPRTase. This should block the main route of purine salvage, thus leading to growth arrest of the parasite. The only

known inhibitors of the purine PRTases at present are purine analogs, and they are weak inhibitors with Ki values in the millimolar range (6). We decided that the most pragmatic strategy to identify novel inhibitors of *T. foetus* HGXPRTase would be to use the three-dimensional structure of this enzyme to guide our search. Furthermore, since the structure of the human HGPRTase has been also determined (7), this structure-driven approach should make it easier to identify compounds that selectively inhibit the parasite enzyme.

MATERIALS AND METHODS

Structure Analysis. The crystal structure of the *T. foetus* HGXPRTase was determined by Somoza *et al.* (8) at 1.9 Å resolution (PDB accession code 1HGX), and the crystal structure of the human HGPRTase was determined by Eads *et al.* (7) at 2.5 Å resolution (PDB accession code 1HMP). The software packages O (9), Insight II (10), MIDAS PLUS (11), Sybyl (12) and GEM (E. Fauman, unpublished work) were used for the display and analysis of the structures.

Docking. DOCK 3.5 (13, 14, 15) was used to screen the Available Chemicals Directory (ACD) for potential HGXPRTase inhibitors (16). The docking process includes four primary tasks: (a) creating a negative image of the putative binding site; (b) overlaying the negative image with small molecules from a database; (c) scoring many orientations of a molecule based on its complementarity to the protein; (d) ranking and reviewing the best scoring orientations of the most complementary small molecules. The docking protocol applied to this particular study was modified from the general method in the following manner. First, chemical compounds were segregated into small, medium and large size categories based on the number of heavy atoms in each compound (small: 10-20; medium: 21-29; large: 30-60). Matching and docking parameters were adjusted for each group of

chemicals in order to achieve similar sampling of the binding site. Second, instead of creating a negative image of the entire active site, the negative image was limited to the region of the active site that interacts with the bound guanine and proximal part of the ribose, and to parts of the active site that differ between the parasite and human enzymes (total number of spheres=34). Finally, the bump filter was modified. This filter is generally used to eliminate ligands which have too many atoms overlapping with the receptor. In addition to using the bump filter in the standard way, the bump grid in this application was modified to filter out molecules that had atoms extending out of the active site into solvent.

Similarity and Superstructure Searches. A set of potential binding mode models was proposed by running DOCK on the initial inhibitors from this study and saving multiple orientations per compound. In some cases, potential binding modes were further refined with steepest descent and conjugate gradient minimization using the TRIPOS force field in Sybyl 6.2 (12). Superstructure and similarity searches of the ACD were used to identify a large set of compounds which were chemically similar to the inhibitors found in the initial DOCK screen (17, 18). These searches were carried out with Daylight's Merlin system (19), using a Tanimoto similarity metric and Daylight's hashed connectivity fingerprints. We then assayed compounds from this set which tested specific aspects of our pharmacophore models. In general, compounds with conservative changes were selected.

Chemicals. All of the chemical compounds used in the enzyme assays were purchased from Sigma Chemical Co., the Sigma-Aldrich Library of Rare Chemicals (SALOR), Aldrich Chemical Company, Inc., Menai or Maybridge. For the preliminary enzyme assays,

[8-^{14}C] hypoxanthine (52 mCi/mmol) was from Moravek Biochemicals and [8-^{14}C] guanine (56 mCi/mmol) was obtained from ICN Radiochemicals. GMP, PRPP, pyrophosphate (PP_i) and the purine bases were purchased from Sigma Chemical Co. in the highest purity available.

Sources of Enzymes. Recombinant *T. foetus* HGXPRTase was purified to homogeneity from *Escherichia coli* strain SØ 606 transformed with the low-phosphate-inducible pBTf-prt expression plasmid (4). Recombinant human HGPRTase was purified from the same strain of *E. coli* transformed with the pBAcprt expression plasmid by a previously described procedure (20).

Enzyme Assays. For preliminary testing of the computer selected compounds, a radioactivity assay of HPRTase (using radiolabeled hypoxanthine as substrate) or GPRTase (using radiolabeled guanine as substrate) activity was employed (21). In the initial screen, each chemical compound was tested at 1 mM. Compounds exhibiting over 50% inhibition of the enzyme activity were titrated to lower concentrations for determination of the IC_{50} values. Chemical samples were dissolved in dimethylsulfoxide (DMSO) to a concentration of 10 mM and tested in assay solutions of no more than 10% DMSO, which has no effect on the enzyme activity. For kinetic analysis of the enzyme-catalyzed reactions, a spectrophotometric assay was performed as previously described (22).

Kinetic Analysis. Data on the initial rate of the enzyme catalyzed reaction were fitted to equations 1-3 using kinetics analysis software and plotted based on weight-based linear regression analysis (23, 24).

For competitive inhibition

$$v = VmaxS \; / \; [Km\,(1 + I/Kis\,) + S \;] \qquad (1)$$

For noncompetitive inhibition

$$v = VmaxS \; / \; [Km\,(1 + I/Kis\,) + S\,(1 + I/Kii\,)](2)$$

For uncompetitive inhibition

$$v = VmaxS \; / \; [Km + S\,(\,1 + I/Kii\,)] \qquad (3)$$

The best fit was determined in each case by the relative fit error and errors in the constants. The nomenclature is that of Cleland (25): *v*, initial velocity; *Vmax*, maximum velocity; *S*, substrate concentration; *Km*, apparent Michaelis constant; *Kis* and *Kii*, slope and intercept inhibition constants, respectively; *I*, inhibitor concentration.

***In Vitro T. foetus* Cell Culture.** *T. foetus* kv1 strain trophozoites were cultivated to mid-logarithmic phase in Diamond's TYM medium at 37o C (26), and inoculated into fresh medium at a 10-fold dilution with resumption of the incubation. Time samples were taken and the cell number in each sample was determined using a hemocytometer under a microscope. The concentration of DMSO in the culture medium was maintained at or below 1% in order to avoid any adverse effect on cell growth.

RESULTS AND DISCUSSION

The Active Site of *T. foetus* HGXPRTase. Figure 1 shows the active site of *T. foetus* HGXPRTase with bound GMP. The phosphate, ribose, and purine base moieties of the bound GMP interact with the protein through a network of hydrogen bonds. In addition,

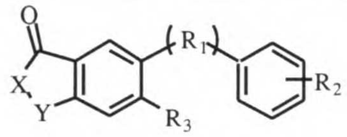
the purine base is inserted into a hydrophobic pocket, formed on one side by Tyr 156 and on the other side by Phe 162 and Ile 104. The active site of HGXPRTase extends beyond the region where the GMP is bound, and includes a loop formed by residues 46 through 49. This loop appears to interact with pyrophosphate (27).

Somoza *et al.* provided a detailed comparison of the active site of *T. foetus* HGX-PRTase with that of human HGPRTase (8). This comparison revealed significant differences between the two active sites in the region that interacts with the C2 substituent of the purine base. Differences in this region were expected because the parasite and mammalian enzymes have different substrate specificities. The parasite enzyme accepts hypoxanthine, guanine, and xanthine as substrates with similar K_m values (28), while the mammalian enzyme accepts only hypoxanthine and guanine as substrates (29). The only difference between these three purine bases lies in the identity of the C2 substituent.

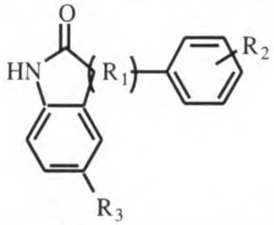
Using the Structural Information to Identify Potential HGXPRTase Inhibitors. The docking algorithm of Kuntz and co-workers (13, 14, 15) was used to screen the ACD for compounds showing van der Waals and electrostatic complementarity with the active site of *T. foetus* HGXPRTase. Since the HGXPRTase active site is large and shallow (approximately 10Å x 10Å x 5Å), we focused primarily on the region encompassing the GMP binding site. This region is well defined by the electron density map and differs from the corresponding region of the human HGPRTase.

Based on our computational screen, we assayed eighteen chemical compounds for their ability to inhibit *T. foetus* HGXPRTase *in vitro*. The two most active compounds from this initial screen are shown at the top of Table 1 (#1 and #2).

Table 1:



Cpd	X	Y	R1	R2	R3	Tf IC$_{50}$(µM)	Hu IC$_{50}$(µM)
1	**O**	**C=O**	**C(=O)NH**	**3-NO$_2$**	**H**	**300**	**>1000**
3	O	C=O	SCH2	4-Cl	NO$_2$	425	>1000
4	O	CH2	HNC=O	4-Cl	H	380	>1000
5	O	C=O	C(=O)NH	4-NHC(=O)C	H	140	>1000
6	O	C=O	C(=O)NH	2,4-diCl	H	80	>1000
7	O	C=O	C(=O)NH	2-OCH$_3$,5-NO$_2$	H	70	>1000
8-TF1	O	C=O	C(=O)NH	3,4-diCl	H	50	>1000
9	O	C=O	C(=O)NH	2-OCH$_3$,5-Cl	H	22	>1000



Cpd	R1	R2	R3	Tf IC$_{50}$(µM)	Hu IC$_{50}$(µM)
2	**C=NN=CH**	**p-NO$_2$**	**H**	**240**	**200**
10	C=NN=CH	2-OH,5-Br	H	460	>1000
11	C=NNC(=O)C H$_2$	3,4-diOCH$_3$	Br	320	>1000
12	C=NN=CH	3,4-diCl	Br	300	>1000
13	C=C(CH$_3$)CH$_2$CH$_2$	p-OCH$_3$	H	200	>1000
14	C=NNC(=O)CH$_2$O	p-NO$_2$	H	180	>1000
15	C=NNC(=O)CH(CH$_3$)O	2,4-diCl	Br	85	>1000
16	C=NN=CH	p-Cl	H	10-75	>1000
17	C=NN=CH	p-NO$_2$	H	50	>1000

Table 1. Inhibition of *T. foetus* HGXPRTase (Tf) and human HGPRTase (Hu) by the indol-2-one (isatin) (right) and phthalic anhydride (left) analog series. The two active chemical compounds that arose directly from the docking screen are shown at the top of Table 1 as compounds #1 and #2. The remaining 16 inactive compounds from the docking screen that showed no effect on *T.foetus* HGPRTase up to 1 mM are: from Sigma; acycloguanosine; phenolphthalein monophosphate di-(cyclohexylammonium) salt; periodate-oxidized, borohydride-reduced guanosine; 2',3'-o-p-anisylideneguanosine; from Salor; 2-(3-carboxy-4-hydroxyphenyl)quinoline-4-carboxylic acid; 4,6-phenoxathinedicarboxylic acid; 2-amino-6-benzylthiopurine; 5-amino-3-(4-bromophenyl)-4-oxothieno(3,4-D)pyridineazine-1-carboxylic acid ethyl ester; 2-(5-nitro-2-benzimidazolylimino)-4,5-imidazolidinedione; 4'-methyl-3'-nitrosuccinanilic acid; and from Maybridge; 2-[(1,2,4)triazolo(3,4-C)-(1,2,4)benzotriazin-1-ylsulfanyl]acetic acid; 2-[(carboxyethyl)sulfanyl]benzoic acid; 4-[(anilinocarbothionyl)oxy]-2-oxo-1,2-dihydroquinoline; 5-fluoro-2-(2-hydroxy-5-nitrophenyl)-2,3-dihydro-4(1H)-quinazolinone; O-3-[(6-chloro-3-carbonyl)pyridyl]-5-methyl-3-isoxazolecarbo-hydroximamide; 2-{[3-amino-4-oxo-4H-(1,3,4)thiadiazolo(2,3-C)-(1,2,4)triazin-7-yl]-sulfanyl}acetic acid.

These structures contain an indol-2-one (isatin) and a phthalic anhydride nucleus, respectively, each attached to a nitro-substituted benzene ring. Our modeling results led us to hypothesize that the indole-2-one (Figure 1B) and the phthalic anhydride (Figure 1C) fill the guanine binding pocket, while

the nitro-benzenes fill a hydrophobic groove near the ribose phosphate binding loop of HGXPRTase. This hypothesis predicts competitive enzyme inhibition against guanine and GMP by these two compounds, which was verified in subsequent studies (see below). However, the precise mode of their bindings to the active site will have to be demonstrated by further structural analysis.

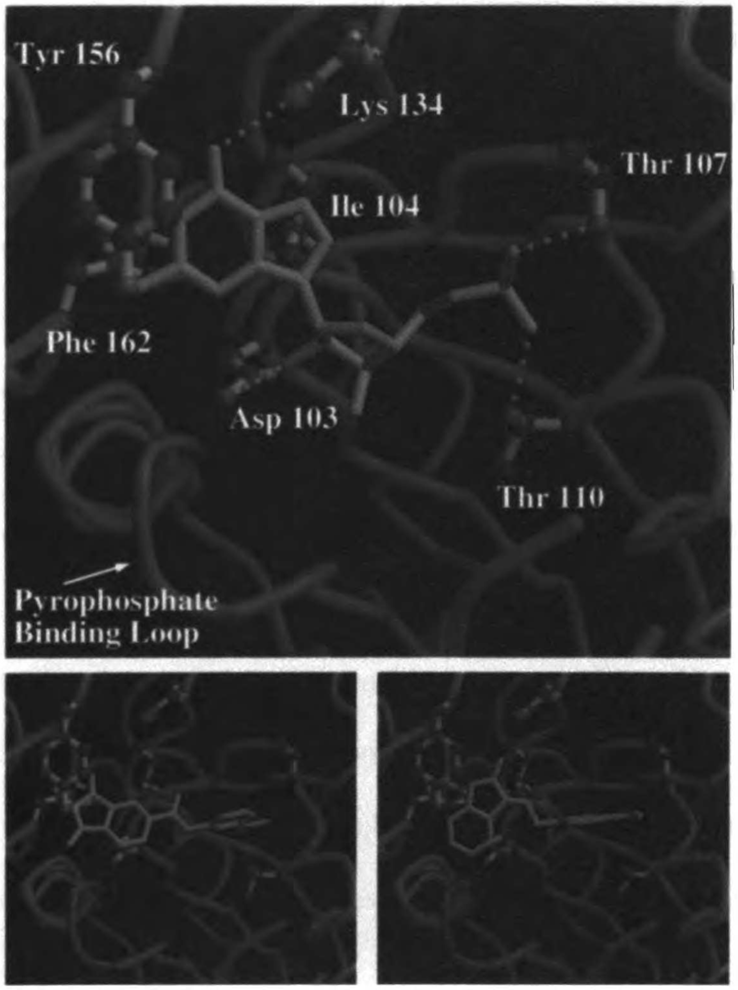


Figure 1. The active site of *T. foetus* HGXPRTase based on the crystal structure of this enzyme (8). Figure **1 top** shows the bound GMP as seen in the crystal structure. Figures **1 bottom left** and **1 bottom right** show the results of our modeling of the phthalic anhydride (Table 1, compound #1) (bottom left) and the indol-2-one (isatin) (Table 1, compound #2) (bottom right) in the active site of *T. foetus* HGXPRTase. The sidechains that are shown are those that interact directly with the GMP. Our modeling suggests that the nitro-benzene moiety of each ligand interacts with Ile 105, Ile 109 and Met 111 (sidechains not shown for clarity).

In order to improve the potency, selectivity, and aqueous solubility of the inhibitors, we used superstructure and similarity searches of the ACD to identify compounds which were chemically similar to the two initial inhibitors (#1 and #2 in Table 1). In the isatin series, we attempted to identify compounds with single changes in either the isatin moiety, the aromatic ring substituents, or in the linker. In the phthalic anhydride series, we attempted to identify compounds with single changes in either the anyhydride, the aromatic ring substituents, or in the linker. From this set of potential inhibitors, we assayed 22 compounds, which, according to our proposed pharmacophore, explored different regions of the HGXPRTase binding site. Eighteen (82%) of these twenty two compounds had IC_{50}'s below the 1 mM screening threshold, and ten (45%) inhibited HGXPRTase with an equal or higher potency than the two original inhibitors. The best compounds demonstrate IC_{50}s more than an order of magnitude lower than those of the lead compounds (Table 1). Furthermore, several of these compounds showed significantly more potent inhibition of the parasite HGXPRTase than of the mammalian HGPRTase (Table 1).

We chose one of the compounds shown in Table 1 for further characterization. 4-(N-(3,4-dichlorophenyl)carbamoyl)phthalic anhydride (compound 8, referred to as TF1) was selected because it was one of the more potent inhibitors of the *T. foetus* HGXPRTase (IC_{50}=50 μM). Furthermore, it did not inhibit human HGPRTase in our screen (IC_{50}>1 mM), and it is sufficiently soluble in aqueous media to allow further testings *in vitro*.

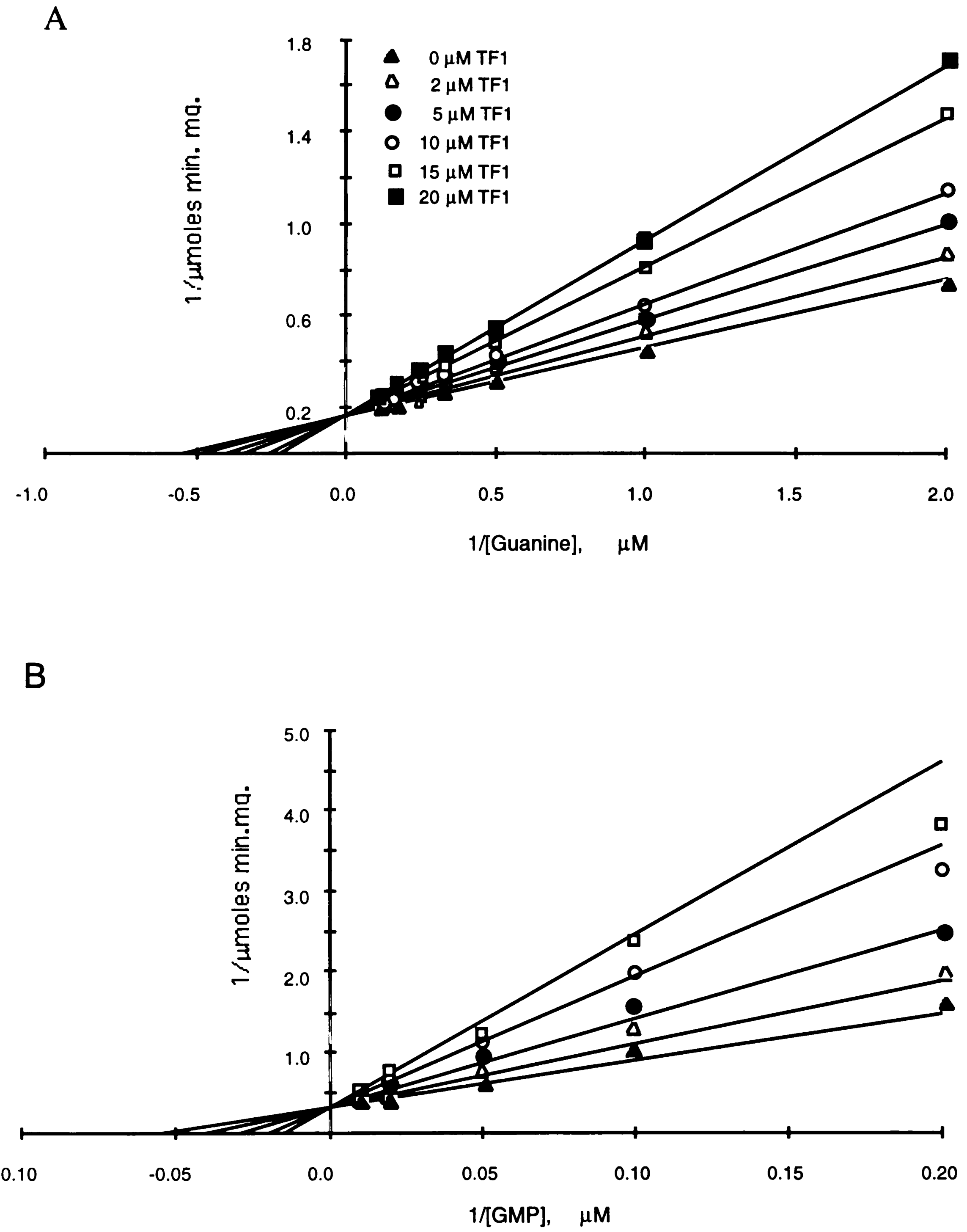


Figure 2. Kinetic data showing that 4-(N-(3,4-dichlorophenyl)carbamoyl) phthalic anhydride (TF1) is a competitive inhibitor of guanine, with a Ki of 13.23±2.03 μM, (**A**); and a competitive inhibitor of GMP, with a Ki of 9.97±1.39 μM, (**B**) in the reverse reaction. Ki is the abscissa intercept of the replot of slope versus concentration of the inhibitor.

A steady-state kinetic analysis using equations 1-3 as described above showed that TF1 is a competitive inhibitor of guanine in the forward reaction, with a *Ki* of 13 μM, as well as a competitive inhibitor of GMP in the reverse reaction, with a *Ki* of 10 μM (Fig. 2). The residual errors for the fit of the data to the models for competitive, non-competitive and uncompetitive inhibition were $5.3\mathrm{x}10^{-6}$, $2.8\mathrm{x}10^{-2}$ and 1.8, respectively, for the forward reaction, and 0.31, 14.8 and 2.6, respectively, for the reverse reaction. These results are consistent with the modeling results, which predicted that the compound would bind to a region of the active site that overlaps with the GMP binding site.

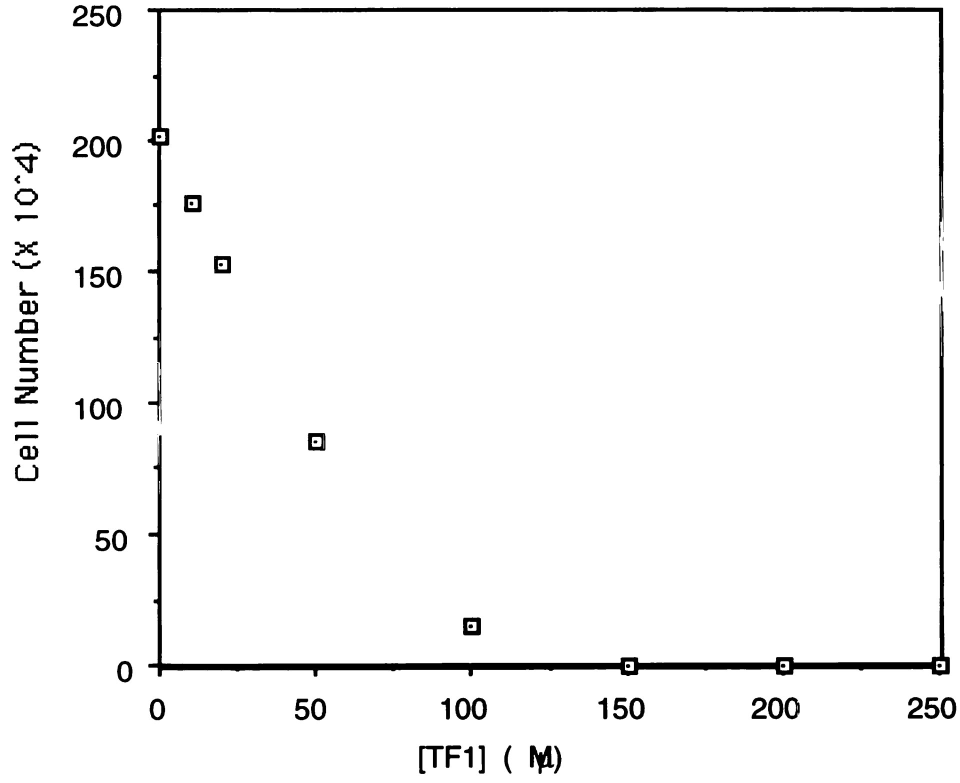


Figure 3. Data showing the effect of 4-(N-(3,4-dichlorophenyl)carbamoyl) phthalic anhydride (TF1) on the *in vitro* growth of *T. foetus*. Based on these data, the IC50 of TF1 is estimated to be approximately 40 μM.

Having shown that TF1 is a competitive inhibitor of *T. foetus* HGXPRTase, we explored the effect of this compound on the growth of *T. foetus* in *in vitro* culture. As

shown in Figure 3, TF1 is a concentration-dependent inhibitor of *T. foetus* growth in culture with an EC50 of approximately 40 μM. Furthermore, the inhibition of *T. foetus* growth by TF1 could be reversed by increasing the concentration of hypoxanthine in the growth medium (Figure 4). The growth inhibition was also associated with decreased incorporation of exogenous radiolabeled guanine into the nucleic acid fraction of *T. foetus* (data not shown). Overall, these data support the conclusion that TF1 inhibits the growth of *T. foetus* in culture by functioning as a competitive inhibitor of the parasite's HGXPRTase.

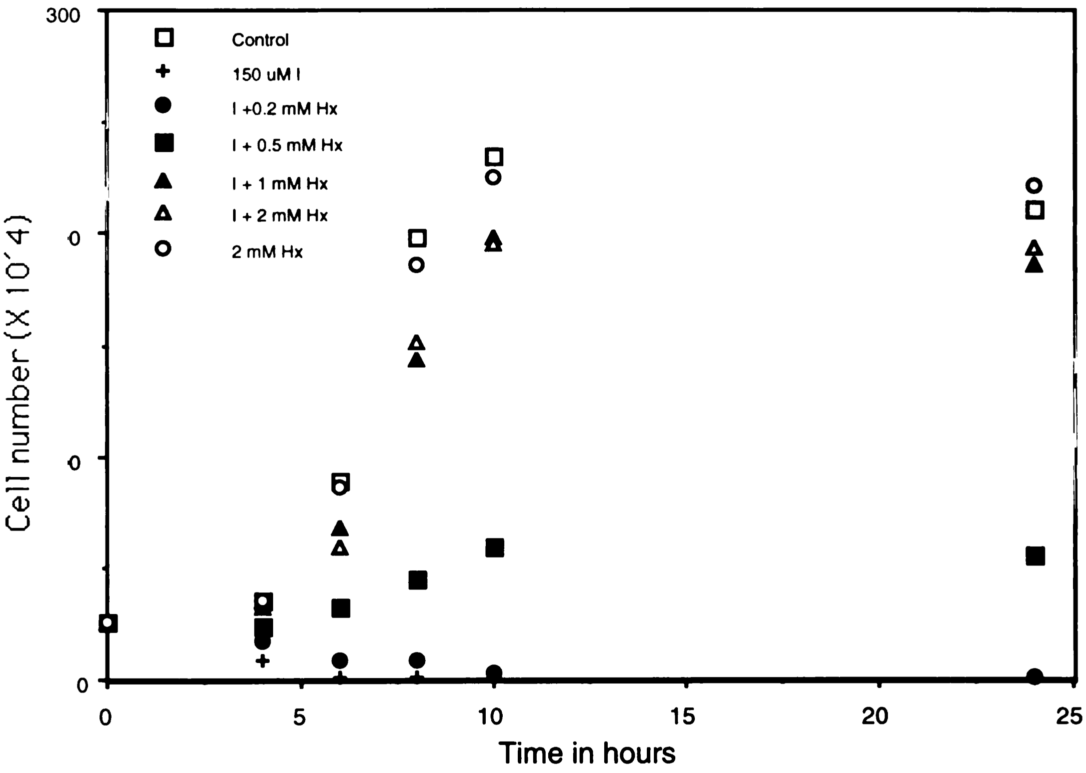


Figure 4. Data showing that the inhibition of 150 μM TF1 on the growth of *T. foetus* can be reversed by the addition of 1 mM exogenous hypoxanthine (Hx). (+):0 Hx + 150 μM TF1; ( ):0.2 mM Hx + 150 μM TF1; ( ): 0.5 mM Hx + 150 μM TF1; ( ): 1.0 mM Hx + 150 μM TF1; (*q*):2.0 mM Hx + 150 μM TF1; (O):2.0 mM Hx + 0 TF1; ( ):0 Hx + 0 TF1. Exogenous guanine reverses the growth inhibition caused by TF1 in a similar manner (data not shown).

Conclusion

We have identified two novel classes of non-purine-analog competitive inhibitors of a purine salvage enzyme. These inhibitors bind to the HGXPRTase approximately as well as any of the enzyme's natural substrates, and inhibit the enzyme better than any other compound that has been found so far (6). The success of this project confirms the conclusions from our previous studies (3), that *T. foetus* HGXPRTase is a good target for anti-tritrichomonal drug design, and that it is possible to use structure-based drug design to identify compounds that inhibit this enzyme but do not substantially inhibit the mammalian homolog. It is likely that the approach taken in this work will have a broad applicability to the design of other antimicrobials.

ACKNOWLEDGMENTS

We thank Jeffrey Champlin, Shinichi Katakura and Jonathan Page for their technical assistance, Donna Hendrix for her thoughtful discussions, and Molecular Design Limited, Tripos Associates Incorporated, and Daylight Information Systems for use of their software. This research was supported by an RO1 grant, AI-19391, from the National Institutes of Health to CCW, and by a Pharmaceutical Chemistry Training grant, GM07075, from the National Institutes of Health to AGS.

Bibliography

1. Wang, C. C. (1984) *Journal of Medicinal Chemistry* *27*, 1-9.

2. Anderson, M. L., Barr, B. C. and Conrad, P. A. (1994) *Veterinary Clinics of North America-Food Animal Practice* *10*, 439-461.

3. Wang, C. C., Verham, R., Rice, A. and Tzeng, S. F. (1983) *Molecular and Biochemical Parasitology* *8*, 325-337.

4. Chin, M. S. and Wang, C. C. (1994) *Molecular and Biochemical Parasitology* *63*, 221-229.

5. Sege-Peterson, K., Chambers, J., Page, T., Jones, O. W. and Nyhan, W. L. (1992) *Human Molecular Genetics* *1*, 427-432.

6. Jadhav, A. L., Townsend, L. B. and Nelson, J. A. (1979) *Biochemical Pharmacology* *28*, 1057-1062.

7. Eads, J. C., Scapin, G., Xu, Y., Grubmeyer, C. and Sacchettini, J. C. (1994) *Cell* *78*, 325-334.

8. Somoza, J. R., Chin, M. S., Focia, P. J., Wang, C. C. and Fletterick, R. J. (1996) *Biochemistry* *35*, 7032-7040.

9. Jones, T. A., Zou, J. Y., Cowan, S. W. and Kjeldgaard, M. (1991) *Acta Cryst.* *A47*, 110-119.

10. Insight II, version 2.3.0, Biosym Technologies, San Diego, CA.

11. Ferrin, T. E., Huang, C. C., Jarvis, L. E. and Langridge, R. (1988) *Journal of Molecular Graphics* *6*, 13-27, 36-37.

12. Sybyl, version 6.3, Tripos Associates, St. Louis, MO.

13. Kuntz, I. D., Blaney, J. M., Oatley, S. J., Langridge, R. and Ferrin, T. (1982) *Journal of Molecular Biology* *161*, 269-288.

Case

Case

14. Meng, E. C., Shoichet, B. K. and Kuntz, I. D. (1992) *Journal of Computational Chemistry* *13*, 505-524.

15. Meng, E. C., Gschwend, D. A., Blaney, J. M. and Kuntz, I. D. (1993) *Proteins: Structure, Function and Genetics* *17*, 268-78.

16. Available Chemicals Directory, version 95.1, Molecular Design Limited Information Systems, San Leandro, CA.

17. Brown, R. D. and Martin, Y. C. (1997) *Journal Of Chemical Information and Computer Sciences* *37*, 1-9.

18. Matter, H. (1997) *Journal Of Medicinal Chemistry* *40*, 1219-1229.

19. Daylight, version 4.51, Daylight Chemical Information Systems, Inc., Santa Fe, NM.

20. Kanaaneh, J., Craig, S. P., III and Wang, C. C. (1994) *Eur. J. Biochem.* *223*, 595-601.

21. Beck, J. T. and Wang, C. C. (1993) *Mol. and Biochem. Parasitol.* *60*, 187-194.

22. Yuan, L., Craig, S. P., McKerrow, J. H. and Wang, C. C. (1992) *Biochemistry* *31*, 806-810.

23. K-CAT, Biometallics, Inc.

24. KinetAsyst, Gauss-Newton Analysis, version II, IntelliKinetics, State College, PA.

25. Cleland, W. W. (1963) *Biochim. Biophys. Acta* *67*, 104-137.

26. Wang, A. L. and Wang, C. C. (1985) *Molecular and Biochemical Parasitology* *14*, 323-335.

27. Scapin, G., Ozturk, D. H., Grubmeyer, C. and Sacchettini, J. C. (1995) *Biochemistry* *34*, 10744-10754.

28. Munagala, N. R., Chin, M. S. and Wang, C. C. (1998) *Biochemistry 37:4045-4051.*

29. Xu, Y., Eads, J., Sacchettini, J. C. and Grubmeyer, C. (1997) *Biochemistry* *36*, 3700-3712.

Prologue to Chapter 7

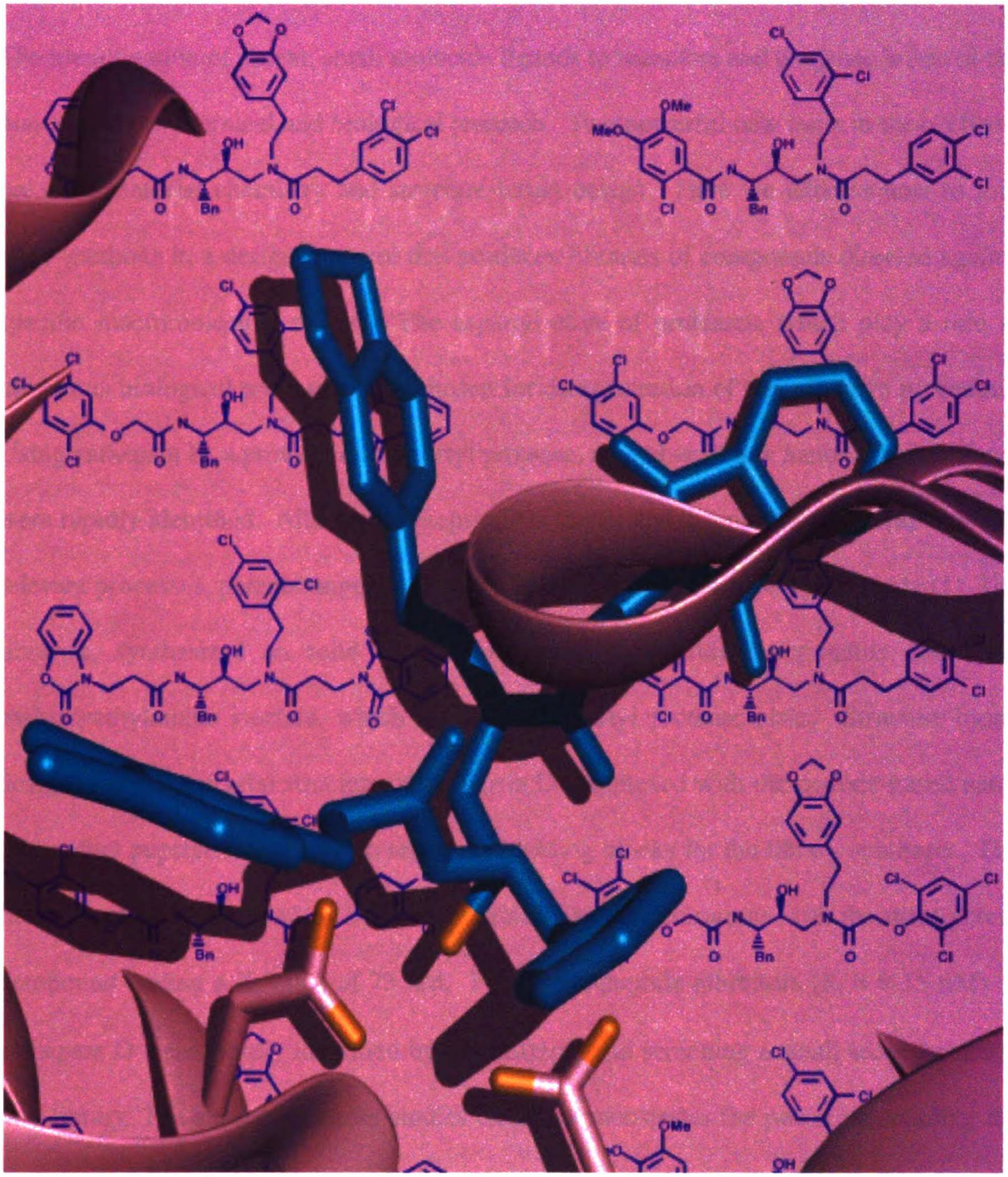
To me, the project presented in this chapter is a prototypical example of the great benefits and huge pitfalls of collaborative work. This project was begun when Jon Ellman (University of California, Berkeley) approached the Kuntz group with the proposal that we should design a library of (hydroxyethyl)amine inhibitors of the aspartyl protease cathepsin D. We designed a head-to-head comparison of a diverse library, which I designed, and a structure-based (directed) library, which Diana Roe designed. Both 1000 compound libraries were synthesized, characterized, and assayed by Ellen Kick. The organization and communication necessary to manage this project was enormous and Ellen and I developed a great friendship through literally *hundreds* of e-mails across the bay. In the end, the project was quite successful, yielding the best non-peptide cathepsin D inhibitors known. It was only after the final draft of the paper (to be submitted to Science) was written that I discovered a systematic error had been made in constructing each of molecules tested in designing the directed library. An extra methylene carbon had been added to the proximal end of every side-chain before it was attached to the scaffold, yet despite this we had achieved excellent results! The manner in which this mistake was resolved is a tribute to the ethics of everyone involved. The entire directed library was reanalyzed. Rather than using visual inspection (the norm), which might have been biased by our prior knowledge of the system, Ellen and I selected compounds from the directed library using a clustering of the best-scoring compounds. Finally, Ellen synthesized, characterized, and assayed a second 1000 compound directed library. As you will see, the inhibition results from this second library were also outstanding. It was a gratifying experience to participate in this collaboration, and I am particularly grateful to my friends Ellen Kick and Diana Roe.

Chapter 7: Structure-Based Design and Combinatorial Chemistry Yield Low Nanomolar Inhibitors of Cathepsin D.

by

Ellen K. Kick, Diana C. Roe, A. Geoffrey Skillman, Guangcheng Liu, Todd J. A. Ewing, Yaxiong Sun, Irwin D. Kuntz and Jonathan A. Ellman

Reproduced with permission from
Chemistry and Biology, April 1997, 4(4):297-307.



Abstract

The identification of potent, small molecule ligands to receptors and enzymes is one of the major goals of chemical and biological research. Two powerful new tools in these efforts are combinatorial chemistry and structure-based design. Here we address how to join these methods in a design protocol that produces libraries of compounds directed against specific macromolecular targets. The aspartyl class of proteases, which play a role in numerous biological processes, was chosen for demonstration of this effective procedure. Using cathepsin D, a prototypical aspartyl protease, a number of low nanomolar inhibitors were rapidly identified. Although cathepsin D is implicated in a number of therapeutically relevant processes, potent nonpeptide inhibitors have not been reported previously.(1) The libraries, synthesized on solid support, displayed nonpeptide functionality about the hydroxyethylamine isostere, which targets the aspartyl protease class. Structure-based design, using the crystal structure of cathepsin D complexed with the peptide-based natural product pepstatin, was used to select the building blocks for the library synthesis. The library yielded a "hit rate" of 6-7% at 1 Minhibitor concentrations with the most potent compound having a K_i value of 73 nM. Potent, nonpeptide inhibitors (K_i = 9-15 nM) of cathepsin D were rapidly identified by synthesizing and screening a small second generation library. The success of these studies clearly demonstrates the power of coupling the complementary methods of combinatorial chemistry and structure-based design. We anticipate that the general approaches described here will be successful for other members of the aspartyl protease class and for many other enzyme classes.

Introduction

A cherished goal of chemists is to design and synthesize compounds with a specific set of properties. This goal is particularly urgent in biological and medicinal chemistry as a part of the drug discovery process. Two powerful new tools in this effort are structure-based design (2- 3) and combinatorial chemistry (4- 5). Structure-based design uses information gleaned from crystallographic and magnetic resonance experiments on a target macromolecule, frequently an enzyme, to guide the selection or design of inhibitors. Computation plays a major role (3- 6). Combinatorial chemistry is based on general chemical transformations that allow different building blocks to be combined in high yield. These transformations can be performed in parallel to synthesize libraries of related compounds rapidly and efficiently (4- 5). Nonetheless, the discovery of a new lead compound or the improvement of the properties of an existing lead are still demanding tasks. The issue we address in this paper is how to join computational and combinatorial methods in a design protocol that produces libraries of compounds directed against specific macromolecular targets.

Combinatorial approaches to ligand identification initially focused on biopolymer libraries prepared by either chemical or biological methods (7). For these libraries, all possible combinations of the building blocks are typically used since there are only four natural nucleotide building blocks for nucleic acid libraries and 20 proteinogenic amino acid building blocks for peptide libraries. Both the structures of the compounds and the

theoretical number of compounds in the library are determined by the length of the biopolymer chain. Recently, considerable efforts have been directed toward the preparation of libraries of compounds that encompass a wider spectrum of chemical transformations, leading to compounds with a broader range of properties than found in peptides or oligonucleotides (4- 5). These new approaches introduce significant challenges in library design.

A crucial element of any library design is the procedure for selecting which compounds to synthesize. This includes the choice of the scaffold, the basic reactions and the nature of the building blocks. If the building blocks are readily available components such as amines, aldehydes or carboxylic acids, the number of potential compounds to be considered can be quite large. For example, combining three building blocks with thousands of components at each position leads to over 1 billion compounds. While different strategies have distinct practical limits, typically one is prepared to synthesize only thousands of spatially separate compounds and tens of millions of compounds in mixtures. Furthermore, evaluation and deconvolution of a very large library become rate-limiting activities (8). Thus, there would be significant advantages to a method of reducing the synthetic effort to a small subset of compounds biased towards the desired properties.

How can the potential choices be efficiently reduced? Two standard strategies are diversity selection and directed selection. Diversity approaches attempt to maximize the sampling of chemical and biological properties given a fixed number of compounds (9). In directed libraries the size and often the diversity of the library is reduced by selecting

those building blocks that are predicted to have favorable interactions with the target, or by eliminating candidates that are believed to have unfavorable interactions. A directed library can be based on substrate preferences, information about known inhibitors, or, in the work described here, an assessment of the potential interaction of specific functional groups with the target. Both diversity and directed strategies permit a multistage attack with secondary libraries generated from active compounds found in the first round.

The development of general and efficient approaches to identify small, nonpeptidic inhibitors of proteases continues to be of interest because proteases have important roles in therapeutically relevant processes (10-13). Proteases have also proven to be excellent targets for structure-based approaches (14- 15). Our target, cathepsin D, has been implicated in breast cancer tumor metastasis, melanoma metastasis (16) and Alzheimer's disease (17-18). Here we describe the efficient development of a combinatorial library with functionality selected using structure-based design. These studies resulted in the identification of potent inhibitors of cathepsin D, that do not contain amino acids and have molecular weights under 800 daltons.

Material and Methods

Directed Library Design

The structure-based design process began with coordinates for pepstatin in a complex with cathepsin D (20). The scaffold is identical to pepstatin on the P_1-P_3 side, but differs on the $P_{1'}$-$P_{3'}$ side and cannot form the same hydrogen bonds with the enzyme (Fig.

3a). Thus, we used the pepstatin positions for the P_1-P_3 side and systematically rotated the three scaffold torsion angles on the $P_{1'}$-$P_{3'}$ side. Each rotation was followed by energy minimization within the cathepsin D active site, using the AMBER (33) force field in the program Sybyl, a molecular modeling software package from Tripos Associates, St. Louis, MO. During minimization, the enzyme was kept rigid but full flexibility of the scaffold was allowed. Both *S* and *R* epimers, structures **1** and **2**, were modeled using methyl groups for each of the R_1-R_4 groups. The conformational energies of the *R* epimers were generally ca. 2 kcal higher than for *S* epimers, leading us to predict that the *S* epimers would bind more tightly than the *R* epimers. All minimized conformations of *S* epimers within a 2 kcal/mol range were collected and clustered into four families based on geometric similarity (Fig. 3b). A benzyl group was added to each family at the R_4 position. The processed list of compounds from the ACD was passed through Sybyl to obtain Gasteiger and Marsili partial atomic charges for each component (34- 35). To reduce the computational time for searching the components, compounds with more than 4 torsional bonds were identified and removed. A new feature for our BUILDER molecular modeling program (24- 25), called BUILDERopt (26), was used to position each of the R_1, R_2, and R_3 components onto the scaffold and to perform a full conformational search for the torsion angles of the substituent at 15 degree increments. In order to reduce the combinatoric problem, the R_1, R_2, and R_3 components were examined independently, but a probability-based clash grid was constructed to identify R_1 and R_2 conformations that might overlap. For example, if an R_1 conformation clashed with more than 50% of the R_2 components, that conformation was discarded. Each rotation was then examined for intramolecular clashes with the scaffold and overlap with cathepsin D. Each accepted conformation was

rigid-body minimized (36) and scored with a force-field grid (37). The total computer time required to evaluate all torsion angles for all sidechains attached to four different scaffold conformations was 16 hours on a Silicon Graphics Iris R4400. The fifty best scoring components for all families were merged for each of the three variable positions, and sorted by overall lowest score. Components with cost above $35/gm were removed, leaving 34, 35, and 41 components at R_1, R_2 and $R_{3,}$ respectively. Each remaining component was structurally fingerprinted (28) and hierarchically clustered (similarity cutoff = 0.63) (38) using the Tanimoto similarity metric (29- 30). For R_1, R_2, and $R_{3,}$ the ten best scoring components from unique clusters were selected for the directed library.

Diverse Library Design

Components from the original ACD list were clustered with the Jarvis-Patrick algorithm (27) using the Daylight connectivity measure of similarity (28) and a binary Tanimoto metric (29- 30). In the Jarvis-Patrick method, two compounds are placed in the same cluster if they; 1) are neighbors of one another, and 2) share at least p neighbors from a list of q nearest neighbors, where p and q are adjustable parameters. The compound nearest the cluster centroid was chosen as the cluster representative.

The R_1 (amine) components were clustered directly as the primary amines. The R_2 and R_3 acylating agents were each attached to a portion of the scaffold before clustering to yield the proper chemical context at the linkage site. The first round of clustering yielded 47, 154, and 162 clusters using p/q = 4/11, p/q = 4/12, and p/q = 4/12 for R_1, R_2, and R_3, respectively. The representative R_2 and R_3 components were clustered a second time (p/q = 4/7 for R_2 and p/q = 4/7 for R_3), resulting in 23 R_2 and 35 R_3 components. We note that

it is not practical to condense a large number of compounds into an arbitrarily small number of clusters because the cluster membership can become very diverse. Final selection of ten compounds from each list was based upon: size, cost, availability, and synthetic feasibility. Additionally, we sought a balance of functional groups for each set of sidechains. A comparison of the directed and diverse libraries (Figs. 4, 5) shows the much greater range of functionality spanned in the diverse library.

Library Synthesis

We have previously reported the optimization of the solid-phase synthesis sequence to prepare the hydroxyethylamine inhibitors and the demonstration of reaction generality (23). Further testing was performed to establish that the different functionality to be displayed at R_1, R_2 and R_3 would be successfully incorporated into the potential inhibitors. First, all the amines and acylating agents to be incorporated in both the diverse and directed libraries were treated with trifluoroacetic acid for 2 h at room temperature to ensure stability to the support-cleavage conditions, by far the harshest reaction conditions in the synthesis sequence. Second, components that might pose difficulties on chemical or steric grounds were evaluated by trial syntheses. Five amines and four carboxylic acids that did not provide the expected final compound in high yields or purity were discarded. The following amines and acylating agents were successfully tested in the synthesis sequence: R_1 = B, C, E, F, a, e, h, i, j; R_2 = B, C, D, E, H, a, e, f; R_3 = A, D, E, H, a, b, e, g, h, i. The remaining components were assumed to be compatible with the synthesis sequence.

The library synthesis was performed on polystyrene beads (20-40 mesh) prepared in

our laboratory. The library was synthesized in a spatially separate array using a 96-well filter apparatus. Transfer of the resin to the individual wells was performed using an isopycnic mixture of *N,N*-dimethylformamide (DMF) and 1,2-dichloroethane. Incorporation of R_1 was carried out using 1.0 M free amine in *N*-methylpyrrolidinone (NMP) at 80 Cfor 36 h. Incorporation of R_2 was carried out using stock solutions of 0.3 M carboxylic acid, 0.3 M benzotriazole-1-yl-oxy-tris-pyrrolidino-phosphonium hexafluorophosphate (PyBOP), 0.3 M 7-aza-1-hydroxybenzotriazole (HOAt), and 0.9 M *i*Pr_2EtN in NMP overnight. The coupling reactions were performed twice to ensure that complete coupling had occurred. After azide reduction with $SnCl_2$, PhSH and Et_3N, incorporation of R_3 was carried out as reported above for R_2. Carboxylic acid R_2 = E was coupled using *O*-(7-azabenzotriazol-1-yl)-1,1,3,3-tetramethyl-uronium hexa-fluorophosphate (HATU) instead of PyBOP due to formation of a precipitate under the standard coupling procedure. The isocyanate R_2 = b was coupled at 0.3 M in NMP overnight, and the sulfonyl chlorides R_2 = e and R_3 = c were coupled at 0.3 M in NMP that was 0.9 M in *i*Pr_2EtN. Cleavage of the material from support was achieved by subjecting the resin to 95:5 trifluoroacetic acid : H_2O for 30 min, followed by rinsing the resin and concentration of the filtrates using a Jouan 10.10 centrifugation concentrator. Toluene was added to form an azeotrope with trifluoroacetic acid during the concentration step. After concentration, the libraries were stored at -20 C.

Compounds from each library, picked by random number generation, were analyzed by mass spectrometry in a matrix of a-cyano cinnamic acid on a Perseptive Biosystems MALDI instrument. For the diverse library the expected molecular ion peaks were

observed for 46 of 49 compounds (poor ionization was obtained for the other three). Molecular ion peaks were obtained for 44 of 49 compounds from the directed library. In addition, the synthesis has been validated by the reasonable correlation of the approximate IC_{50} values of the crude material from the libraries with purified material that was synthesized on large scale for a number of compounds (see Table **2**).

High throughput Cathepsin D Assay

A fluorometric high through-put assay for activity toward human liver cathepsin D (Calbiochem) was performed in 96-well microtiter plates (31). The peptide substrate (Ac-Glu-Glu(Edans)-Lys-Pro-Ile-Cys-Phe-Phe-Arg-Leu-Gly-Lys(Methyl Red)-Glu-NH_2) used in the assay has been previously reported (Km = 6 M)(20). The assay was performed in DYNATECH Microfluor fluorescence microtiter plates, and readings were taken on a Perkin-Elmer LS-50B with an attached 96-well plate reader. The excitation wavelength was 340 nm. A 340 nm interference filter (Hoya, U-340) for excitation and a 430 nm cut-off filter for emission were used. For the microtiter-based assays the substrate concentration was 5 Mand the cathepsin D concentration was 3 nM in a 0.1 M formic acid buffer (pH = 3.7). DMSO, 10%, was used to ensure complete dissolution of the inhibitors. The fluorescent unit readings were taken at three time points within the linear region of the substrate cleavage, and percent activity of the enzyme was determined by comparing the change of fluorescent units (FU) for each well to the average change in FU for six control wells without inhibitor. Each library was screened at approximately 1 Minhibitor with the concentration based on the assumption that 50% of the theoretical yield was obtained for each inhibitor. All wells that showed <50% cathepsin D activity were screened at subse-

quent three-fold dilutions. All active compounds that showed <60% enzyme activity at 1 M or lower inhibitor concentrations were assayed in duplicate.

Synthesis of Inhibitors

Several of the most potent compounds from the designed and diverse libraries were synthesized on 30 mg scale on the solid-support following the previously reported method (23). These compounds were purified by column chromatography, and characterized by ^{1}H NMR and either mass spectrometry or elemental analysis. The characterization data are listed below after the appropriate compound code.

EHA. ^{1}H NMR (300 MHz, $CDCl_3$) δ 2.43 (m, 2H), 2.55 (t, $J = 7.2$ Hz, 2H), 2.62-2.83 (m, 4H), 2.88 (dd, $J = 2.3$, 14.3 Hz, 1H), 3.27-3.36 (m, 2H), 3.48 (dd, $J = 8.7$, 14.3 Hz, 1H), 3.70 (d, $J = 8.0$ Hz, 1H), 3.85 (t, $J = 7.2$ Hz, 2H), 4.03-4.14 (m, 3H), 5.91 (s, 1H), 5.92 (s, 1H), 6.44 (dd, $J = 1.6$, 7.9 Hz, 1H), 6.52 (d, $J = 1.6$ Hz, 1H), 6.66 (d, $J = 7.9$ Hz, 1H), 7.03-7.26 (m, 9H), 7.70 (dd, $J = 3.0$, 5.4 Hz, 2H), 7.81 (dd, $J = 3.0$, 5.4 Hz, 2H). LRMS (MALDI-TOF) α-cyano-4-hydroxycinnamic acid matrix: mass calcd. for $C_{40}H_{39}N_4O_9$ (MH^+) 719.2, found 718.7.

EHD. ^{1}H NMR (300 MHz, $CDCl_3$) δ 2.43-2.54 (m, 2H), 2.76 (t, $J = 7.6$ Hz, 2H), 2.92-2.98 (m, 2H), 3.02 (dd, $J = 1.9$, 14.2 Hz, 1H), 3.34 (m, 2 H), 3.68 (dd, $J = 8.9$, 14.2 Hz, 1H), 3.82-4.01 (m, 3H), 4.19 (apparent q, $J = 8.0$ Hz, 1H), 4.40 (d, $J = 14.3$ Hz, 1H), 4.51 (d, $J = 14.3$ Hz, 1H), 5.91 (d, $J = 1.5$ Hz, 1H), 5.92 (d, $J = 1.5$ Hz, 1H), 6.45 (dd, $J = 1.6$, 7.9 Hz, 1H), 6.53 (d, $J = 1.6$ Hz, 1H), 6.66 (d, $J = 7.9$ Hz, 1H), 6.95 (s, 1H), 7.17-7.28 (m,

5H), 7.49 (s, 1H), 7.71 (dd, J = 3.1, 5.5 Hz, 2H), 7.82 (dd, J = 3.1, 5.5 Hz, 2H). MALDIMS: m/z 767.1 (M^+ + H). LRMS (MALDI-TOF) α-cyano-4-hydroxycinnamic acid matrix: mass calcd. for $C_{38}H_{35}N_3O_8Cl_3$ (MH^+) 766.1, found 767.1.

EHJ. ^{1}H NMR (300 MHz, $CDCl_3$) δ 2.37-2.56 (m, 2H), 2.63 (t, J = 6.8, 2H), 2.98-3.01 (m, 2H), 3.14 (d, J = 11.2, 1H), 3.41 (m, 2H), 3.82 (s, 3H), 3.86 (s, 3H), 3.76-3.90 (m, 4H), 4.33 (apparent q, J = 8.0 Hz, 1H), 5.87 (d, J = 1.3 Hz, 1H), 5.89 (d, J = 1.3 Hz, 1H), 6.45 (dd, J = 1.6, 7.9 Hz, 1H), 6.49 (d, J = 1.6 Hz, 1H), 6.63 (d, J = 7.9 Hz, 1H), 6.77 (d, J = 9.2 Hz, 1H), 6.78 (d, J = 8.7 Hz, 1H), 7.18 (d, J = 8.7 Hz, 1H), 7.24-7.28 (m, 5H), 7.69 (dd, J = 3.0, 5.4 Hz, 2H), 7.79 (dd, J = 3.0, 5.4). LRMS (MALDI-TOF) α-cyano-4-hydroxycinnamic acid matrix: mass calcd. for $C_{39}H_{39}N_3O_9Cl$ (MH^+) 728.2, found 727.9.

EFA. ^{1}H NMR (300 MHz, $CDCl_3$) δ 2.10-2.33 (m, 2H), 2.49 (t, J = 6.7 Hz, 2H), 2.47-2.86 (m, 7H), 3.17-3.33 (m, 2H), 3.58-3.69 (m, 2H), 4.02-4.14 (m, 3H), 5.90 (s, 2H), 6.38 (dd, J = 1.6, 7.9 Hz, 1H), 6.47 (d, J = 1.6 Hz, 1H), 6.67 (d, J = 7.9 Hz, 1H), 6.88 (dd, J = 2.0, 8.2 Hz, 1H), 7.02-7.26 (m, 10H), 7.29 (d, J = 8.2 Hz, 1H). LRMS (MALDI-TOF) α-cyano-4-hydroxycinnamic acid matrix: mass calcd. for $C_{38}H_{37}N_3O_7Cl_2$ (MH^+) 718.2, found 719.0.

FHA. ^{1}H NMR (300 MHz, $CDCl_3$) δ 2.46 (m, 9H), 2.94 (dd, J = 2.3, 14.3 Hz, 1H), 3.33 (apparent q, J = 7.0 Hz, 1H), 3.46 (dd, J = 8.8, 14.3 Hz, 1H), 3.73 (d, J = 8.2 Hz, 1H), 3.89 (t, J = 7.3 Hz, 2H), 4.02-4.16 (m, 3H), 6.43 (d, J = 9.2 Hz, 1H), 6.98-7.26 (m, 11H),

7.32 (d, J = 2.1 Hz, 1H), 7.70 (dd, J = 3.2, 5.4 Hz, 2H), 7.82 (dd, J = 3.2, 5.4 Hz, 2H). LRMS (MALDI-TOF) α-cyano-4-hydroxycinnamic acid matrix: mass calcd. for $C_{39}H_{36}N_4O_7Cl_2$ (MH^+) 742.2, found 743.0.

fbb. ^{1}H NMR (300 MHz, $CDCl_3$) δ 0.64-1.89 (m, 10H), 2.67 (t, J = 6.4 Hz, 2H), 2.79 (t, J = 6.7 Hz, 1H), 2.86-2.96 (m, 3H), 3.26 (t, J = 6.4 Hz, 2H), 3.43 (t, J = 6.8 Hz, 1H), 3.77-3.87 (m, 5H), 4.06 (d, J = 7.2 Hz, 1H), 4.09 (d, J = 7.2 Hz, 1H), 4.63 (s, 2H), 6.45 (d, J = 9.2 Hz, 1H), 6.64-6.82 (m, 3H), 7.18-7.31 (m, 6H). FABHRMS: 613.2519 (M^+ + H, $C_{31}H_{41}N_4O_5S_2$ requires 613.2518).

fdb. ^{1}H NMR (300 MHz, $CDCl_3$) δ 0.87 (t, J = 6.8 Hz, 4H), 1.20-1.26 (m, 8H), 1.47-1.58 (m, 2H), 1.95-2.14 (m, 2H), 2.67 (t, J = 7.2 Hz, 2H), 2.90-3.01 (m, 2H), 3.34-3.55 (m, 2H), 3.64-3.90 (m, 2H), 3.79 (s, 3H), 4.08 (m, 2H), 4.63 (s, 1H), 6.40 (d, J = 9.0 Hz, 1H), 6.62-6.78 (m, 3H), 7.18-7.31 (m, 6H). FABHRMS: 600.2555 (M^+ + H, $C_{31}H_{42}N_3O_5S_2$ requires 600.2566).

EHM. ^{1}H NMR (300 MHz, $CDCl_3$) δ 2.48 (m, 2H), 2.60 (m, 2H), 2.94 (m, 2H), 3.03 (dd, J = 1.7, 14.8 Hz, 1H), 3.37 (m, 2H), 3.68 (dd, J = 8.9, 14.2 Hz, 1H), 3.83-4.00 (m, 3H), 4.20 (apparent q, J = 8.2 Hz, 1H), 4.42 (d, J = 14.5 Hz, 1H), 4.51 (d, J = 14.5 Hz, 1H), 5.91 (s, 2H), 6.46 (dd, J = 1.6, 7.9 Hz, 1H), 6.53 (d, J = 1.6 Hz, 1H), 6.65 (d, J = 7.9 Hz, 1H), 6.76 (d, J = 8.8 Hz, 1H), 7.17 (dd, J = 2.6, 8.8 Hz, 1H), 7.20-7.26 (m, 5H), 7.40 (d, J = 2.6 Hz, 1H), 7.71 (dd, J = 3.1, 5.4 Hz, 2H), 7.83 (dd, J = 3.1, 5.4 Hz, 2H). LRMS

(MALDI-TOF) α-cyano-4-hydroxycinnamic acid matrix: mass calcd. for $C_{38}H_{35}N_3O_8Cl_2$ (MH^+) 732.2, found 732.5.

EHO. ^{1}H NMR (400 MHz, $CDCl_3$, CD_3OD) δ 2.47 (m, 2H), 2.62 (t, *J* = 7.1 Hz, 2H), 2.86-2.99 (m, 3H), 3.32-3.41 (m, 2H), 3.68 (dd, *J* = 8.8, 14.4 Hz, 1H), 3.79 (d, *J* = 8.6 Hz, 1H), 3.88-3.95 (m, 2H), 4.19 (apparent q, *J* = 8.1 Hz, 1H), 4.38 (d, *J* = 14.7 Hz, 1H), 4.47 (d, *J* = 14.7 Hz, 1H), 4.83 (s, 1H), 5.92 (d, *J* = 1.4 Hz, 1H), 5.93 (d, *J* = 1.4 Hz, 1H), 6.45 (dd, *J* = 1.5, 7.9 Hz, 1H), 6.53 (d, *J* = 1.5 Hz, 1H), 6/67 (d, *J* = 7.9 Hz, 1H), 6.77 (dd, *J* = 2.9, 8.8 Hz, 1H), 6.91 (d, *J* = 9.5 Hz, 1H), 7.03 (d, *J* = 2.9 Hz, 1H), 7.20-7.28 (m, 5H), 7.35 (d, *J* = 8.8 Hz, 1H), 7.72 (dd, *J* = 3.0, 5.4 Hz, 2H), 7.83 (dd, *J* = 3.0, 5.4 Hz, 2H). LRMS (MALDI-TOF) α-cyano-4-hydroxycinnamic acid matrix: mass calcd. for $C_{38}H_{35}N_3O_8Cl_2$ (MH^+) 732.2, found 732.3.

EHR. ^{1}H NMR (300 MHz, $CDCl_3$, CD_3OD) δ 2.40 (m, 2H), 2.58 (t, *J* = 7.0 Hz, 2H), 2.82 (m, 2H), 2.98 (dd, *J* = 3.4, 14.1 Hz, 1H), 3.31-3.46 (m, 3H), 3.74-3.88 (m, 3H), 4.13 (m, 1H), 4.32 (d, *J* = 14.8 Hz, 1H), 4.41 (d, *J* = 14.8 Hz, 1H), 5.84 (s, 2H), 6.41 (dd, *J* = 1.6, 7.9 Hz, 1H), 6.49 (d, *J* = 1.6 Hz, 1H), 6.60 (d, *J* = 7.9 Hz, 1H), 6.74 (dd, *J* = 1.9, 8.3 Hz, 1H), 6.89 (m, 1H), 6.94 (d, *J* = 8.0 Hz, 1H), 7.07-7.22 (m, 6H), 6.67 (dd, *J* = 3.1, 5.4 Hz, 2H), 7.78 (dd, *J* = 3.1, 5.4 Hz, 2H). LRMS (MALDI-TOF) α-cyano-4-hydroxycinnamic acid matrix: mass calcd. for $C_{38}H_{36}N_3O_8Cl$ (MH^+) 698.2, found 698.0.

EHS. ^{1}H NMR (300 MHz, $CDCl_3$) δ 2.49 (m, 2H), 2.65 (t, *J* = 7.1 Hz, 2H), 3.02 (m, 2H),

3.15 (d, J = 12.7 Hz, 1H), 3.41 (m, 2H), 3.75-3.99 (m, 4H), 4.36 (apparent q, J = 8.3 Hz, 1H), 5.90 (d, J = 1.5 Hz, 1H), 5.91 (d, J = 1.5 Hz, 1H), 6.47 (dd, J = 1.6, 7.9 Hz, 1H), 6.55 (d, J = 1.6 Hz, 1H), 6.66 (d, J = 7.9 Hz, 1H), 6.71 (d, J = 9.3 Hz, 1H), 6.90 (s, 1H), 6.98 (s, 1H), 7.17-7.28 (m, 5H), 7.70 (dd, J = 3.1, 5.6 Hz, 2H), 7.82 (dd, J = 3.1, 5.6 Hz, 2H). LRMS (MALDI-TOF) α-cyano-4-hydroxycinnamic acid matrix: mass calcd. for $C_{39}H_{38}N_3O_9Br$ (MH^+) 772.2, found 772.5.

FHO. ^{1}H NMR (400 MHz, $CDCl_3$) δ 2.65 (t, J = 7.2 Hz, 2H), 2.82 (t, J = 7.6 Hz, 2H), 2.91 (m, 2H), 3.07 (dd, J = 2.2, 14.3 Hz, 1H), 3.34-3.49 (m, 2H), 3.63 (dd, J = 8.9, 14.3 Hz, 1H), 3.83 (d, J = 8.4 Hz, 1H), 3.94 (m, 2H), 4.20 (apparent q, J = 8.0 Hz, 1H), 4.38 (d, J = 14.8 Hz, 1H), 4.46 (d, J = 14.8 Hz, 1H), 6.76 (dd, J 3.0, 8.9 Hz, 1H), 6.92 (d, J = 9.4 Hz, 1H), 7.00 (d, J = 8.2 Hz, 1H), 7.02 (d, J = 3.0 Hz, 1H), 7.15 (dd, J = 2.1, 8.2 Hz, 1H), 7.18-7.27 (m, 5H), 7.34 (d, J = 8.9 Hz, 1H), 7.73 (dd, J = 3.0, 5.4 Hz, 2H), 7.84 (dd, J = 3.0, 5.4 Hz, 2H). FABHRMS: *m/e* 756.1204 (M^+ + H, $C_{37}H_{33}N_3O_6Cl_4$ requires 756.1202).

UHD. ^{1}H NMR (300 MHz, $CDCl_3$) δ 2.29 (s, 3H), 2.38-2.57 (m, 2H), 2.68 (t, J = 7.2 Hz, 2H), 2.93 (m, 2H), 3.03 (dd, J = 1.6, 14.2 Hz, 1H), 3.34-3.48 (m, 2H), 3.71 (dd, J = 9.0, 14.2 Hz, 1H), 3.81-3.91 (m, 3H), 4.19 (apparent q, J = 8.1, 1H), 4.41 (d, J = 14.3, 1H), 4.50 (d, J = 14.3 Hz, 1H), 6.91-6.98 (m, 2H), 7.05 (d, J = 8.3 Hz, 2H), 7.17-7.28 (m, 5H), 7.50 (s, 1H), 7.67 (s, 1H), 7.71 (dd, J = 3.0, 5.5 Hz, 2H), 7.83 (dd, J = 3.0, 5.5 Hz, 2H). LRMS (MALDI-TOF) α-cyano-4-hydroxycinnamic acid matrix: mass calcd. for

$C_{38}H_{36}N_3O_6Cl_3$ (MH^+) 736.2, found 736.3.

Cathepsin D Assay

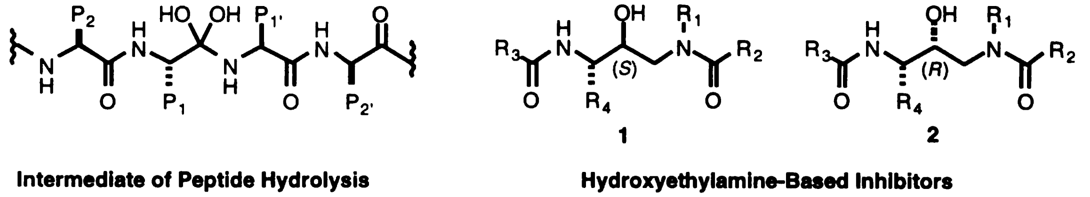
The cathespin D assay for the compounds that had been fully characterized was performed in a quartz cuvette with a Perkin-Elmer LS-50B spectrometer. The substrate concentration was 2.5 Mand the cathepsin D concentration was 10 nM. Inhibition constants (K_i) were determined from IC_{50} values taken from plots of V_i/V_o versus inhibitor concentration, where V_o is the velocity in absence of the inhibitor and V_i is the velocity with inhibitor. Since the substrate concentration is significantly below K_m, the IC_{50} values were converted to K_i by the equation $K_i = (IC_{50} - E_t/2)$, where E_t = enzyme concentration (39).

Results and discussion

Specific Approach

One powerful strategy to target an enzyme class is to incorporate a stable mimetic or isostere of the transition state or of an intermediate of the enzyme-catalyzed reaction (19). The libraries for potential cathepsin D inhibitors are based upon the well-known hydroxyethylamine isostere (Fig. 1). For our initial libraries, the P_1 side chain (R_4) is held constant as the benzyl substituent, based on the X-ray crystallographic structure of cathepsin D complexed with the natural peptide inhibitor pepstatin (20), and upon inhibition constants of peptide-based inhibitors (21- 22).

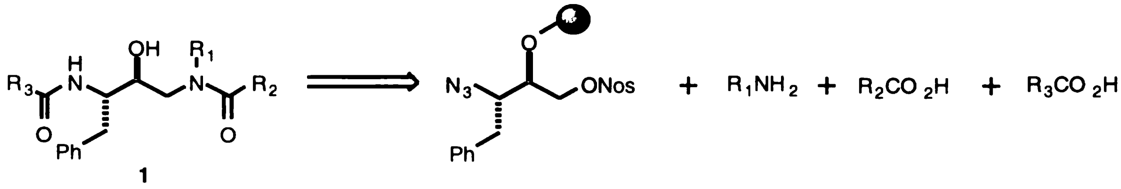
Figure 1


Intermediate of Peptide Hydrolysis
Hydroxyethylamine-Based Inhibitors
(S)
(R)
1
2

Mechanism-based inhibitor design.

In a pilot study both *S* and *R* epimers at the hydroxyl carbon (structures **1** and **2**) were prepared since both diastereomers have been found in potent inhibitors of other aspartic acid proteases (19). Because inhibition at 1 Mwas only found with compounds of scaffold **1** in the pilot study, further syntheses of libraries toward cathepsin D used only scaffold **1**. Computer modeling (see below) predicted that structure **1** (Fig. 1) would provide the most potent inhibitors.

Figure 2

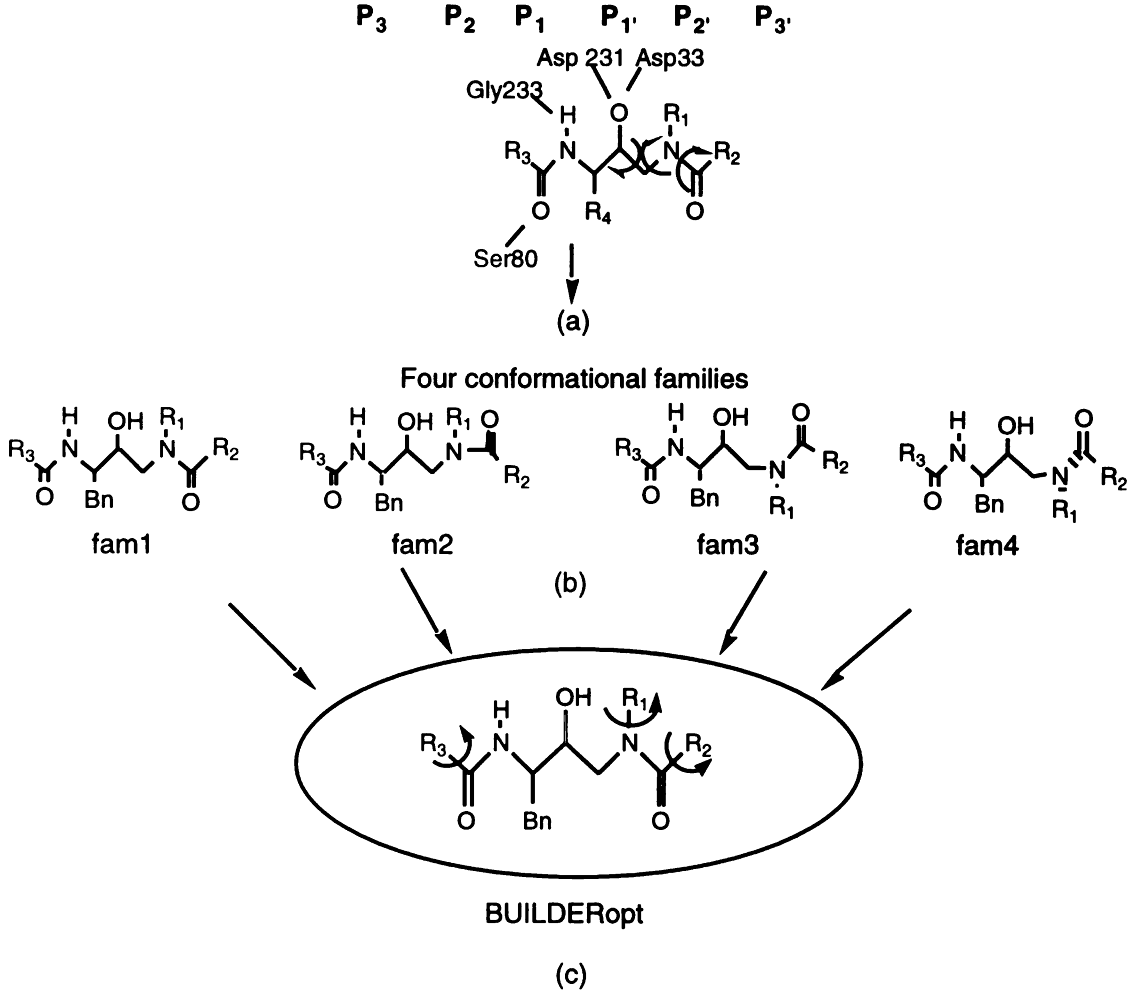


Components employed to prepare the libraries targeting cathepsin D. The same disconnections provide scaffold **2**. Isocyanates and sulphonyl chlorides, which can be used to incorporate R_2 and R_3, provide ureas and sulphonamides, respectively.

The solid-phase synthesis, which we previously reported (23), introduces diversity in three positions: a primary amine for the R_1 substituent and acylating agents for the R_2 and R_3 substituents (Fig. 2). The library synthesis was designed to use commercially available compounds for incorporation of the functionality at R_1, R_2, and R_3. We began our library design by extracting amines, carboxylic acids, sulfonyl chlorides and isocyanates with MW < 275 daltons from the Available Chemical Directory (ACD, version 93.2) from

MDL Information Systems, San Leandro, CA. We eliminated compounds with functionality obviously incompatible with our synthesis. The resulting list included approximately 700 amines and 1900 acylating agents, which would provide access to more than 1 billion compounds. To reduce the number of compounds in a sensible way, we turned to a computational screening process.

Figure 3



Designing the combinatorial library with BUILDERopt: (a) **Modeling the Scaffold.** Coordinates and P_1-P_3 conformations of the pepstatin inhibitor were used as the starting geometry for hydroxyethylamine scaffold. Methyl groups were placed at each of the scaffold's R_1-R_4 positions. (b) **Scaffold Conformation.** A conformational search about the three torsion angles of the scaffold yielded 4 conformational families. A Benzyl sidechain (Bn) was added to each of these families at the R_4 position. (c) **Evaluating library components.** The program BUILDERopt performed a limited conformational search on all possible components at each variable position (R_1 - R_3) on each family, and scored the components by their potential interaction with cathepsin D. The top scoring candidates for each family were merged.

Directed Library Design

We chose a structure-based screening process using a new feature for our BUILDER molecular modeling program (24- 25), called BUILDERopt (26) (see Materi-

als and Methods). To begin the design process the scaffold was modeled in the active site with the assumption that the binding orientation of the scaffold would be similar to pepstatin. (Fig. 3a). A conformational search of the scaffold provided four conformational families of comparable energy that were based on geometric similarity (Fig. 3b). BUILDERopt (26), was used to position each of the R_1, R_2, and R_3 components onto the scaffold and to perform a full conformational search for the torsion angles of the substituent. In order to reduce the combinatoric problem, the R_1, R_2, and R_3 components were examined independently, but a probability-based clash grid was constructed to identify R_1 and R_2 conformations that might overlap. For example, if an R_1 conformation clashed with more than 50% of the R_2 components, that conformation was discarded. The fifty best scoring components for all conformational families were merged for each of the three variable positions. Components with cost above $35/gm were removed. The remaining compounds were hierarchically clustered to maximize the diversity of the top ranking compounds that were selected for library synthesis. For R_1, R_2, and $R_{3,}$ the ten best scoring compounds from unique clusters were selected for each position.

Diverse Library Design

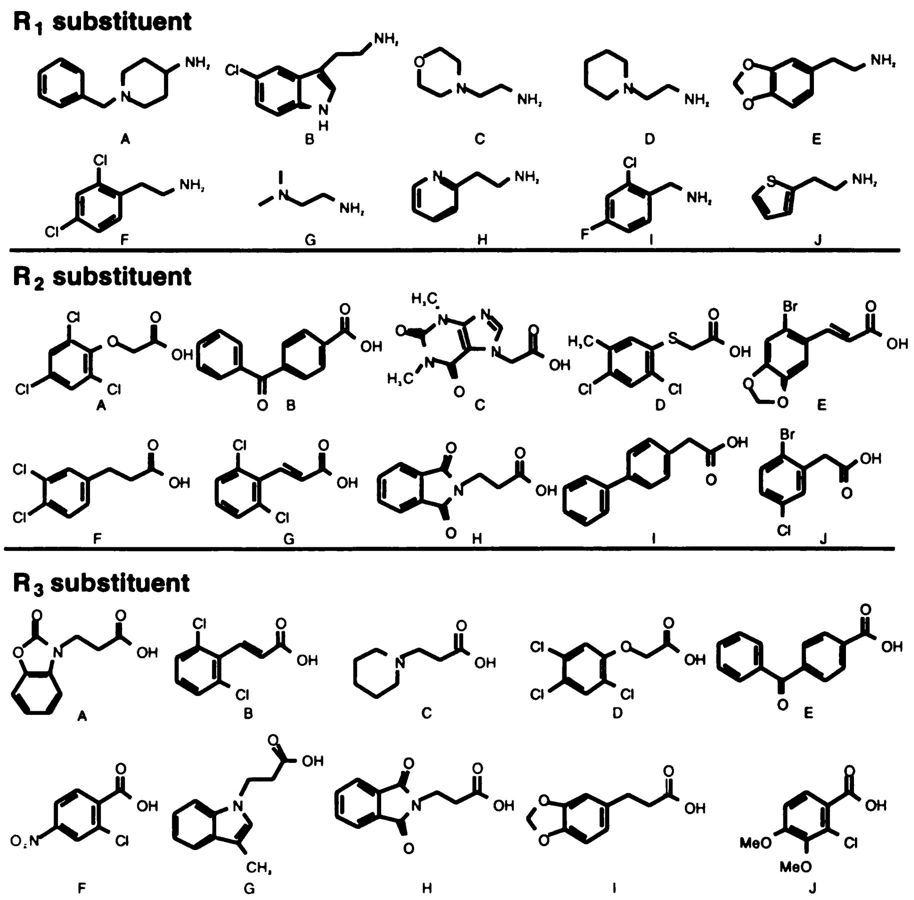
An alternative library based on molecular diversity, which was set at the same size as the directed library, was prepared to provide a "hit" rate when structure-based methods are not employed. The diverse library was designed to maximize the variety of functional groups and structural motifs of the library components. The sidechains for this library were selected by clustering the original list of components based on their similarity to each other. Components were clustered with the Jarvis-Patrick algorithm (27) using the Daylight connectivity measure of similarity (28) and a binary Tanimoto metric (29- 30)

(see Materials and Methods).

Library Synthesis and Screening

The directed and diverse libraries (1000 compounds each) were prepared using diastereomer **1** of the hydroxyethylamine scaffold with the components used in library syntheses shown in figures 4 and 5, respectively. Because the pilot study with *R* and *S* epimers only showed activity at 1 Minhibitor concentration for the *S* epimers, only the *S* epimers of the directed and diverse library were synthesized. All libraries were synthesized in a spatially separate format, and were screened in a high-throughput fluorometric assay for inhibitory activity against cathepsin D (31).

Figure 4



Directed library components are labeled with a letter code. EHA is defined as R_1 = E; R_2 = H; and R_3 = A. Lower case labels are used for the diverse library (Fig. 5).

Figure 5

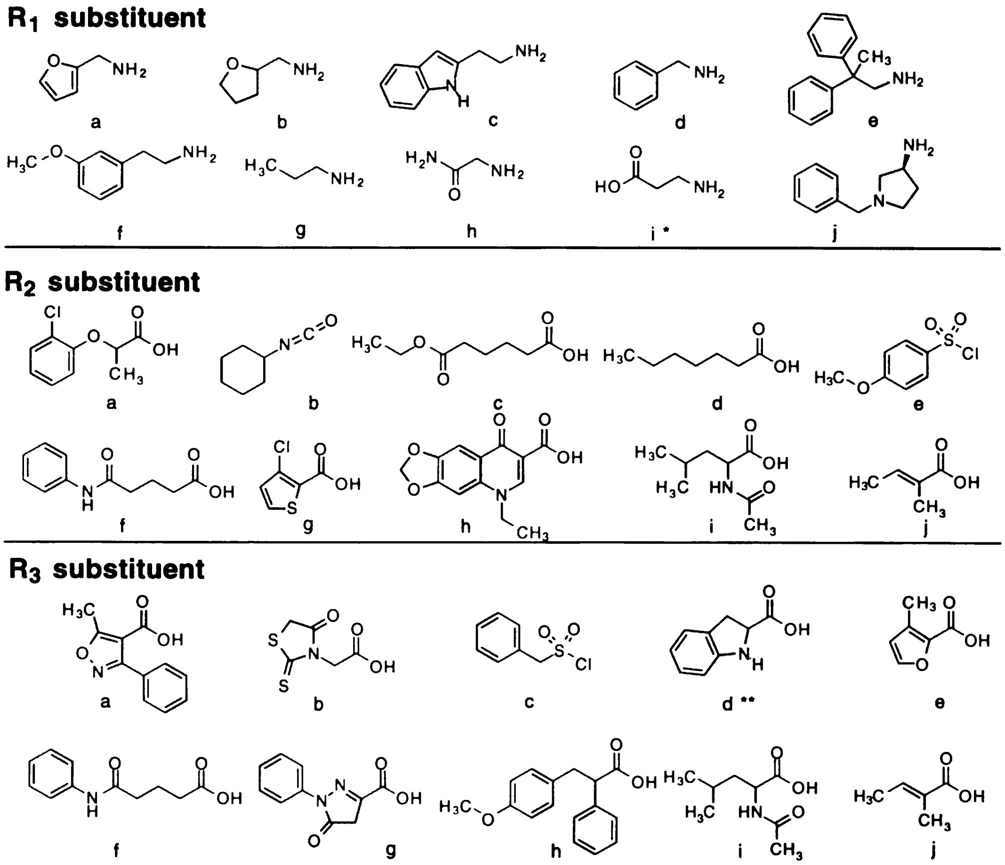


Fig. 5. Diverse library components are labeled by lower case letter code as for the directed library. *The -butyl ester of R_1 = i was used in the coupling reaction. **The Boc protected amine of R_3 = d was used in he coupling reaction.

Diverse library components are labeled by lower case letter code as for the directed library. *The *t*-butyl ester of R_1 = i was used in the coupling reaction. **The Boc protected amine of R_3 = d was used in the coupling reaction. These protecting gropus are removed during TFA:H_2O cleavage.

Assay Results

At approximately 1 Mof crude compound, the directed library yielded 67 compounds that inhibited cathepsin D activity > 50%. Further dilution to 333 nM and 100 nM inhibitor concentrations afforded 23 and 7 compounds, respectively, that inhibited cathepsin D activity > 50% (Table 1). The data for the diverse library are also in Table 1. There are many uncertainties that can influence the results of a high-throughput fluorescence

assay, including the purity of each compound, the concentration of the compounds, and the experimental errors associated with the microtiter fluorescence assay. From repetitive experiments we estimate these errors to be approximately 30%, expressed as enzyme activity.

Table 1: **Number of compounds inhibiting cathepsin D**

[Inhibitor]	Directed Library	Diverse Library[#]
100 nM	7[*]	1[§]
330 nM	23[†]	3[¶]
1 M	67	26
10 M	11/95[‡]	

Compounds which inhibit >50% of cathespin D activity at respective concentrations: [*]EAA, EFA, EHA, EHD, EHI, EHJ, FHA. An additional six compounds provided 40-50% inhibition of cathepsin D. [†]EAA, EFA, EHA, FAA, FFA, FHA, EHB, EFD, EHD, EEF, EHF, FHF, EFH, EHH, FAH, FFH, EFI, EHI, EAJ, EFJ, EGJ, EHJ, FHJ. An additional thirty compounds provided 40-50% inhibition of cathepsin D. [‡]One hundred compounds were selected by random number generation for testing at 10 M.Five compounds were highly fluorescent at these concentrations, so that accurate assay data could not be obtained in these cases. [§]fbb. [¶]fba, fbb, fcb. Four compounds (fca, fdb, fib, hhb) provided 40-50% inhibition of cathepsin D; with the experimental error in the assay, this activity is similar to the activity for the three that are listed. [#]The diverse library was not tested at 10 M.

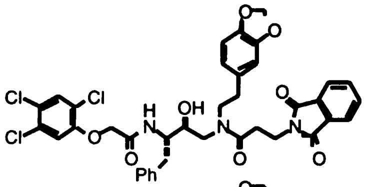
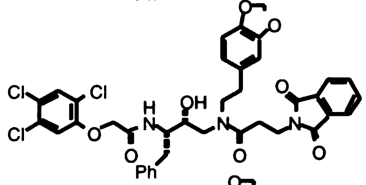
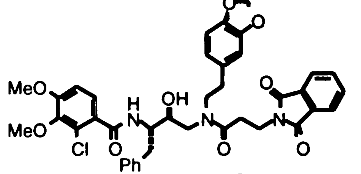
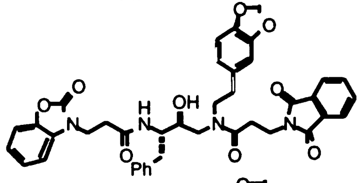
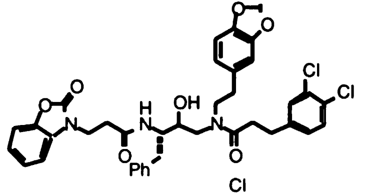
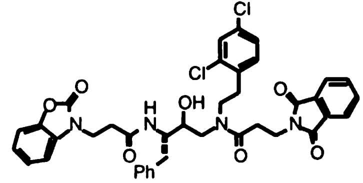
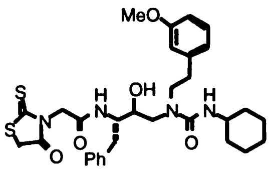
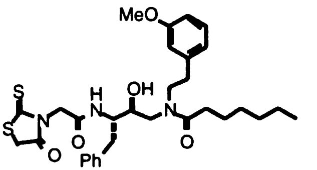
In order to obtain accurate inhibition constants (K_i), several compounds judged to be potent inhibitors based on the library screening were synthesized on a larger scale, purified by chromatography, and characterized by NMR and mass spectrometry. The K_i values were calculated from IC_{50} determinations (Table 2). From the compounds that were fully characterized, we obtained one compound from the directed library with a K_i below 100 nM, whereas the diverse library contained inhibitors that were 3-4 times less potent.

Second Generation Library.

In the design of the directed library we selected against derivatives with a high level of structural similarity by applying a clustering algorithm to the highest scoring components (see Directed Library Design). We re-examined these clusters to explore the important structural elements of the most active compounds from the directed library. In particular, we synthesized and screened a second generation library of thirty-nine compounds from the clusters for the R_1, R_2 and R_3 positions that provided the most active compounds (Fig 6). At 1 M92% of the compounds screened inhibited cathepsin D > 50%, and 18% of the compounds at 100 nM inhibited cathepsin D >50%. Inhibition constants were determined for selected compounds (table 3), providing several potent inhibitors (*K*i = 15 nM) of cathepsin D.

Table 2

Inhibition constants for selected compounds

Inhibitor	Cmp code	K_i (nM)
	**EHD**	**73 ± 9**
	**(R)-EHD**	**>5000**
	**EHJ**	**111 ± 8**
	**EHA**	**131 ± 12**
	**EFA**	**171 ± 25**
	**FHA**	**231 ± 31**
	**fbb**	**356 ± 31**
	**fdb**	**595 ± 66**

Inhibition constants (K_i) were determined for several of the "hits" from the designed and diverse libraries. The K_i values were determined from the IC_{50} values (see Materials and Methods).

Figure 6

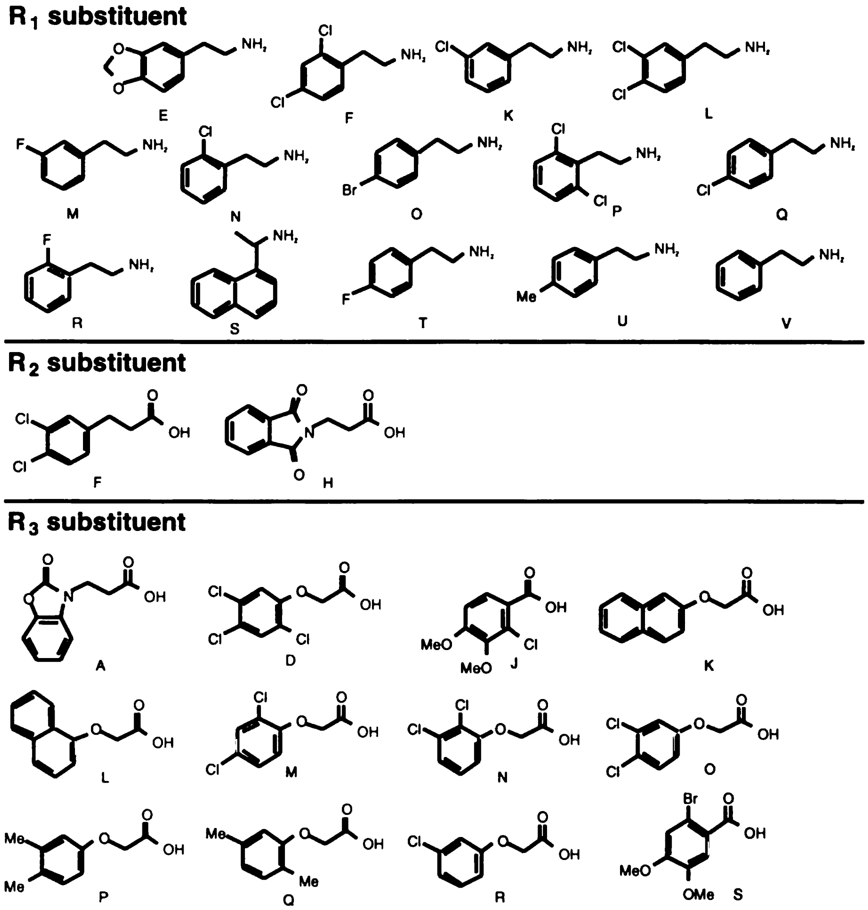
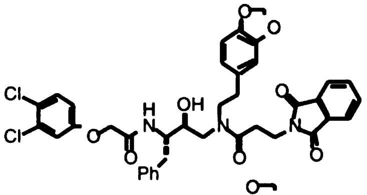
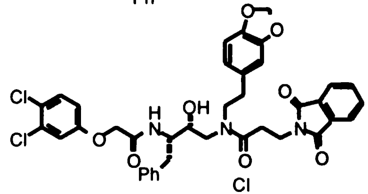
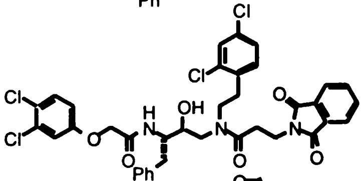
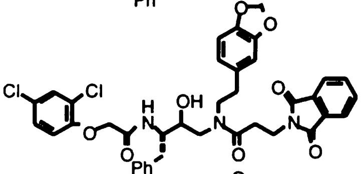
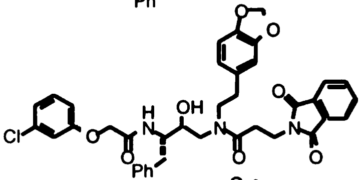
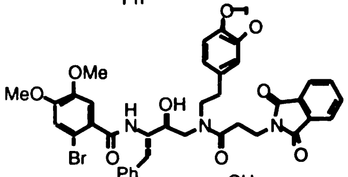
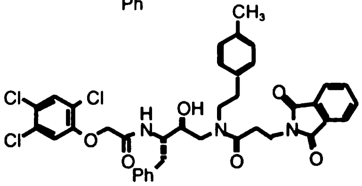


Fig. 6. Components in each of the clusters (see Experimental Design) that contained the most active sidechains, R^1 = E, F; R^2 = F, H; R^3 = A, D, J. Thirty-nine compounds incorporating these sidechains were synthesized on resin as described previously, EFD, EHD, FFD, FHD, KFD, KHD, LFD, LHD, MFD, MHD, NFD, NHD, OFD, OHD, PFD, PHD, QFD, QHD, RFD, RHD, SFD, SHD, TFD, THD, UFD, UHD, VFD, VHD, EHA, EHJ, EHK, EHL, EHM, EHN, EHO, EHP, EHQ, EHR, EHS. The compounds were assayed at 333 nM, 100 nM and 33 nM in high-through put screening. The most active compounds were synthesized on large scale and the K_i values were determined (Table 3).

Table 3
Second Generation Library Inhibition Constants (*K_i*)

Inhibitor	Cmp.code	IC_{50}(nM)	K_i(nM)
	**EHO**	**19 ± 2**	**15**
	**(*R*)-EHO**	**>5000**	
	**FHO**	**18 ± 2**	**14**
	**EHM**	**14 ± 2**	**9**
	**EHR**	**20 ± 15**	**15**
	**EHS**	**64 ± 6**	**59**
	**UHD**	**229 ± 44**	**224**

Inhibition constants (K_i) were determined for selected compounds from the second generation library. The K_i values were determined from the IC_{50} values shown (see Materials and Methods).

Discussion

Novel low nanomolar inhibitors of cathepsin D were identified rapidly using combinatorial chemistry coupled with two different computational strategies. The diverse and

directed libraries together yielded over 90 compounds active at 1 Mand 26 active in the submicromolar range. The "hit rate" for activity at 1 Mis 6-7% for the directed library and 2-3% for the diverse library. When screening was performed at concentrations below 1 M, there were seven times more "hits" in the directed library than the diverse library. The most potent inhibitors from the directed library are 3-4 fold better than those in the diverse library. It is clear from the results that the number and quality of the active compounds can be increased by using relevant information about the target.

A strength of the structure-based procedure is that it leads directly to testable geometrical hypotheses. In this study there are three hypotheses: 1) *S* epimers are predicted to bind better than the *R* epimers; 2) there are two energetically reasonable scaffold conformations (family 1+2, family 3+4), which place R groups into different pockets; 3) all the inhibitors are assumed to bind in approximately the same orientation as pepstatin. The first hypothesis was directly tested in pilot experiments where no inhibitors based upon the *R* epimer had activity at 1 M. In addition, the *R* epimer of one of the most potent compounds had a K_i no better than 5 Mwhile the K_i of the *S* epimer was 15 nM (Table 3). This conclusion and the inhibitor orientations in the cathepsin D complex will be examined crystallographically.

The computational approach outlined in this paper is most applicable when the scaffold orientation can be restricted using information from structures of complexes. We are developing methods (32) that will work even if the scaffold orientation is unknown or uncertain. One of these methods docks scaffolds and sidechains simultaneously. Another

docks sidechains separately and then links them. With these methods we overcome the combinatorial explosion normally associated with generating all possible combinations in advance. Ultimately, the complexities of combinatorial chemistry may require different computational strategies for the design of the full range of oligomeric and small molecule libraries.

The work presented here is seen as the first stage of a process in which active compounds are identified and then, in later stages, the activity is optimized. The optimization criteria can include improved potency, selectivity, pharmacokinetic properties, or reduced toxicity. Each of these issues appears amenable to library design. For example, compounds with five-six fold improved potencies were rapidly identified by synthesizing and screening a small second generation library that explored variants of the most active compounds.

The success of the directed library toward finding potent inhibitors demonstrates the power of coupling combinatorial libraries with structure-based design. Combinatorial libraries allow a larger area of molecular space to be explored with the functionality selected by the structure-based design, removing the need to identify in advance a single "best" target. Similarly, computational methods allow rapid examination of extremely large virtual regimes (> 10^{10} compounds) and focus the chemical efforts into productive regimes.

Conclusion

The identification of potent, small molecule ligands to receptors and enzymes is

one of the major goals of chemical and biological research. Two powerful new tools in these efforts are structure-based design and combinatorial chemistry. In the present work we have integrated these approaches to develop a general method for the rapid identification of potent enzymes inhibitors. We have demonstrated this method by identifying potent, nonpeptide inhibitors (K_i = 9-15 nM) of cathepsin D, a prototypical aspartyl protease that has been implicated in a number of therapeutically relevant processes, but for which potent nonpeptide inhibitors have not previously been reported.(1)

The significance of this study is three fold. First, this general method should be directly applicable to the rapid identification of potent inhibitors of the other members of the aspartyl protease class. Moreover, with careful selection of the appropriate scaffold, we anticipate that the general approaches described here will also be successful for many other enzyme classes. Finally, the success of this work clearly demonstrates the power of coupling the complementary approaches of combinatorial chemistry and structure-based design.

Acknowledgments

Support for this work was provided by the National Institutes of Health to IDK and by the National Science Foundation, the A.P. Sloan Foundation and the Burroughs Wellcome Fund to JAE. We gratefully acknowledge predoctoral fellowship support from The American Chemical Society Division of Medicinal Chemistry and Parke-Davis Research to EKK, Pharmaceutical Training Grant, GM07075, to DCR and AGS and Pharmacia-Upjohn to GL. We thank Dr. Jack Kirsch for advice on kinetic analysis and use of a

fluorescence spectrophotometer, Dr. Barbara Chapman for advice and assistance in the fluorescence assay, Christina E. Lee and Dr. Connie Oshiro for helpful comments and discussions. Structure searches were performed using the ACD provided by MDL Information Systems, Inc., 14600 Catalina St., San Leandro, CA 94577.

References

1. Recently researchers at Eli Lilly and Company reported several benzophenones that inhibit cathepsin D. The most potent reported compound had an IC_{50} value of 210 nM. Whitesitt, C.A., et al. & Panetta, J.A. (1996). Synthesis and Structure-activity Relationships of Benzophenone as Inhibitors of Cathepsin D. *Bioorg. & Med. Chem. Lett.* 6, 2157-2162.

2. Kuntz, I.D. (1992). Structure-Based Strategies for Drug Design and Discovery. *Science* **257**, 1078-1082.

3. Kuntz, I.D., Meng, E.C., & Shoichet, B.K. (1994). Structure-Based Molecular Design. *Accts. Chem. Res.* **27**, 117-123.

4. Thompson, L.A., & Ellman, J.A. (1996). Synthesis And Applications Of Small Molecule Libraries. *Chem. Rev.* **96**, 555-600.

5. Gordon, E.M., Barrett, R.W., Dower, W.J., Fodor, S.P.A., & Gallop, M.A. (1994). Applications Of Combinatorial Technologies To Drug Discovery. 2. Combinatorial Organic Synthesis, Library Screening Strategies, And Future Directions. *J. Med. Chem.* **37**, 1385-1401.

6. Cohen, N.C., Blaney, J.M., Humblet, C., Gund, P., & Barry, D.C. (1990). Molecular Modeling Software and Methods for Medicinal Chemistry. *J. Med. Chem.* **33**, 883-

894.

7. Gallop, M.A., Barrett, R.W., Dower, W.J., Fodor, S.P.A., & Gordon, E.M. (1994). Applications Of Combinatorial Technologies To Drug Discovery.1. Background And Peptide Combinatorial Libraries. *J. Med. Chem.* **37**, 1233-1251.

8. Terrett, N.K., et al. & Steele, J. (1995). The Combinatorial Synthesis of a 30,752-Compound Library: Discovery of SAR Around the Endothelin Antagonist, FR-139,317. *Bioorg. Med. Chem. Lett.* 5, 917-922.

9. Simon, R.J., et al. & Bartlett, P. A. (1992). Peptoids: A Modular Approach To Drug Discovery. *Proc. Natl. Acad. Sci. U. S. A.* **89**, 9367-9371.

10. Takahashki, K. (1995) Aspartic Proteinases Structure, Function, Biology, and Biomedical Implications, Plenum Press, New York.

11. Adams, J., & Stein, R. (1996). Novel Inhibitors of the Proteasome and Their Therapeutic Use in Inflammation. *Ann. Rep. Med. Chem.* **31**, 279-288.

12. Edmunds, J.J., Rapundalo, S.T., & Siddiqui, M.A. (1996). Thrombin and Factor Xa Inhibition. *Ann. Rep. Med. Chem.* **31**, 51-60.

13. Miller, D.K. (1996). Regulation of Apoptosis by Members of the ICE Family and the Bcl-2 Family. *Ann. Rep. Med. Chem.* **31**, 249-268.

14. Lam, P.Y.S., et al. & Erickson-Viitanen, S. (1994). Rational Design of Potent, Bioavailable, Nonpeptide Cyclic Ureas as HIV Protease Inhibitors. *Science* **263**, 380-384.

15. DeCamp, D., Ogden, R., Kuntz, I., & Craik, C. (1996). Site-Directed Drug Design. In *Site-Directed Drug Design*, (J. L. Cleland and C. S. Craik, eds.), pp. 467-505, Wiley-Liss, New York.

16. Westley, B.R., & May, F.E.B. (1996). Cathepsin D And Breast Cancer. *Eur. J. Cancer* **32**, 15-24.

17. Ladror, U.S., Snyder, S.W., Wang, G.T., Holzman, T.F., & Krafft, G.A. (1994). Cleavage At The Amino And Carboxyl Termini Of Alzheimers Amyloid-Beta By Cathepsin D. *J. Biol. Chem.* **269**, 18422-18428.

18. Cataldo, A.M., Hamilton, D.J., Barnett, J.L., Paskevich, P.A., & Nixon, R.A. (1996). Properties of the Endosomal-Lysosomal System in the Human Central Nervous System: Disturbances Mark Most Neurons in Populations at Risk to Degenerate in Alzheimer's Disease. *J. Neurosci.* **16**, 186-199.

19. Wiley, R.A., & Rich, D.H. (1993). Peptidomimetics Derived from Natural Products. *Med. Res. Rev.* **13**, 327-384.

20. Baldwin, E.T., et al. & Erickson, J.W. (1993). Crystal structures of native and inhibited forms of human cathepsin D: Implications for lysosomal targeting and drug design. *Proc. Natl. Acad. Sci. U. S. A.* **90**, 6796-6800.

21. Jupp, R.A., et al. & J. Kay (1990). The selectivity of statine-based inhibitors against various human aspartic proteinases. *Biochem. J.* **265**, 871-878.

22. Agarwal, N.S., & Rich, D.H. (1986). Inhibition of Cathepsin D by Substrate Analogues Containing Statine and by Analogues of Pepstatin. *J. Med. Chem.* **29**, 2519-2524.

23. Kick, E.K., & Ellman, J.A. (1995). Expedient Method For The Solid-Phase Synthesis Of Aspartic Acid Protease Inhibitors Directed Toward The Generation Of Libraries. *J. Med. Chem.* **38**, 1427-1430.

24. Lewis, R.A., et al. & Kuntz, I.D. (1992). Automated site-directed drug design using

molecular lattices. *J. Mol. Graphics* **10**, 66-78.

25. Roe, D.C., & Kuntz, I.D. (1995). BUILDER v2: Improving the Chemistry of a De Novo Design Strategy. *JCAMD* 9, 269-282.

26. Roe, D.C. (1995) Development and Application of Tools for Structure-based Drug Design, University of California, San Francisco.

27. Jarvis, R.A., & Patrick, E.A. (1973). Clustering Using a Similarity Measure Based on Shared Near Neighbors. *IEEE Comput* **C22**, 1025-1034.

28. Daylight Clustering Toolkit, v., Daylight Chemical Information Systems, Inc., Santa Fe, NM Daylight Clustering Toolkit, version 4.41, Daylight Chemical Information System, Inc., Santa Fe, CA.

29. Willett, P., Winterman, V., & Bawden, D. (1986). Implementation of non-hierarchical cluster-analysis methods in chemical information systems: selection of compounds for biological testing and clustering of substructure search output. *J. Chem. Inf. Comput. Sci.* **26**, 109-118.

30. Willett, P. (1987). *Similarity and Clustering in Chemical Information Systems*John Wiley & Sons, New York, NY.

31. Krafft, G.A., & Wang, G.T. (1994). Synthetic Approaches to Continuous Assays of Retroviral Proteases. *Methods Enzymol.* **241**, 70-86.

32. Y. Sun, T. J. A. Ewing, I. D. Kuntz, in preparation.

33. Weiner, S.J., et al. & Weiner, P.A. (1984). A New Force Field for Molecular Mechanical Simulation of Nucleic Acids and Proteins. *J. Am. Chem. Soc.* **106**, 765-784.

34. Gasteiger, J., & Marsili, M. (1980). Iterative partial equalization of orbital electronegativity: rapid access to atomic charges. *Tetrahedron Lett* **36**, 3219.

35. Gasteiger, J., & Marsili, M. (1981). *Organ. Magn. Reson.* **15**, 353.

36. Gschwend, D.A., & Kuntz, I.D. (1996). Orientational Sampling and Rigid-Body Minimzation in Molecular Docking, Revisited: On-the-fly Optimization and Degeneracy Removal. *J. Compt-Aided Drug Design* **10,** 123-132.

37. Meng, E.C., Shoichet, B.K., & Kuntz, I.D. (1992). Automated Docking with Grid-Based Energy Evaluation. *Journal of Computational Chemistry* **13**, 505-524.

38. Romesburg, H.C. (1984). *Cluster Analysis For Researchers*Lifetime Learning Publications, Belmont, CA.

39. Cha, S., Agarwal, R.P., & R. E. Parks, J. (1975). Tight-binding Inhibitors-II: Nonsteadyt State Nture of Inhibition of Mild Xanthine Oxidase by Allopurinol and Alloxanthine and of Human Erythrocytic Adenosine Deaminase by Coformycin. *Biochem. Pharmacol.* **24**, 2187-2197.

Epilogue to Chapter 7

Retrospectively, this first collaboration with the Ellman group produced two important results. First, it reasserted a role for structure-based methods in the design of combinatorial libraries. Second, the data obtained in this study provided an excellent tool for retrospective analysis of subsequent methods. The thirteen Ki values reported here, the crystal structure of EHO bound to Cathepsin D (Erickson, personal communication), and the 23 compounds which inhibit Cathepsin D with IC_{50} of < 330nM provide an excellent measure of the reliability of both scoring functions and structure-based design methods. When considering these test cases, one must bear in mind that all of the side-chains in this library are quite rigid compared to many of the side-chains in the potential virtual library. Despite these minor drawbacks, this data set is an excellent measure of basic ability of combinatorial algorithms to enrich hits in a database.

This paper was an attempt to compare structure-based design methods to diversity design methods. However, Yvonne Martin has recently reported that when screening individual compounds from a database random selections of molecules is as good (identifies the same number of inhibitors) as selection of a diverse set of molecules (Yvonne Martin, ACS Meeting, March 1999, Anaheim, CA.). If this is true, then the diverse selection of molecules used in this paper might be no better than a random selection.

Prologue to Chapter 8

Chapter 8 is a combination of some of the most enjoyable elements from Chapters 6 and 7. In this chapter, we extend the combinatorial design and synthesis approach taken in chapter 7 to another parasite target (chapter 6). In this instance, I designed a series of (hydroxyethyl)amine combinatorial libraries (both diverse and structure-based), while Tas Haque (of the Ellman group) synthesized, characterized, and assayed the libraries. Here the target was plasmepsin II, an essential aspartyl protease of *Plasmodium falciparum*. While Tas and I benefited from already having UC_Select (chapter 1) to help with reagent selection, this project suffered from being on the *bleeding edge* of Dock 4.0 development. I am particularly grateful to Todd Ewing for his willingness to carry out minute-by-minute bug-fixes and add several features to Dock 4.0 as this project worked its way through the new code. Finally, I want to mention the important work of Chris Lipinski. He established a set of molecular criteria necessary for small molecules to crossing biological membranes (*vida infra*). With the help of these molecular criteria, early in the design process we identified compounds with *in vivo* as well as *in vitro* potency. As in our cathepsin D project, it was a pleasure working with the brilliant, careful, and diligent scientists from Jon Ellman's group at University of California, Berkeley.

Chapter 8: Single Digit Nanomolar, Low Molecular Weight Non-Peptide Inhibitors of Malarial Aspartyl Protease Plasmepsin II.

by

Tasir S. Haque, A. Geoffrey Skillman, Christina E. Lee, Hiromu Habashita, Ilya Y. Gluzman, Todd J. A. Ewing, Daniel E. Goldberg, Irwin D. Kuntz, and Jonathan A. Ellman.

Reproduced with permission from
The Journal of Medicinal Chemistry, April 1999, 42(8):1428-1440.

Abstract

A number of single digit nanomolar, low molecular weight plasmepsin II aspartyl protease inhibitors have been identified using combinatorial chemistry and structure-based design. By identifying multiple potent, small molecule inhibitors, it was possible to select for compounds with desirable characteristics including enzyme specificity and minimal binding to serum proteins. The best inhibitors identified have *K*is of 2-10 nM, molecular weights between 594 and 650 Da, between 3 to 15-fold selectivity towards plasmepsin II over cathepsin D, the most closely related human protease, good calculated logP values (2.86-4.56), and no apparent binding to human serum albumin at 1 mg/mL in an *in vitro* assay. These compounds represent the most potent non-peptide plasmepsin II inhibitors reported to date.

Introduction

Malaria is a parasitic disease that afflicts 300-500 million people worldwide, killing 1-2 million annually. 1 It is estimated that up to 40% of the world's population lives in regions where malaria is endemic. *Plasmodium falciparum* is the most dangerous form of the four malaria parasites that infect humans, and is responsible for more than 95% of malaria-related deaths. 2 Increasing resistance of *Plasmodium falciparum* to existing therapies has heightened concerns about malaria in the international health community. However, there has been little economic incentive for the development of new drug-based antimalarial therapies. While quinine has been used for hundreds of years to treat malaria, its use as an antimalarial agent is hindered by parasite resistance. Chloroquine,

an inexpensive quinine-derivative first identified as an antimalarial agent in the 1930s, also faces widespread resistance throughout the world. Other commonly used antimalarial drugs such as mefloquine and Fansidar can provide effective treatment in cases of chloroquine resistance, but parasites resistant to these alternative therapies have also been reported.[3] Indeed, some *P. falciparum* strains have been identified that are resistant to all known antimalarial drugs.[4] The alarming spread of parasite resistance clearly indicates that new strategies are necessary for finding safe and effective means of treating malaria. The plasmepsin aspartyl proteases of *P. falciparum*[5] are potentially new chemotherapeutic targets. Therefore, we used combinatorial chemistry and structure-based design methods to rapidly identify potent and selective, low molecular weight aspartyl protease inhibitors using several iterative focused libraries.

The *Plasmodium* parasite invades human erythrocytes and consumes up to 75% of the hemoglobin present.[6] Three enzymes have been identified that digest hemoglobin in an acidic parasite food vacuole; a cysteine protease, falcipain, and two aspartyl proteases, plasmepsins I and II (Plm I and II).[5] The two plasmepsins have a high degree of sequence homology to one another (73% identical), and are closely related in structure to human cathepsin D (Cat D). However, neither of the plasmepsins is targeted by any of the currently available antimalarial therapies. Of the two aspartyl proteases present in the malarial acidic food vacuole, Plm II was chosen as a primary target because the crystal structure was available,[7] and because sufficient quantities of expressed enzyme could be obtained for high through-put screening of compound libraries. At the initiation of the project, Plm I was only available via purification of parasite extract, whereas Plm II could be expressed and obtained in larger quantities.[8] The high degree of sequence homology of the two plasmepsins suggested that by targeting Plm II, both enzymes could ultimately

be inhibited by the same compounds.

Our approach to obtaining potent Plm II inhibitors began by comparing the crystal structures of Plm II and Cat D both complexed to the peptide-based natural product pepstatin. Plm II and Cat D have a significant sequence homology (35%), and have an even higher sequence homology in the active site region. We had previously reported the successful identification of potent Cat D inhibitors by the design, synthesis, and screening of a library of 1039 mechanism-based inhibitors incorporating the hydroxyethylamine isostere (Figure 1). 9 Due to the close similarity of Plm II and Cat D, we screened this library against Plm II to identify lead compounds.

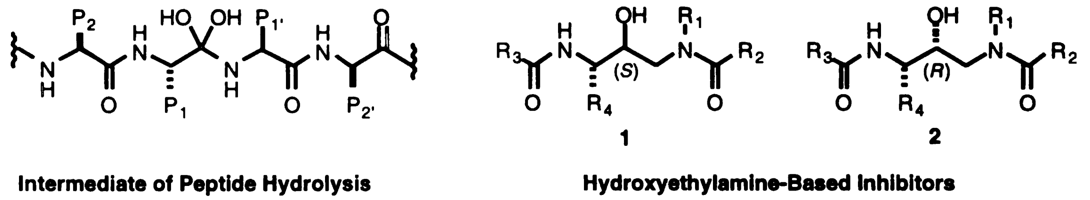

Intermediate of Peptide Hydrolysis
Hydroxyethylamine-Based Inhibitors
1
2

Figure 1. Tetrahedral intermediate of hydrolysis and hydroxyethylamine isostere.

We next used six iterative libraries to optimize the Plm II inhibitors. First, each of the three sites of variation (RA, RB, RC in Figure 1) was examined individually. By synthesizing these compounds in parallel, we were able to examine relatively large numbers of side chains at each position. Two methods were used to select subsets of side chains from the thousands of commercially available amines and acylating agents. One set of side chains was selected by modeling side chains in the Plm II active site using the program DOCK 10-12 and the second set of side chains was selected by clustering to maximize the diverse display of functionality. We have previously shown that modeling can be more efficient at generating potent molecules; 9 however, by using both modeling and

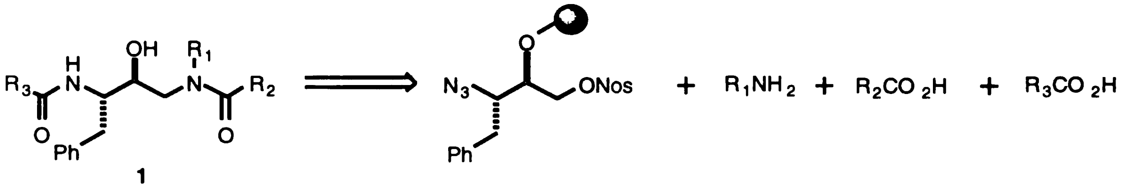
diversity clustering, we hoped to find compounds that would be anticipated to bind in the active site (modeled compounds), while simultaneously investigating whether we could identify side chains with unanticipated binding interactions (diverse compounds). The final two library iterations involved the simultaneous variation of two or more sites around the hydroxyethylamine scaffold to search for cooperative effects.

Results and Discussion

Chemistry

The hydroxyethylamine isostere has served as a particularly effective mechanism-based inhibitor of aspartyl proteases, where the tetrahedral intermediate of the peptide bond cleavage is approximated by the stable hydroxyethylamine compound (Figure 1). The isostere is also amenable to the introduction of a wide variety of side chains on both sides of the secondary alcohol whose position corresponds to the scissile bond of the peptide substrate.

We had previously developed two solid-phase synthesis sequences to prepare libraries of hydroxyethylamine-based inhibitors. In the first approach (Figure 2: Method 1), diverse functionality is introduced at three sites, R_A, R_B, and R_C, using amines and acylating agents. (13) In the second approach, multiple side chains can also be introduced at the P_1 position using Grignard reagents, thereby allowing diverse functionality to be introduced at all possible sites of the inhibitor structure. (14) Both synthesis sequences were used to prepare libraries targeting Plm II.



Components employed to prepare the libraries targeting cathepsin D. The same disconnections provide scaffold **2**. Isocyanates and sulphonyl chlorides, which can be used to incorporate R_2 and R_3, provide ureas and sulphonamides, respectively.

Figure 2. Synthesis approaches toward aspartyl protease inhibitors. (13,14)

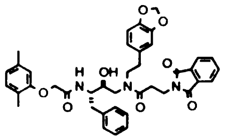
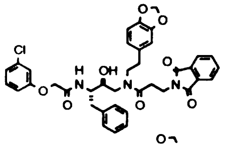
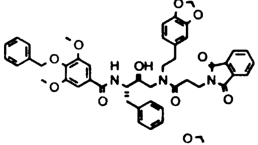
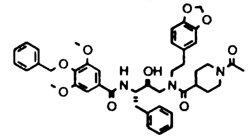
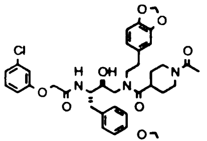
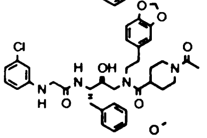
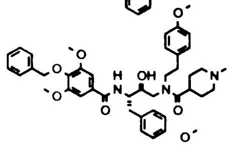
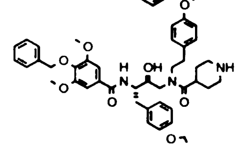
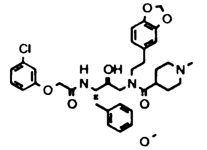
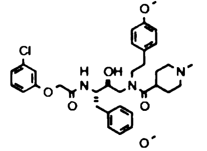
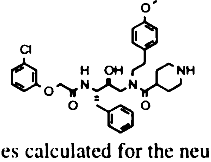
Lead Identification

A library of 1000 compounds (10 x 10 x 10 side chains at the three sites of variation) targeting Cat D and a second 39 compound library optimized toward Cat D were screened against Plasmepsin II using a fluorogenic peptide-based assay with an inhibitor concentration of 1 M. (15) Of the 1039 compounds screened, 13 compounds showed greater than 50% inhibitory activity against Plm II. Compounds **1** and **2**, the most potent of the 13 crude compounds, were prepared on larger scale, purified, and evaluated as representative lead compounds (Table 1). Both of these compounds were validated as submicromolar inhibitors of Plm II, as well as being potent Cat D inhibitors (*K*i = 15 nM, 220 nM for **2** against Cat D and Plm II, respectively).

Iterative Optimization of Individual Sites

The two compounds described above were used to begin optimization against Plm II. Initially, each of three possible sites of variation (RA, RB, and RC) was individually optimized. This iterative approach allowed for the exploration of relatively large numbers of side chains at each site without having to resort to the extremely large numbers of compounds that would be necessary if all sites were examined simultaneously. Subsequent

Table 1.

Entry No.		mol. wt.	ClogP*	OH+NH	N+O	"Rule of 5" fulfillments	K_i (nM)	
							Cat D	Plm II
1		692	5.86	2	11	1	-	300 ± 22
2		698	5.71	2	11	1	15 ± 2**	220 ± 6
3		800	5.50	2	13	1	-	100 ± 3
4		752	2.78	2	12	2	4.3 ± 0.5	4.8 ± 0.6
5		650	2.99	2	10	3	1.9 ± 0.6	4.0 ± 0.4
6		649	2.86	3	10	3	1.3 ± 0.2	3.0 ± 0.3
7		710	4.35	2	10	3	58 ± 10	4.1 ± 0.2
8		696	3.50	3	10	3	71 ± 16	9.0 ± 0.2
9		622	4.00	2	9	3	5.8 ± 0.6	2.0 ± 0.1
10		608	4.56	2	8	3	9.8 ± 1.6	2.0 ± 0.1
11		594	3.71	3	8	3	63 ± 12	4.3 ± 0.1

* ClogP values calculated for the neutral compounds. ** The K_i for compound **2** was determined previously.[9]

libraries probed for cooperativity between two or more sites around the hydroxyethylamine core.

Library R_A. The first site examined was the R_A side chain. The two other variable sites were fixed based on the structures of the lead inhibitors **1** and **2** (R_B = 3,4-methylenedioxyphenethylamine; R_C = 3-phthalimidopropionic acid). A list of commercially available acylating agents was generated by screening the Available Chemical Directory (16) and selecting for characteristics such as chemical compatibility, molecular weight, and cost. From this list, two sets of side chains were established for R_A (Figures 3 and 4). The diverse set was generated by clustering the acylating agents to explore a broad range of functionality at the site (side chains A1-A44, Figure 3). More specifically, the selection technique involved a complete linkage hierarchical agglomerative clustering of side chains based on the atomic content and connectivity of each molecule. The modeled set of side chains was designed to fit specifically into the enzyme active site. DOCK (10-12) was used to model each acylating agent in the appropriate part of the crystallographic active site of Plm II (side chains A45-A74, Figure 4). (7) Nearly all of the 50 highest-scoring modeled side chains had molecular weights between 250 and 300 Da. Since low molecular weight (MW) and potent inhibitors were desired, we selected the 18 highest-scoring side chains with a MW between 250 and 300 Da and the 12 highest-scoring side chains with a MW < 250 Da.

Upon screening the R_A library, compound **3**, one of the modeled compounds, had significantly improved inhibitory activity compared to the lead compounds (Table 1). At the R_A site, this inhibitor incorporated side chain A52 (4-benzyloxy-3,5-dimethoxybenzoic acid). A K_i of 100 nM was determined for analytically pure inhibitor **3** against Plm II, a 2-3-fold improvement over the initially identified lead compounds (Figure 5 and 6). A 12-member library was prepared examining variants of sidechain A52 in an attempt to determine the nature of

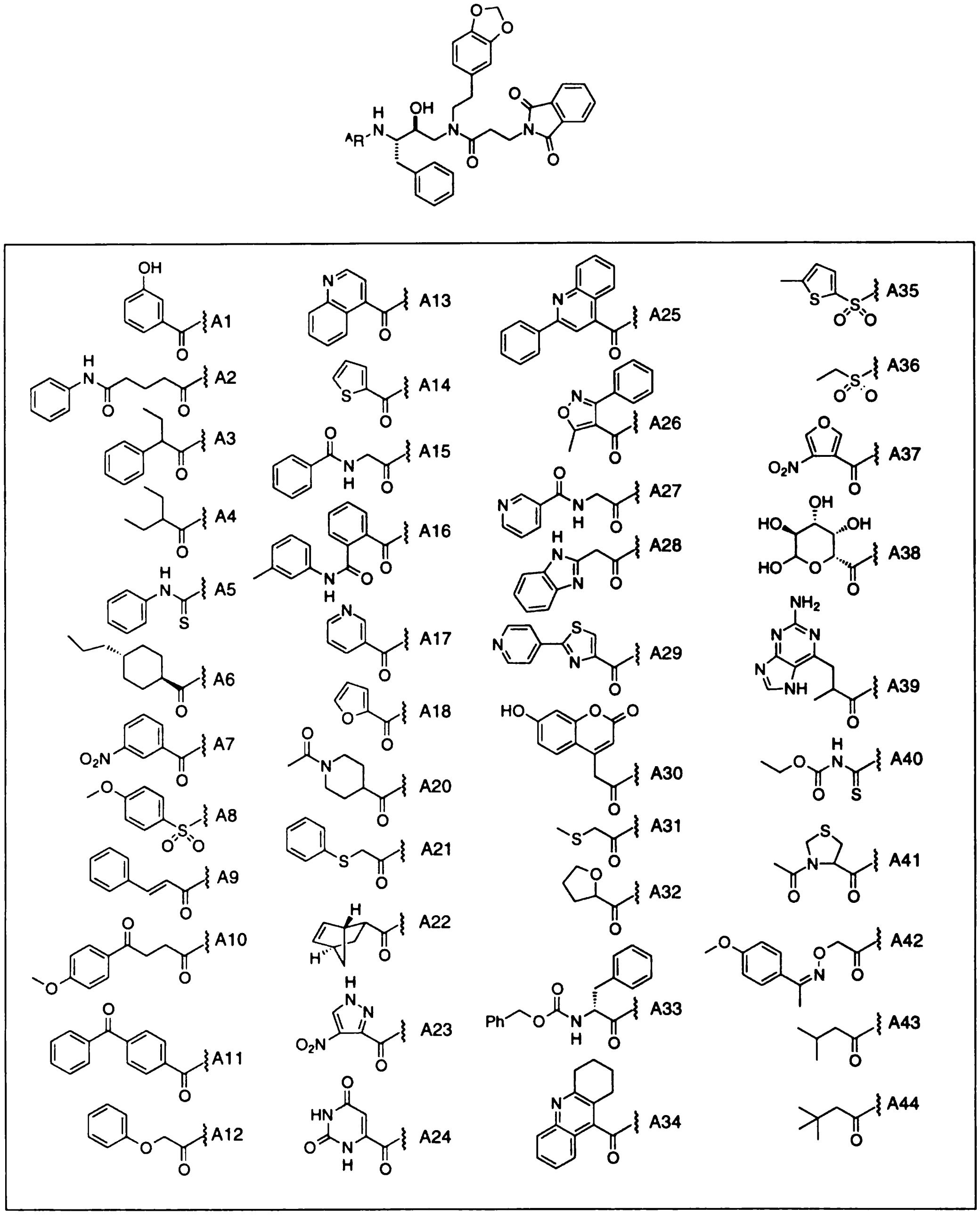


Figure 3. Diverse side chains for Library R^A.

Figure 3. Diverse side chains for library R^A.

the increased affinity of **3** to the Plm II active site (Figure 7). The IC_{50} values of the entire 12-member library were at least 10-fold higher than inhibitor **3** towards Plm II. For example, the compound containing the R_A = A52e (4-benzyloxy-3-methoxybenzoic acid) has a

K_i of 3.7 M against Plm II, demonstrating that removal of just one of the methoxy groups

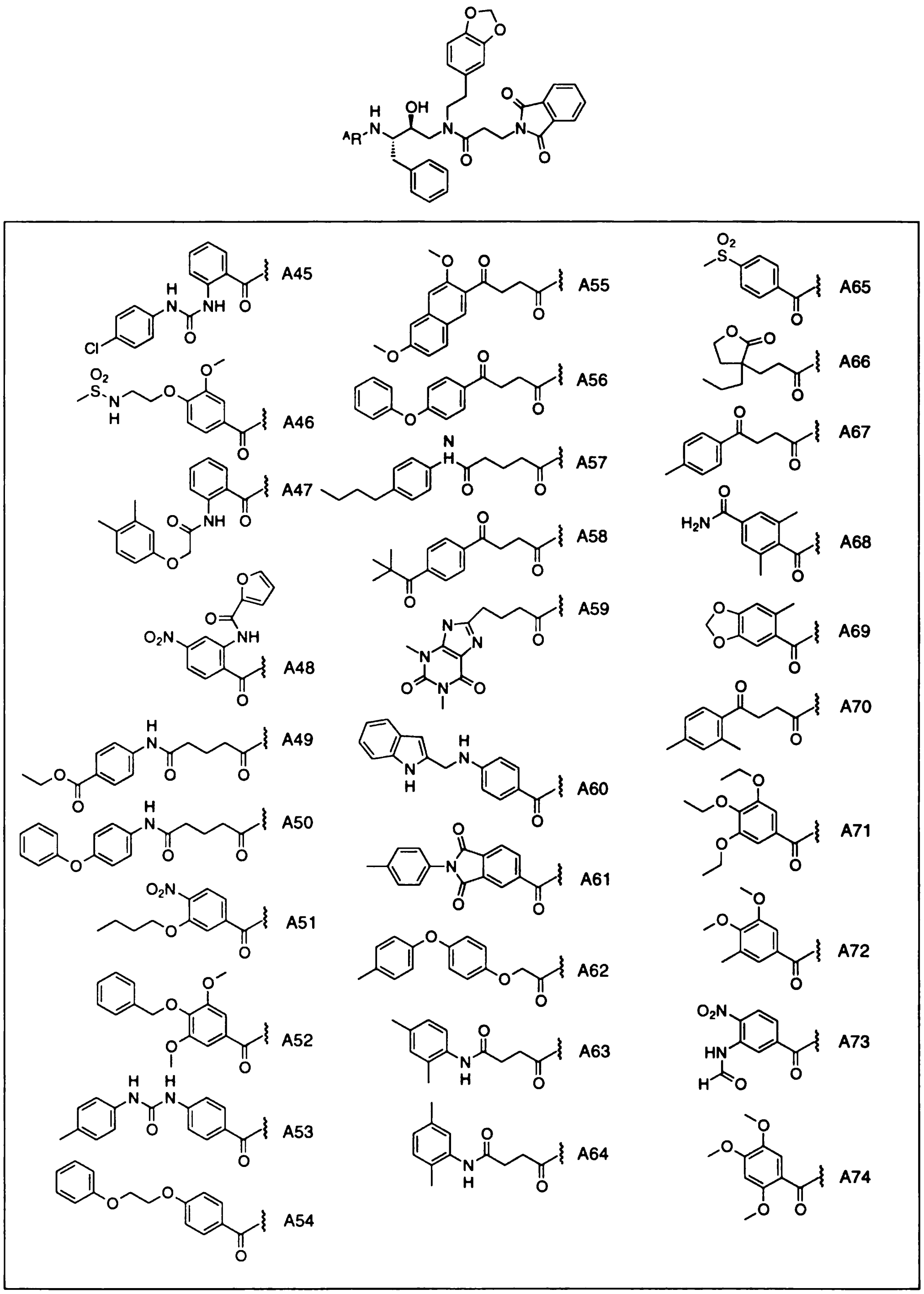


Figure 4. Modeled side chains for library R^A.

from the RA side chain results in significantly decreased affinity to the Plm II active site. This was consistent with the modeled structure in which both methoxy groups make sig-

nificant interactions with the protein.

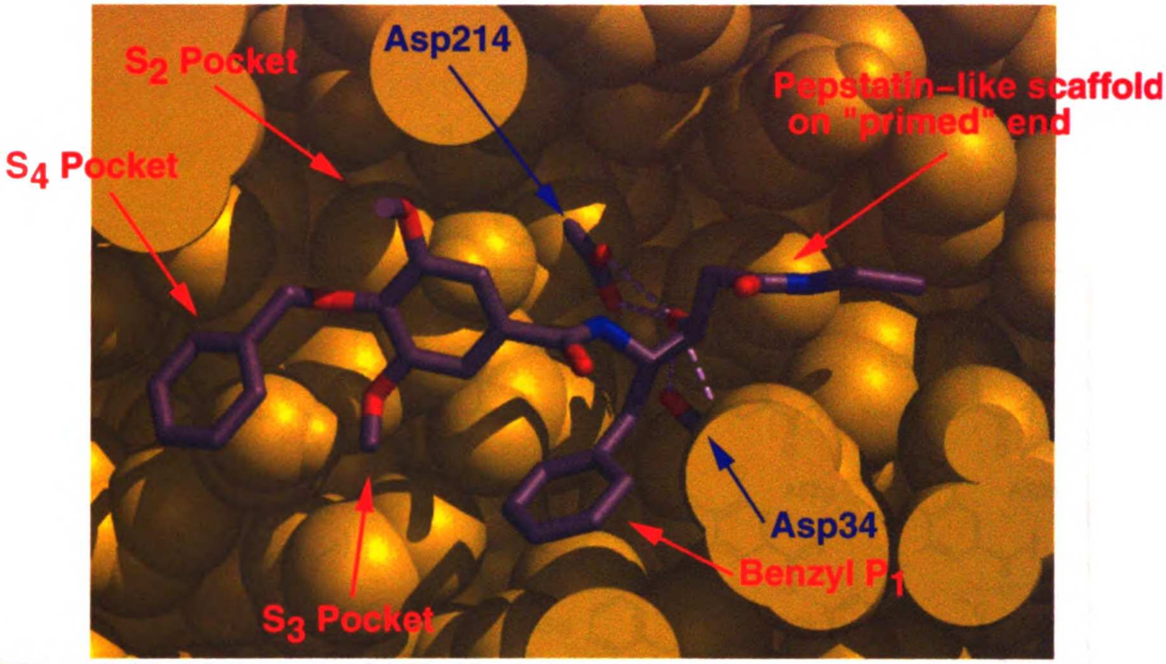


Figure 5. Model structure of A52 side chain.

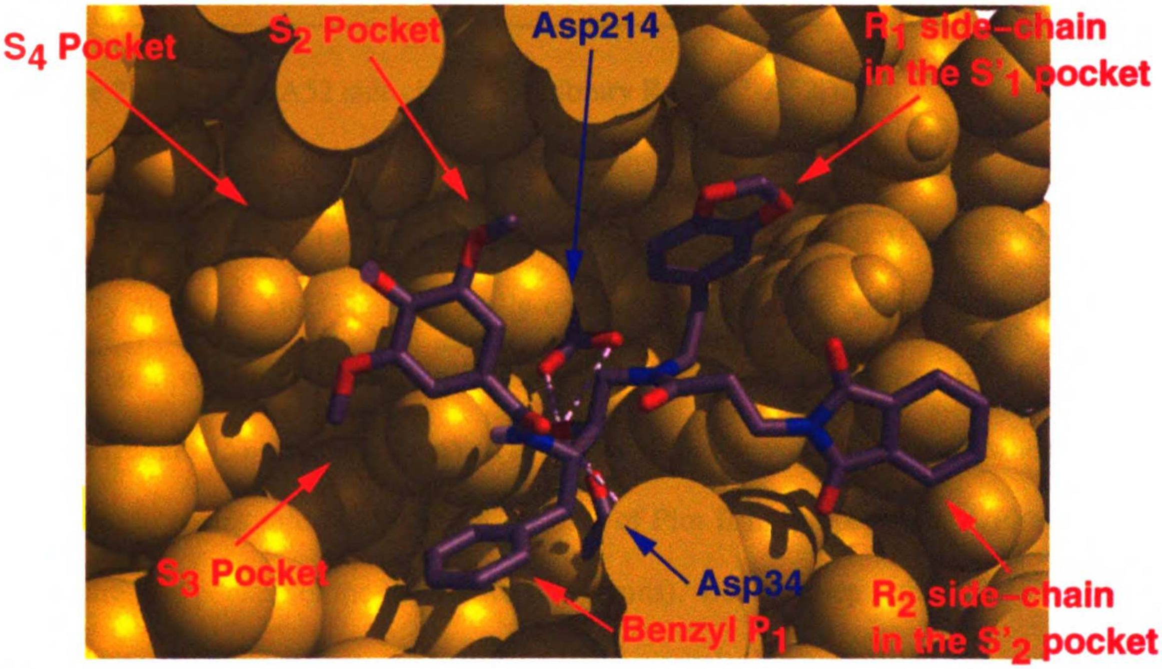

S$_4$ Pocket
S$_2$ Pocket
Asp214
R$_1$ side-chain in the S'$_1$ pocket
S$_3$ Pocket
Asp34
Benzyl P$_1$
R$_2$ side-chain in the S'$_2$ pocket

Figure 6. Crystal structure of compound 3/Plasmepsin II complex (Silva, in preparation). The distal portion of A52 is truncated because it interacts with a crystal contact loop with a symmetry related protein. Note the opening of the S$_1$' and S$_2$' pockets relative to figures 5 and 12.

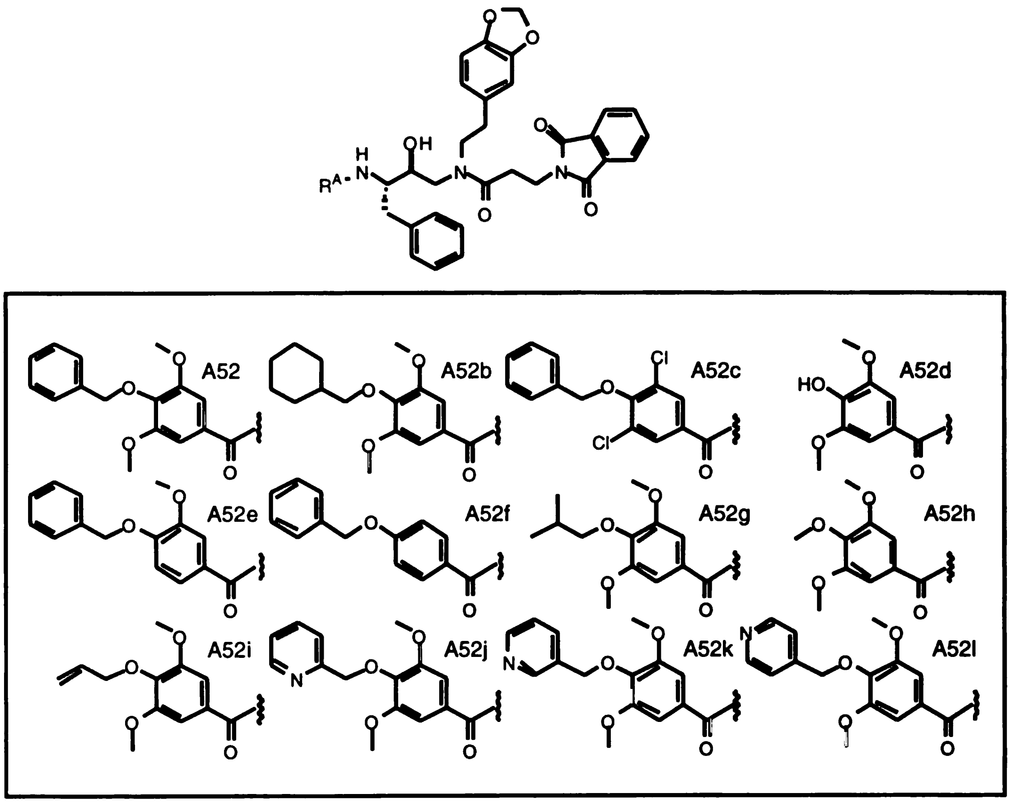


Figure 7. Modified A52 side chains for Library RA (A52-A521).

Library RB. In a similar fashion to the RA Library, we next examined the functionality at the RB site. The chemically compatible amine side chains were clustered to maximize diversity and modeled in the Plm II active site to generate two lists of side chains (Figure 15). It should be noted that some difficulty was experienced when trying to model side chains into the S_1' and S_2' pocket of Plm II in the pepstatin A/Plm II crystal structure (the pocket for the RB and RC side chains). Because pepstatin A does not fill the S_1' subsite, the site is somewhat collapsed in the crystal structure. When the hydroxyethylamine scaffold was modeled in the Plm II active site, there was close contact between the enzyme and the scaffold near the attachment point of the RB and RC side chains. Although the RB and RC side chains could be modeled in, they were highly constrained. We hypoth-

esized that the enzyme might relax significantly to accomodate inhibitors, by opening the "flap" loops that cover the aspartyl protease active site. Indeed, this hypothesis was substantiated by the crystal structure of compound **3** bound to Plm II. (17) In that structure, the closest contact across the S_1' subsite (γ-carbon of V78 to ζ-carbon of F294) was 8.11 Å compared to only 5.4 Å in the pepstatin/Plm II complex. (7)

No novel R_B side chains were identifed that showed significant improvement over the lead compounds. Therefore, we carried the 3,4-methylenedioxyphenethylamine (R_B = B25) into subsequent libraries.

Library R_C. The R_C site was the remaining site to be examined. Due to the difficulty of modeling the R_B and R_C side chains, only a diverse set of side chains was selected for inclusion into the library (Figure 8). Due to chemical compatibility considerations with the reaction conditions used in the library synthesis, the list of diverse side chains for R_C differs from that used for the R_A library. For example, compounds containing reducible functionality were eliminated from the R_C set because of the subsequent $SnCl_2$ reduction required to expose the R_A amine coupling site.

After screening the R_C library, several new side chains were identified as having both good inhibitory potency against the enzyme *and* reduced molecular weight compared to the R_C side chain from the previously identified inhibitors. Compounds containing these side chains were prepared on larger scale and purified for evaluation with Plm II. Of the several new R_C side chains examined in the crude assay, compound **4** containing the *N*-acetyl nipecotic acid side chain (C33) was the most potent, as well as the most promising in terms of potential for future modification via the piperidine amine. Analytically pure **4** with a K_i of 4.8 nM is 20-fold more potent than compound **3** and marked our first single digit nanomolar inhibitor to Plm II (Table 1).

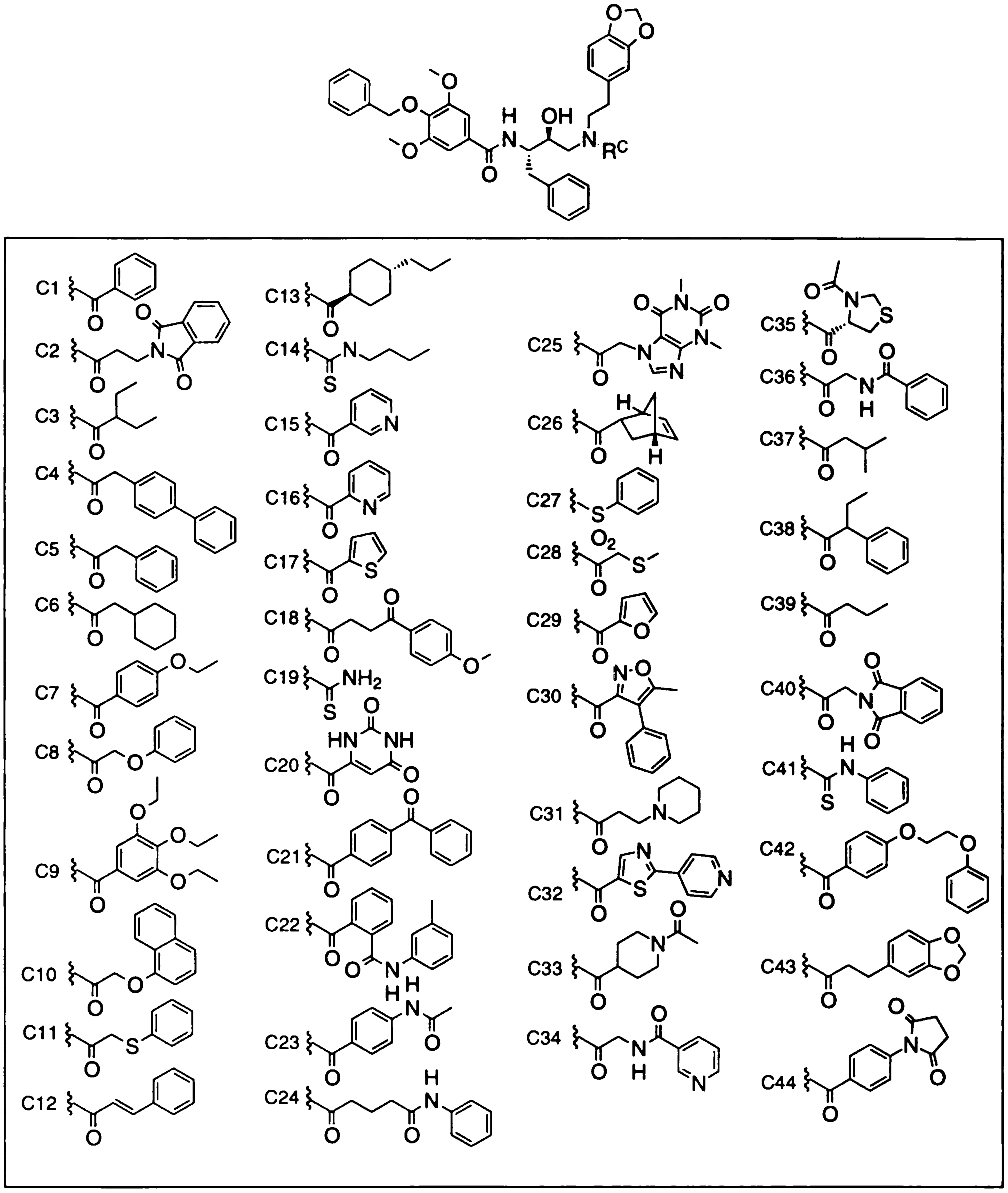


Figure 8. Diverse side chains for library R^C.

Inhibitor Optimization by Combinatorial Evaluation at Multiple Sites

Library RaRc. One of the initial goals of this project was to identify potent, *low molecular weight* inhibitors of Plm II. The compounds identified above, while being potent Plm II inhibitors, do not lie within our desired molecular weight upper limit of 650 Da based on known aspartyl protease drugs. In order to gain insight into the structure-

activity relationship of the Plm II inhibitors with the active site and also to identify low molecular weight compounds, a library of 80 compounds was synthesized where both the RA and RC positions were simultaneously varied (Figure 9). This library allowed us to explore any cooperative effects between the RA and RC sites that might enhance the affinity of the inhibitors for the Plm II active site. While the scaffold weighs only 163 Da, the inclusion of the 4-benzyloxy-3,5-dimethoxybenzoic acid side chain (A52; MW = 288 Da as the free acid) in a potent inhibitor would make it difficult to achieve an overall molecular weight goal of less than 650 Da. Therefore, a wide range of smaller RA side chains was included in an attempt to identify new RARC side chain combinations resulting in smaller inhibitors. Several phenoxyacetic acids that were present in or similar to the initially identified lead inhibitors, were also included at RA (A75-A77).

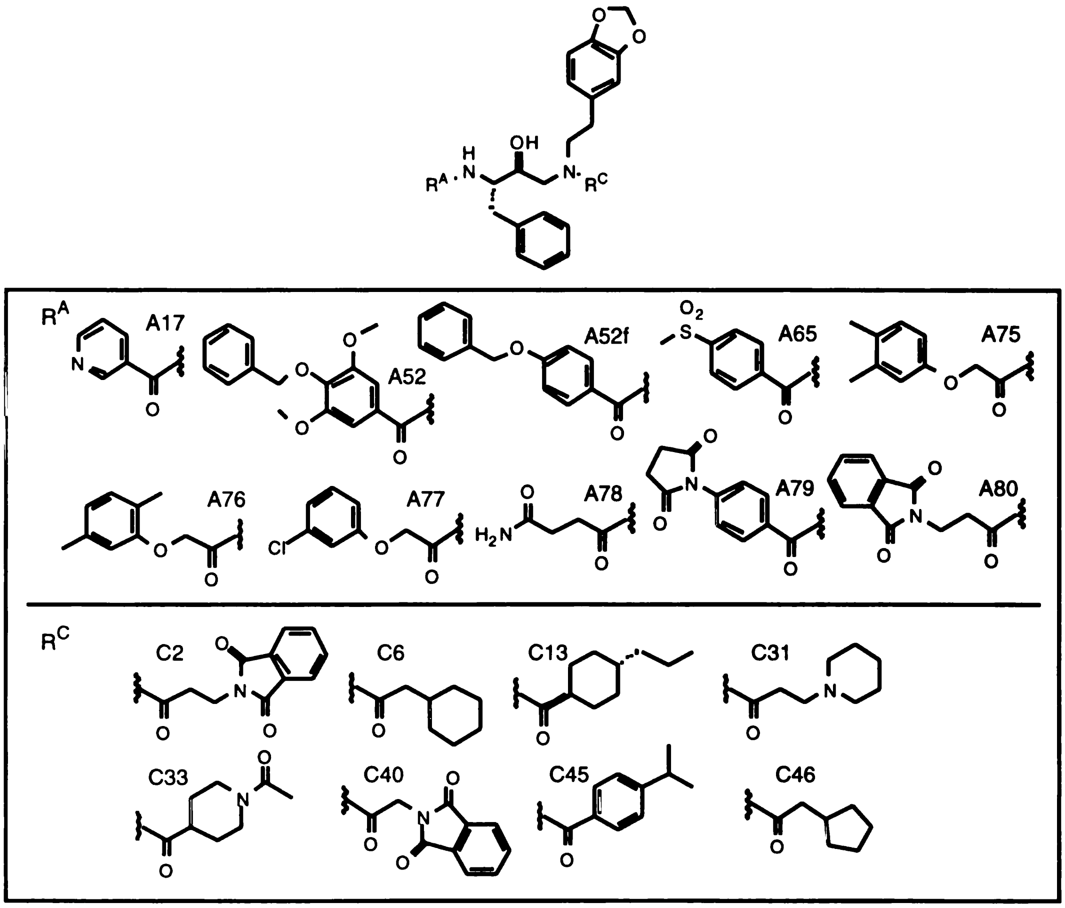


Figure 9. Side chains for Library RARC.

Upon evaluation against Plm II, several trends became clear. First, compounds containing RC side chains related to cyclohexyl carboxylic or substituted acetic acids tended to inhibit Plm II in the fluorogenic assay (C6, C13, C33 and C46, Figure 9), with the *N*-acetyl nipecotic acid side chain (C33) being present in the most potent compounds. Second, at the RA position, the 4-benzyloxy-3,5-dimethoxybenzoic acid side chain (A52) as well as all three of the phenoxyacetic acids side chains (A75-A77) provided potent inhibitors. Based on the crude assay data, accurate K_{i} determinations were performed on analytically pure **5** in Table 1 (K_{i} = 4.0 nM). Inhibitor **5** is slightly more potent than **4**, but

with a significantly reduced molecular weight (650 Da) relative to **4** (752 Da). Compound **6**, the aniline derivative of the 3-chlorophenoxyacetic R_A side chain in **5**, was individually synthesized and evaluated to determine whether a hydrogen bond-accepting group in the R_A side chain would significantly affect the binding of the inhibitors. As demonstrated by the K_i value of 3.0 nM for analytically pure **6**, there is a slight improvement in inhibitory potency upon changing the oxygen to nitrogen in the R_A side chain.

Library $R_AR_BR_CP_1$. The final library simultaneously examined all four sites around the hydroxyethylamine scaffold (Figure 10). The isobutyl moiety (leucine equivalent) was included to attempt to identify smaller Plm II inhibitors. The R_A position was designed to explore several phenoxyacetic acids and cinnamic acids, where the cinnamic acids provide a rigidified phenoxyacetic acid equivalent. At the R_B position, both the 3,4-methylenedioxyphenethylamine (B25) and the smaller 4-methoxyphenethylamine (B55) were included. The R_C collection is primarily focused on exploring isonipecotic acid derivatives (C47, C48) and proline derivatives (C51, C52), cyclohexane carboxylic acid (C49), and isobutyric acid side chains (C50). Nipecotic acid and isonipecotic acid were selected for this position based on good activity of *N*-acetyl nipecotic acid (C33) as well as by modeling the side chain into the crystal structure of **3** bound to Plm II. (17)

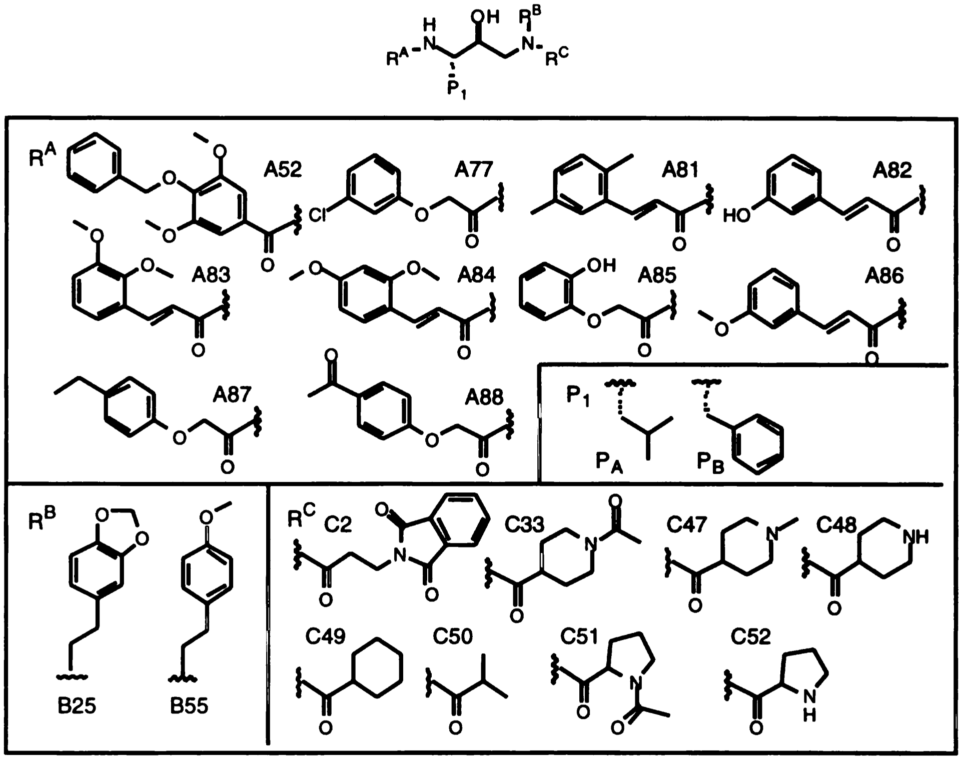


Figure 10. Side chains for Library $R_AR_BR_CP_1$.

The compounds containing the isobutyl P_1 side chain were uniformly less potent in the crude assay than the compounds containing the benzyl P_1 side chain. The preference for benzyl over isobutyl at the P_1 site may be explained by the preferred cleavage site for Plm II (-Phe-Leu-), (18) where the phenylalanine side chain would occupy the P_1 pocket of the protease.

Both R_B side chains in the library resulted in potent inhibitors. The effectiveness of the 4-methoxyphenethylamine (B55) was a moderate step towards our goal of reducing the size of the Plm II inhibitors with no cost in binding affinity, as demonstrated by compounds **9** and **10** (Table 1). There was a preference for isonipecotic acids at R_C, and either A52 or phenoxyacetic acids at R_A, especially the 3-chlorophenoxyacetic acid (A77). Of

the numerous inhibitors identified in the final library, several of the more potent inhibitors were selected for scale-up and purification for exact *K*i determination (compounds **7-11,** Table 1). Analysis of the purified compounds revealed several small, highly potent Plm II inhibitors (compounds **9-11,** Table 1).

Screening Potent Inhibitors for Desired Characteristics

By employing an iterative library design, synthesis and evaluation sequence, multiple single digit nanomolar inhibitors were identified that displayed a significant level of structural variety. This diverse set of inhibitors enabled selection of compounds based on desirable characteristics distinct from potency.

First, we evaluated the most potent Plm II inhibitors for activity against the most highly homologous human protease, cathepsin D. Plm II selectivity would be desired from a therapeutic standpoint, since it may be undesirable to inhibit human cathepsin D. While some of the most potent Plm II inhibitors, **4**, **5**, and **6**, showed no selectivity for Plm II over Cat D, several of the single digit nanomolar, low molecular weight inhibitors, **7**, **8** and **11**, showed up to 15-fold selectivity (Table 1). Comparison of the selective and non-selective inhibitors indicates that the piperidine-based side chains at the Rc position are likely the most important contributor to the observed selectivity.

A representative set of the most potent Plm II inhibitors (compounds **4**, **10**, and **11**) were also assayed in a buffer containing human serum albumin (HSA). Binding to serum proteins could significantly reduce the effectiveness of the inhibitors *in vivo* as has been observed for some HIV protease inhibitors. (19) At 1 mg/mL of HSA, the potency of inhibitors **4**, **10**, and **11** was not effected, indicating that little if any binding to HSA occurs (the fluorescence-based enzyme assay could not be performed at higher concentrations

due to interactions between HSA and the enzyme or substrate). These results suggest that HSA binding will likely not diminish inhibitor potency significantly, even at the high concentrations of HSA found in serum (30-45 mg/mL).

The potent, Plm II inhibitors were also screened for desired calculated properties thought to be important for pharmacokinetics, including calculated logP (ClogP) values, number of hydrogen bond acceptors, and number of hydrogen bond donors (Table 1). Desired ranges for these properties described by Lipinski based upon the evaluation of compound databases include ClogP < 5, number of hydrogen bond donors < 5, number of hydrogen bond acceptors < 10, and molecular weight < 500 Da. (20) A compound that fulfills at least 3 out of the 4 criteria is in agreement with Lipinski's "Rule of 5." The most potent and selective inhibitors, **7**, **8**, and **11**, satisfy 3 out of the 4 criteria (Table 1). Although the molecular weights of the most potent inhibitors (594-650 Da) are above Lipinski's upper limit of 500 Da, it is significant that they are within the molecular weight range of the currrently approved HIV protease inhibitors. (21)

Finally, the top Plm II inhibitors, compounds **9**, **10**, and **11**, were determined to be moderately *more* potent towards Plm I than Plm II enabling simultaneous inhibition of both aspartyl proteases in the parasite digestive vacuole with a *single* inhibitor. These inhibitors were also determined to have 1-2 μM IC_{50} values for inhibition of parasite growth in cultured parasite-infected human erythrocytes.

Conclusion

By applying a rational and directed approach to the iterative design of a small number of libraries, a diverse collection of potent, low molecular weight inhibitors of Plm II were rapidly identified. The use of focused libraries made it possible to identify multi-

ple potent inhibitors with some structural variety, allowing us to select a subset of the potent inibitors that have additional desirable characteristics. For example, inhibitors **5**, **6**, **9**, **10**, and **11** stand out as being of low molecular weight (594 -650 Da), having desirable ClogP characteristics (2.86-4.56 calculated values), having < 5 hydrogen bond donors, having < 10 hydrogen bond acceptors, containing *no* chiral centers outside of the hydroxyethylamine core, and being made up of commercially available side chains. Inhibitors **7** and **11** also show approximately 15-fold selectivity over the most closely related human aspartyl protease cathepsin D. In addition, there is no observable decrease in inhibitory potency for compounds **4**, **10**, and **11** when assayed in buffer containing HSA at 1 mg/mL (the HSA upper limit of the assay), suggesting that Plm II inhibition should not be dramatically effected at *in vivo* HSA concentrations. Notably, compound **11** has all of these desired characteristics with a K_i of 4.3 nM. While in recent years several Plm II inhibitors have been reported, (7,20,22) the compounds discussed in this paper represent the most potent and lowest molecular weight Plm II inhibitors to be reported. We are continuing to examine related compounds in *in vitro* and in cell-based assays. Finally, we believe that the lead identification and optimization protocols described in this report could successfully be applied to rapidly identify potent, low molecular weight, and specific inhibitors to any aspartyl protease.

Experimental Section

Computational methods:

Structure examination. To use a structure-based drug design approach to the optimization libraries, we first examined the crystal structure of a Plm II/pepstatin A complex from *Plasmodium falciparum* (Figure 11). This crystal structure has a resolution of 2.7 Å and

an R value of 0.195. There are two copies ("a" and "b") of the protease in the unit cell, each with a copy of pepstatin A bound in the active site. The unit cell also contained 122 crystallographic waters. The two copies of the protease are quite similar (0.92 Å αC RMSD). We chose to use copy "a" for library screening based on the slightly lower B factors and slightly better bond angles in its copy of pepstatin.

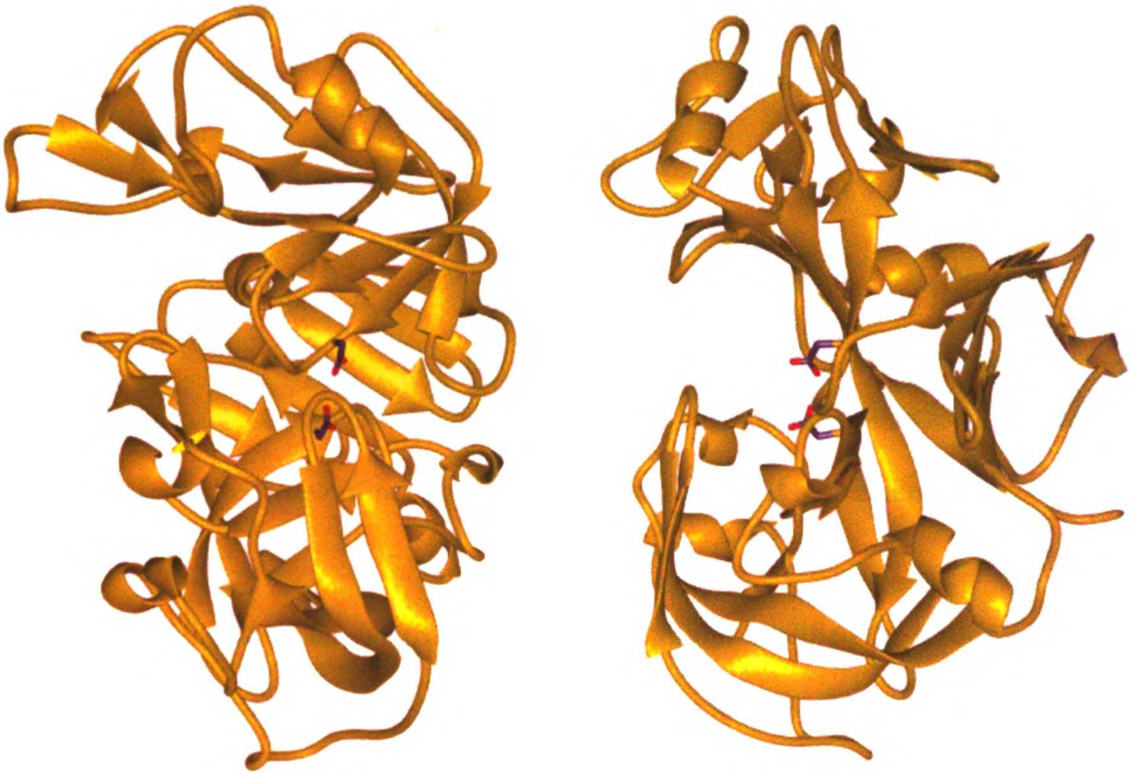


Figure 11. Two views of Plasmepsin II from Plasmepsin II/pepstatin A complex. Catalytic Aspartyl Groups shown in detail.

RA Scaffold generation. Based on previous modeling and crystallography of Cat D, (9,23) we assumed that our hydroxyethylamine inhibitors would bind in a manner similar to the crystallographic pepstatin (Figure 12).

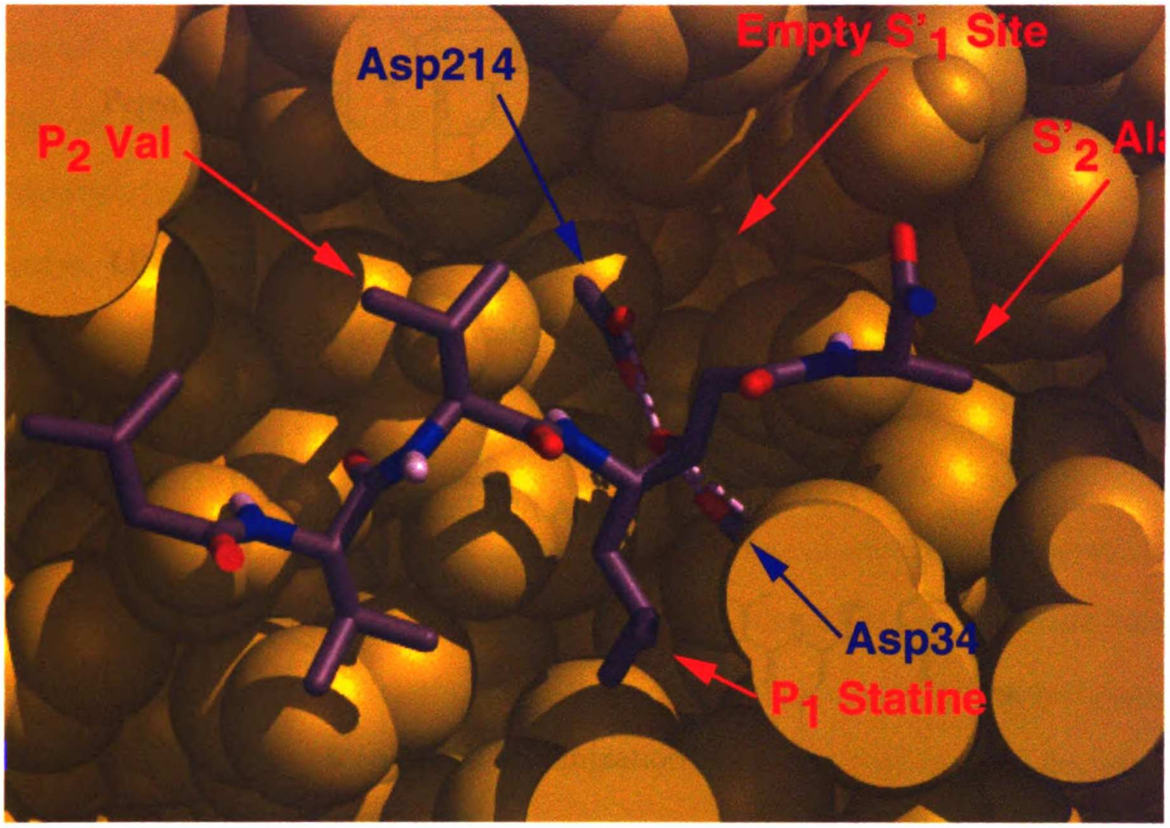


Figure 12. Truncated view of pepstatin A in the Plasmepsin II active site.

To model the RA side-chain, we truncated pepstatin A and modified the statine side chain to a benzyl side chain (Figure 13: Library RA, Steps 1 and 2). This scaffold was minimized for 50 steps within a rigid Plm II active site under the AMBER (24) force field in Sybyl 6.3 (25) (Figure 13: Library RA, Step 3). This scaffold orientation and conformation was used as the basis for all RA library modeling.

case

case

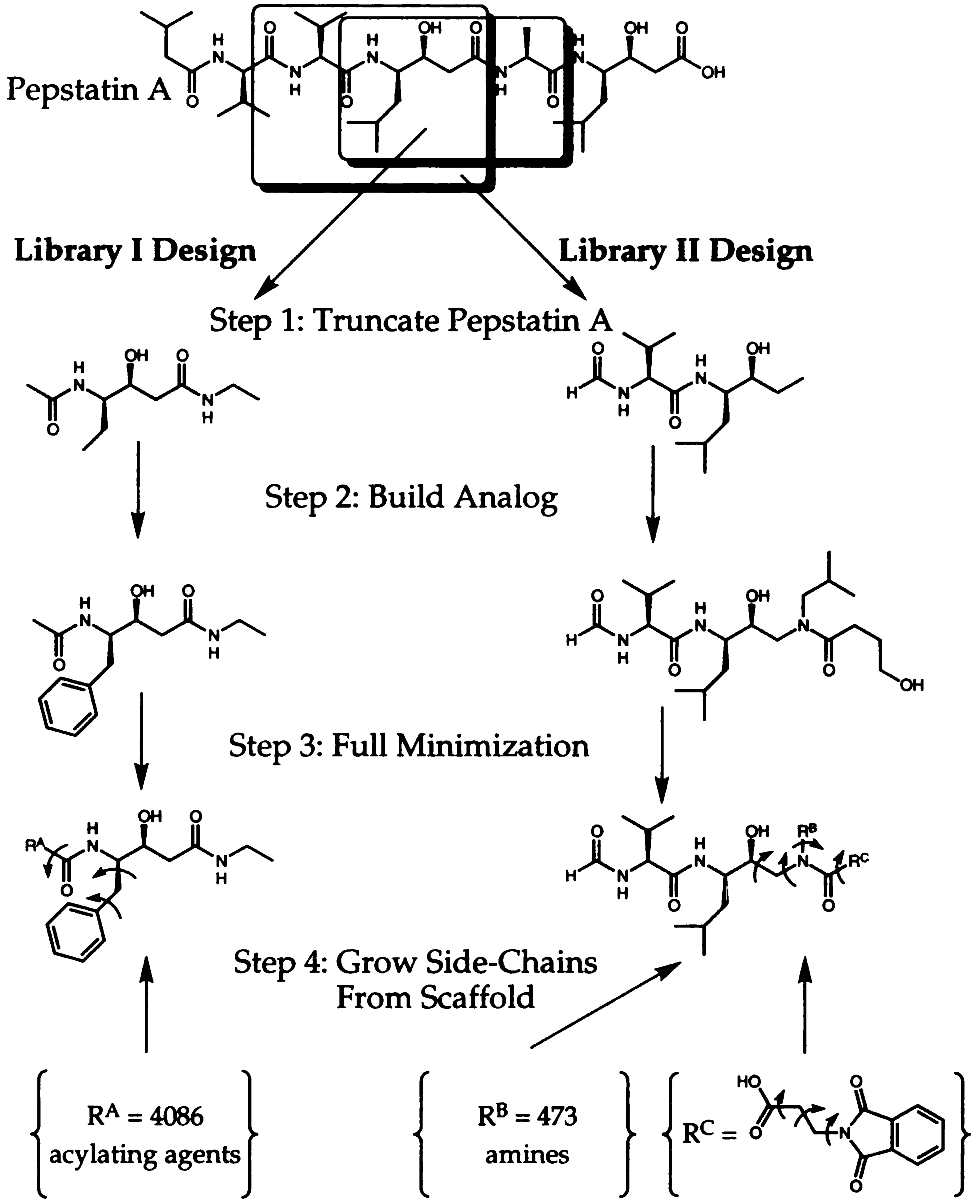

Pepstatin A
Library I Design
Library II Design
Step 1: Truncate Pepstatin A
Step 2: Build Analog
Step 3: Full Minimization
Step 4: Grow Side-Chains From Scaffold
R^A = 4086 acylating agents
R^B = 473 amines
R^C =

Figure 13. Structure-based Design Strategies.

Reagent Selection and Preparation (RA and RC libraries). The web-based program UC_Select, (26) which incorporates Daylight toolkits (27) was used to select potential reagents for each library from the Available Chemicals Directory (ACD). (16) A variety of acylating agents were selected to generate the RA library, including carboxylic acids,

acid halides, isocyanates, isothiocyanates, and sulfonyl halides. Compounds were limited to a 100-300 Da molecular weight range. Compounds with acid halides, anhydrides, azos, sulfonic acids, >1 nitro, epoxides, azides, >4 halides, peroxides, macrocycles, metals, or boron were eliminated by default. Furthermore, compounds with aldehyde, alkyl halide, amine, hydroxylamine, thiol, or multiple acylating functionalities were eliminated. Suppliers were limited to Aldrich, Fluka, Sigma, Calbiochem, ICN, Pfaltz & Bauer, TCI America, Lancaster, Acros Organics, Maybridge International, and Trans World. Final lists included 3458 carboxylic acids, 224 acid halides, 103 isocyanates, 150 isothiocyanates, and 158 sulfonyl halides. Each of these side chains was converted to the appropriate amide derivative using a Daylight toolkit program, (26,27) then Cartesian coordinates were generated for each side chain using distance geometry. Each side chain was then transformed into the orientation of the scaffold and attached by overlaying the appropriate amide of the scaffold with the amide-like bond of the side chain. This resulted in the generation of 4086 unique scaffold side chain compounds.

Similarly for the Rc library, a molecular weight range of 30-275 Da was used. Acylating agents were identified using the screening methods listed above for the RA site, with the additional elimination of side chains containing alcohol, nitro, or phenol groups.

Docking of RA side chains (RA library). The "anchor and grow" algorithm implemented in DOCK (28) was used to model each scaffold and side chain from the virtual library in the Plm II active site (Figure 14). For each side chain, the scaffold, in its minimized pose (scaffold orientation and conformation), was used as the "anchor". The benzyl side chain and the RA side chain were "grown" from this scaffold in the following manner (Figure 13: Library RA, Step 4). First, the molecule's flexibility was pre-analyzed. Each side chain was broken into rigid fragments separated by flexible bonds and a list of preferred torsion

angles for each flexible bond was determined. The molecule was divided into layers starting with the anchor fragment. The first layer contains the rigid fragments which attach to the anchor fragment at each “growth point”. The second layer contains subsequent rigid fragments which attach to fragments from the first layer. Additional layers were generated until the entire molecule had been analyzed. Second, the molecule was "grown" into place. The rigid anchor was placed in its minimized pose (*vide supra*). The fragments from the first layer were placed in each of their acceptable torsion positions. These partial molecules were scored (*vide infra*) and minimized using the six degrees of freedom associated with the pose as well as torsional degrees of freedom from the current layer and the two previous layers. The list of partial molecules was ranked according to score and RMSD from the best scoring partial molecule, and the top 25 partial molecules were used as starting positions for subsequent layers of growth. For each additional layer, all torsions of that layer were again generated, minimized, and ranked. At each layer, the top 25 fragments were saved for further growth. This greedy method resulted in 25 final conformations. The best scoring molecule was saved for comparison with other molecules in the database. For this study, scores consisted of the intermolecular van der Waals and electrostatic terms as well as intramolecular van der Waals and electrostatic terms excluding 1,2 and 1,3 terms from the AMBER force field. (24)

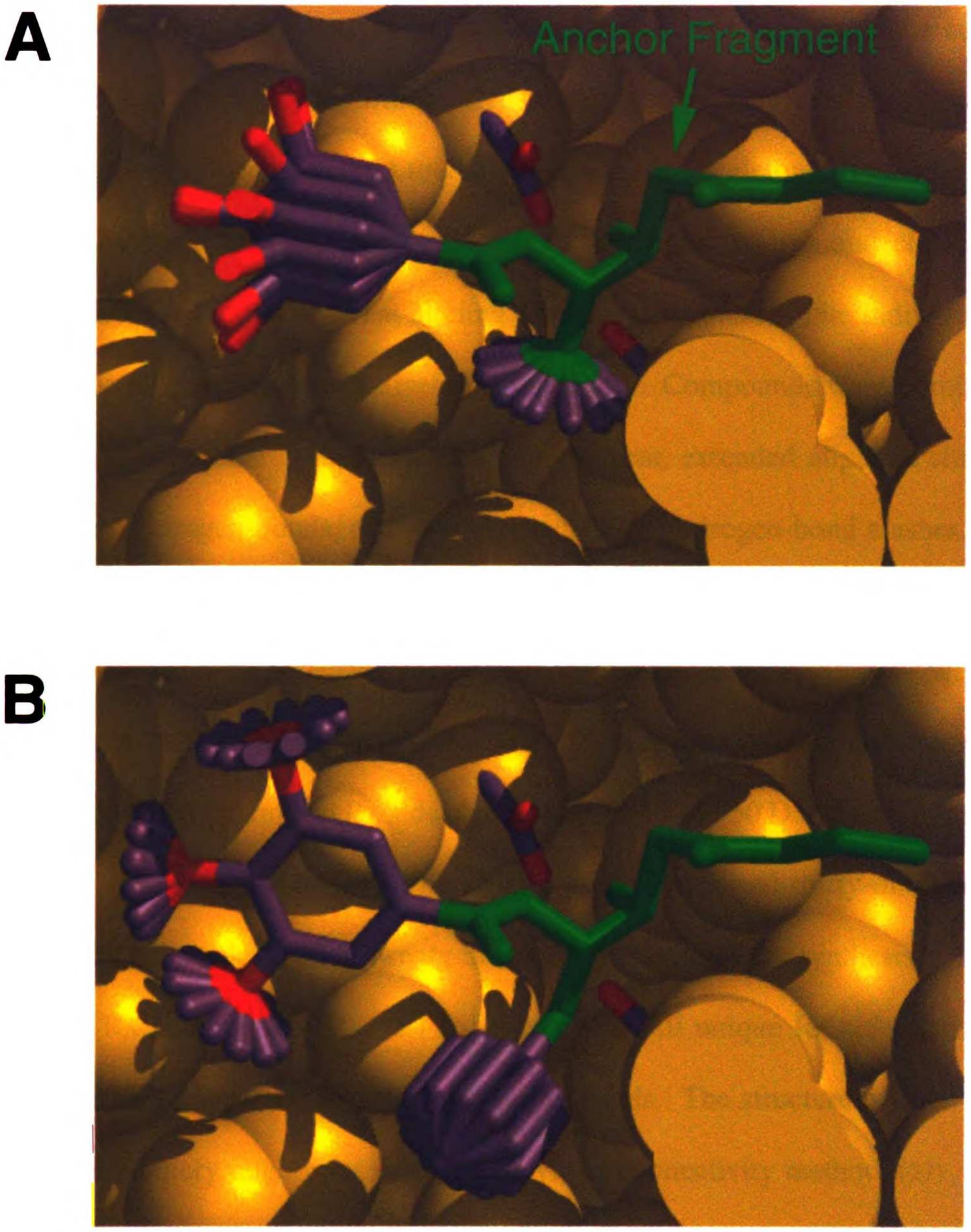


Figure 14. Anchor and Grow Docking method applied to the **RA** library. Modified pepstatin scaffold in green. **A**) The first (proximal) rigid layer is torsionally sampled. The best-scoring conformations are selected and passed to subsequent sampling. **B**) Torsion sampling of the second layer based on the best conformation of the first layer.

The best scoring compounds from the DOCK runs were divided into two categories based on molecular weight ("large" = Da 250-300 and "small" = Da 100-249). The 100 best scoring compounds were examined on graphics and evaluated based on confor-

mation and hydrogen-bond formation. The best 74 compounds were clustered using Daylights connectivity structure measure, (27) a Tanimoto similarity metric, and a complete linkage hierarchical clustering algorithm. This minimal clustering used an intra-cluster similarity of 0.85 which generated 64 total clusters with 55 singletons. The best scoring compound in each cluster was considered the cluster representative and clusters with members predicted to bind in different binding modes were separated into individual clusters. Each remaining molecule was assessed graphically. Compounds were eliminated if they had large hydrophobic groups extending into solvent, extended aliphatic chains, no heteroatoms, no hydrogen-bonds, amino-acid subunits, or hydrogen-bond clashes. There were 30 "large" compounds and 5 "small" compounds remaining. The top scoring 25 "large" compounds and the 5 "small" compounds were selected for synthesis. We sought twenty additional "small" compounds. The next 250 best scoring compounds were then analyzed. Sixty-nine of them had R_A side chains with a MW < 250 Da. These compounds were assessed graphically with the same criteria used above, and thirty-one additional compounds were selected. The top twenty of these were selected for synthesis.

Clustering of R_A side chains (R_A and R_C libraries). All unique R_A side chains were clustered to identify a representative set of 50 compounds. The structure of each reagent was encoded in a binary fingerprint using the Daylight connectivity method. (27) An all-by-all similarity-distance matrix was calculated using a Tanimoto similarity metric. (27) Complete-linkage hierarchical clustering was carried out until only 50 clusters remained. This corresponded to a minimum internal cluster similarity of 0.1659 and no singletons. One compound for each cluster was chosen as a representative of that cluster. A similar clustering method was applied to side chains for the R_C site in the R_C library, using a modified list of acylating agents (as discussed above).

RB and RC scaffold generation (RB and RC libraries). Similarly to the RA library, the scaffold for RB and RC libraries was generated by starting with the crystallographic structure of pepstatin A bound to Plm II. Pepstatin A was truncated as shown in Figure 13 (Libraries RB and RC, Step 1). A hydroxyethylamine scaffold was then built and leucine and serine side chains were attached at the RB and RC side chain positions respectively (Figure 13: Libraries RB and RC, Step 2). These side chains were minimized in a rigid receptor similarly to the RA scaffold (Figure 13: Libraries RB and RC, Step 3). The side chains were then truncated and the resulting fragment orientation and conformation was used as the scaffold for all RB and RC libraries.

RB reagent selection (RB library). Similarly to RA and RC reagents, RB reagents were selected using the UC_Select program. Primary amines were selected from the ACD 95.1. In addition to the defaults used in RA and RC selection, compounds with alcohol, aldehyde, alkyl halide, amino acid, carboxylic acid, hydroxylamine, nitro, phenol, thio, phosphoric acid or ester and phosphonic acid or ester functional groups were eliminated. Suppliers were limited as with the RA and RC reagents. Furthermore, reagents were limited to a molecular weight range of 30-275 Da. This selection process generated a list of 473 potential amine side chains for the RB library.

Docking of RB side chains (RB library) (Figure 15). For each RB reagent, RC was fixed with the 3-phthalimidopropionic acid side chain (C2, Figure 15) Molecules were generated by attaching the appropriate RB side chain and the RC = C2 side chain to the support. Docking was started with the rigid scaffold anchor described above. The RB and RC side chains were grown in layer by layer as described for the RA library (Figure 13: Libraries RB and RC, Step 4).

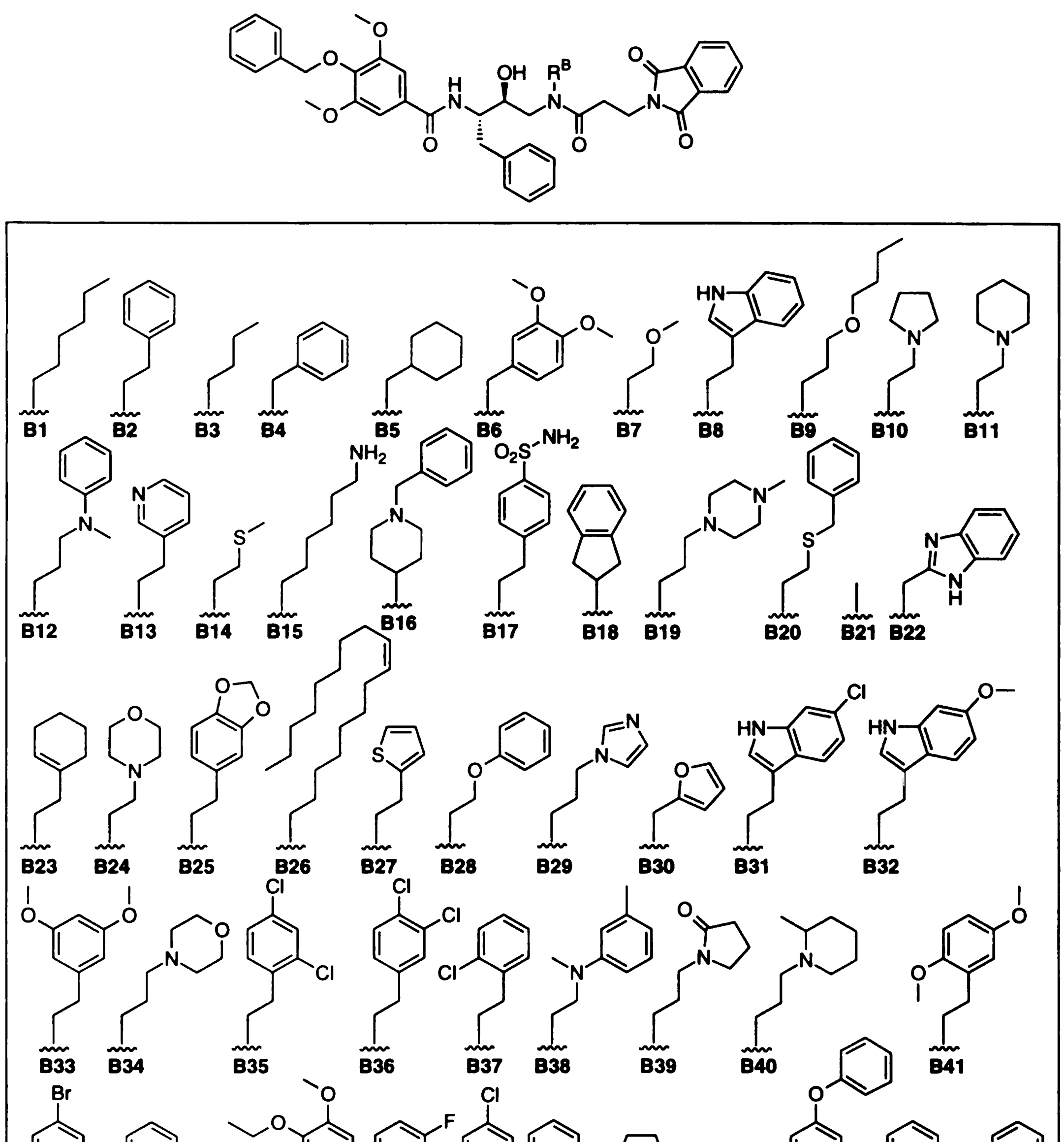


Figure 15. Diverse (B1-B30) and modeled (B31-B54) side chains for Library R^B.

The 100 top-scoring compounds were examined visually and clustered as described

above. Sixteen of these top scoring compounds had already been used in the initial screening libraries. After removing compounds which did not pass the criteria described for the RA library, the top 47 compounds were selected for synthesis.

Clustering RB side chains (RB library) (Figure 15). The 473 RB reagents were clustered in a manner similar to the RA and RC reagents. The 50 RB clusters, however, contained 13 singletons, with only 71 reagents in the largest cluster. The minimum internal similarity among the 50 clusters was 0.36.

Calculation of ClogP Values. A SMILES string for each compound in its neutral form was generated. Structures were made from each SMILES with the PRADO utility and checked visually for accuracy. (27) Finally, we used the ClogP program to calculate an octanol/water partition coefficient for each compound based on its SMILES string. (27)

Synthesis methods:

Library synthesis (Method 1). Libraries RA and RB were synthesized on solid support as reported previously (Figure 2). (13) Side chains with questionable stability were tested by exposure to trifluoroacetic acid for 1 h at room temperature (equivalent to extreme cleavage conditions). Side chains with questionable acylation/amine displacement characteristics were checked via incorporation into minimal hydroxyethylamine test compounds. Compounds were then cleaved from support and validated by TLC and MALDI-MS. Libraries were synthesized in a spatially separate array in a 96-well format, using commercially available Merrifield resin (Novabiochem). Note that libraries derived from docked and/or diverse side chains often contained several components less than the number listed in the computational methods, since some of the selected side chains were either back-ordered or no longer available. Efforts were made to replace such side chains with

other side chains from the same diverse cluster whenever possible. Acylating side chains containing alcohols or phenols were incorporated into libraries R_A and R_B using 0.3 M side chain, 0.3 M dicyclohexylcarbodiimide (DCC), 0.3 M N-hydroxysuccinimide (HOSu) and 0.9 M *i*Pr$_2$EtN in *N*-methylpyrrolidinone (NMP) for a minimum of 4 h. Several side chains (particularly cinnamic acid derivatives and *N*-phthaloyl-β-alanine) were coupled using 0.3 M O-(7-azabenzotriazol-1-yl)-1,1,3,3-tetramethyluronium hexafluorophosphate (HATU) instead of PyBOP due to observed precipitation in the PyBOP coupling solution. Random compounds from each library (10-15% of total library) were checked by MALDI-MS for presence of desired molecular ion peak. The desired molecular ion peak was observed in 90-100% of the compounds checked.

Library synthesis (Method 2). The R_C, R_AR_C and $R_AR_BR_CP_1$ libraries were synthesized on solid support as recently reported. (14) The P_1 side chain is incorporated by Grignard addition to a support-bound pyrrolidine amide (Figure 2). After diastereoselective reduction of the resulting ketone, nosylation and azide displacement of the secondary alcohol, and removal of the primary alcohol protecting group and subsequent nosylation, the support-bound scaffold segment is provided. Modifications of coupling conditions similar to those used in Method 1 (*vide supra*) were also applied to libraries synthesized using Method 2.

High through-put Plm II assay. Compounds were tested for inhibitory activity against Plm II in 96-well microtiter plates using a fluorometric high through-put assay. (29) The fluorogenic substrate used in the assay was DABCYL-GABA-Glu-Arg-Nle-Phe-Leu-Ser-Phe-Pro-EDANS. (30) The assay was performed in Microfluor "W" (DYNEX Technologies, Inc.: Chantilly, VA) fluorescence microtiter plates, and readings were obtained on a

Perkin-Elmer LS-50B fluorometer with an attached 96-well plate reader. An excitation wavelength of 336 nm and emission wavelength of 490 nm were used, with a 430 nm emission cutoff filter in place. Typical substrate concentrations of 1 Mand Plm II concentrations of 1.25-2.5 nM were used. Assays were performed in a 0.1 M sodium acetate buffer (pH = 5.0) containing 10% glycerin, and 0.01% Tween-20. All libraries were initially assayed using crude products at 1 Minhibitor concentration and assuming a 50% overall yield. Active compounds were further diluted and assayed. Inhibitors were dissolved in a DMSO stock solution prior to addition to the buffer. Assays were performed in 5% DMSO to ensure dissolution of the inhibitors.

*K*i Determinations

The Plm II assays for the fully characterized compounds were performed in a quartz cuvette (Starna Cells, Inc.: Atascadero, CA) with a Perkin-Elmer LS-50B spectrometer (Excitation = 336 nM, Emission = 490 nM). The substrate (Bachem California, Inc.: Torrence, CA) used was as reported for the high-throughput assay. A 0.1 M sodium acetate buffer (pH = 5.0) with 10% glycerin and 0.01% Tween-20 was used with a final concentration of 0.6 - 0.7 nM Plm II. The Plm II pro-enzyme is stored in H20 and activated in the sodium acetate buffer (pH = 5.0) described above. In a typical assay with a final volume of 600 µL, to 545 µL of buffer is added 30 µL of inhibitor in DMSO, followed by 15 µL of Plm II stock (buffer). The mixture is incubated at 25 Cfor 4.5 minutes followed by the addition of 10 µL of substrate stock (buffer). The change of fluorescence intensity was recorded as a function of time. Assays were performed in duplicate or triplicate at five - six inhibitor concentrations for each *K*i determination. Data were fitted by nonlinear regression analysis to the equation derived by Williams and Morrison. (31)

$$v = \frac{v_o}{2E_t} * \left(\sqrt{\left(K_i \left(1 + \frac{S}{K_m} \right) + I_t - E_t \right)^2 + 4K_i \left(1 + \frac{S}{K_m} \right) E_t} - \left(K_i \left(1 + \frac{S}{K_m} \right) + I_t - E_t \right) \right)$$

The K_m for the substrate was determined to be 0.75 μM by using a Lineweaver-Burke plot (the K_m was previously determined by Goldberg and coworkers to be 0.96 μM). (8) The variables S, E_t and It are the concentrations of substrate, active enzyme, and inhibitor, respectively.

The Plm II assays containing human serum albumin (HSA) were performed using the same conditions described above except the to the sodium acetate buffer was added 1 mg of HSA (nondenatured from Calbiochem: San Diego, CA) per 1 mL of buffer. Control experiments performed with only enzyme and substrate excluding inhibitor, in buffer with HSA concentrations above 1 mg/mL resulted in a constant or even decrease in fluorescence intensity over a 20-minute time period. This is presumably due to HSA interacting with either the enzyme or substrate.

The human liver cathepsin D (Calbiochem: San Diego, CA) assays were performed in a similar fashion to the Plm II assays. The same substrate that was used in the Plm II experiments was employed for the cathepsin D assays. A 0.1 M formic acid buffer (pH = 3.7) was used with a final concentration of 0.7 nM cathepsin D. In a typical assay with a final volume of 600 μL, to 560 μL of buffer is added 20 μL of inhibitor in DMSO, followed by 10 μL of cathepsin D stock (0.05% Triton X-100 in H20). The mixture is incubated at 25 C for 4.5 minutes followed by the addition of 10 μL of substrate stock (DMSO). The K_m was determined to be 0.87 μM using a Lineweaver-Burke plot.

Plasmodium falciparum-infected erythrocyte assay. Assays of inhibitory activity in cultured parasite-infected human erythrocytes were performed as previously described. (18)

Scaled-up synthesis of inhibitors. Selected inhibitors were synthesized on larger scale for purification and exact K_i determinations. Compounds were typically synthesized on a 20-30 mg scale on solid support, using the previously described library synthesis Method 1 (Figure 2). The compounds were then cleaved from support and purified by silica gel flash chromatography to yield the desired pure compounds. Special care was taken to promptly remove residual acid and immediately purify compounds containing the 4-benzyloxy-3,5-dimethoxybenzoic acid side chain, as moderate loss of the terminal benzyl group was observed under the 1:1 trifluoroacetic acid:1,2-dichloroethane cleavage conditions.

Compound 1. $_1$H NMR (400 MHz, $CDCl_3$), δ (ppm) 2.10 (m, 1 H), 2.25 (s, 3 H), 2.29 (s, 3 H), 2.40-2.81 (m, 3 H), 2.95 (m, 3 H), 3.38 (m, 2 H), 3.64 (m, 1 H), 3.82 (m, 3 H), 4.30 (m, 1 H), 4.45 (m, 2 H), 5.91 (m, 2 H), 6.41-6.57 (m, 2 H), 6.63 (d, *J* = 7.8 Hz, 1 H), 6.72 (bd, *J* = 7.1 Hz, 1 H), 7.02 (d *J* = 7.5 Hz, 1 H), 7.13-7.30 (m, 8 H), 7.72 (m, 2 H), 7.83 (m, 2 H). FAB-HRMS: calculated for $C_{40}H_{42}N_3O_8$ ($M+H_+$) = 692.29719, observed = 692.29892. Anal. ($C_{40}H_{41}N_3O_8{\cdot}0.5H_2O$) C, H, N.

Compound 2. $_1$H NMR (300 MHz, $CDCl_3/CD_3OD$), δ (ppm) 2.40 (m, 2 H), 2.58 (t, *J* = 7.0 Hz, 2 H), 2.82 (m, 2 H), 2.98 (dd, *J* = 3.4, 14.1 Hz, 1 H), 3.31-3.46 (m, 3 H), 3.74-3.88 (m, 3 H), 4.13 (m, 1 H), 4.32 (d, *J* = 14.8 Hz, 1 H), 4.41 (d, *J* = 14.8 Hz, 1 H), 5.84 (s, 2 H), 6.41 (dd, *J* = 1.6, 7.9 Hz, 1 H), 6.49 (d, *J* = 1.6 Hz, 1 H), 6.60 (d, *J* = 7.9 Hz, 1 H), 6.74 (dd, *J* = 1.9, 8.3 Hz, 1 H), 6.89 (m, 1 H), 6.94 (d, *J* = 8.0 Hz, 1 H), 7.07-7.22 (m, 6 H), 7.67

(m, 2 H), 7.78 (m, 2 H). FAB-HRMS: calculated for $C_{38}H_{37}N_3O_8Cl$ $(M+H^+)$ = 698.22692, observed = 698.22560. Anal. ($C_{38}H_{36}N_3O_8Cl$) C, H, N.

Compound 3. ^{1}H NMR (400 MHz, $CDCl_3$), δ (ppm) 2.47 (m, 2 H), 2.57 (t, *J* = 7.2 Hz, 2 H), 2.85 (d, *J* = 7.9 Hz, 2 H), 2.93 (m, 1 H), 3.21 (m, 1 H), 3.51 (m, 1 H), 3.64 (s, 3 H), 3.75 (m, 1 H), 3.83 (s, 3 H), 4.12 (m, 2 H), 5.75 (bs, 1 H), 5.91 (dd, *J* = 1.5, 3.1 Hz, 2 H), 6.43 (d, *J* = 7.7 Hz, 1 H), 6.53 (m, 3 H), 6.65 (d, *J* = 7.8 Hz, 1 H), 7.07-7.39 (m, 12 H), 7.72 (m, 2 H), 7.84 (m, 2 H). FAB-HRMS: calculated for $C_{46}H_{46}N_3O_{10}$ $(M+H^+)$ = 800.31832, observed = 800.31861. Anal. ($C_{46}H_{45}N_3O_{10}$) C, H, N.

Compound 4. ^{1}H NMR (400 MHz, $CDCl_3$) δ (ppm) 1.04 (bs, 1 H), 1.27 (m, 1 H), 1.41-1.70 (m, 1 H), 2.03 (d, *J* = 2.9 Hz, 3 H), 2.25 (m, 1 H), 2.36 (m, 1 H), 2.67 (m, 2 H), 2.83 (m, 2 H), 3.00-3.14 (m, 2 H), 3.70-3.95 (m, 3 H), 3.85 (s, 6 H), 4.28 (m, 1 H), 4.55 (bd, *J* = 14.0 Hz, 1 H), 5.04 (s, 2 H), 5.91 (s, 2 H), 6.45 (d, *J* = 7.7 Hz, 1 H), 6.55 (d, *J* = 1.7 Hz, 1 H), 6.61 (d, *J* = 9.4 Hz, 1 H), 6.70 (d, *J* = 7.9 Hz, 1 H), 6.91 (d, *J* = 3.1 Hz, 1 H), 7.24-7.34 (m, 8 H), 7.46 (d, *J* = 6.6 Hz, 2 H). Anal. ($C_{43}H_{49}N_3O_9{\bullet}0.8H_2O$) C, H, N.

Compound 5. ^{1}H NMR (400 MHz, $CDCl_3$) δ (ppm) 1.35-1.80 (m, 5 H), 2.03 (d, *J* = 3.9 Hz, 3 H), 2.20 (m, 1 H), 2.36 (m, 1 H), 2.65 (m, 2 H), 2.80-2.95 (m, 4 H), 3.35-3.50 (m, 2 H), 3.77 (m, 1 H), 4.20 (dd, *J* = 8.0, 8.9 Hz, 1 H), 4.39 (d, *J* = 14.8 Hz, 1 H), 4.47 (d, *J* = 14.8 Hz, 1 H), 4.48 (m, 1 H), 4.96 (bd, 1 H), 5.91 (s, 2 H), 6.44 (m, 1 H), 6.54 (m, 1 H), 6.71 (d, *J* = 7.8 Hz, 1 H), 6.79 (d, *J* = 7.0 Hz, 1 H), 6.92 (s, 1 H), 7.00 (m, 1 H), 7.15-7.27 (m, 5 H). Anal. ($C_{35}H_{40}N_3O_7Cl$) C, H, N.

Compound 6. ^{1}H NMR (400 MHz, $CDCl_3$) δ (ppm) 1.26 (m, 1 H), 1.40-1.62 (m, 3 H), 1.82 (bs, 1 H), 2.05 (s, 3 H), 2.20 (m, 1 H), 2.36 (m, 1 H), 2.67 (m, 2 H), 2.80-2.99 (m, 4 H), 3.45 (m, 2 H), 4.17 (dd, J = 7.6, 16 Hz, 1 H), 4.45 (m, 1 H), 4.53 (bd, J = 13 Hz, 1 H), 4.86 (bd, J = 29 Hz, 1 H), 5.93 (s, 2 H), 6.44 (m, 2 H), 6.56 (m, 2 H), 6.73 (m, 2 H), 6.98

(dd, J = 4.4, 9.6 Hz, 1 H), 7.08 (t, J = 8.0 Hz, 1 H), 7.17-7.32 (m, 5 H). Anal. ($C_{35}H_{41}N_4O_6Cl\bullet1.3H_2O$) C, H, N.

Compound 7. $_1$H NMR (400 MHz, $CDCl_3$) δ (ppm) 1.18-1.45 (bm, 1 H), 1.64-1.87 (bm, 4 H), 2.02 (m, 1 H), 2.24 (bs, 3 H), 2.66 (m, 2 H), 2.83 (m, 2 H), 3.03 (d, *J* = 7.2 Hz, 2 H), 3.09 (d, *J* = 13.4 Hz, 1 H), 3.45 (m, 1 H), 3.47 (m, 3 H), 3.75 (s, 6 H), 3.77 (m, 1 H), 4.27 (m, 1 H), 5.04 (s, 2 H), 6.66 (d, *J* = 9.1 Hz, 1 H), 6.81 (d, *J* = 9.4 Hz, 2 H), 6.92 (s, 2 H), 6.94 (d, *J* = 8.6 Hz, 2 H), 7.18-7.35 (m, 8 H), 7.46 (d, *J* = 6.6 Hz, 2 H). Anal. ($C_{42}H_{51}N_3O_7$) C, H, N.

Compound 8. $_1$H NMR (400 MHz, $CDCl_3$) δ (ppm) 1.19 (bd, 1 H), 1.34 (bd, 1 H), 1.50 (bq, 1H), 1.90-2.20 (m, 4 H), 2.43 (m, 2 H), 2.67 (m, 2 H), 3.00-3.11 (m, 4 H), 3.43 (m, 3 H), 3.74 (s, 3 H), 3.79 (m, 1 H), 3.85 (s, 6 H), 4.30 (m, 1 H), 5.04 (s, 2 H), 6.88 (m, 4 H), 6.94 (s, 2 H), 6.95 (d, *J* = 8.5 Hz, 2 H), 7.15-7.34 (m, 7 H), 7.46 (d, *J* = 7.2 Hz, 2 H). Anal. ($C_{41}H_{49}N_3O_7\cdot0.9F_3CCO_2H$) C, H, N.

Compound 9. $_1$H NMR (400 MHz, $CDCl_3$) δ (ppm) 1.40 (m, 1 H), 1.67-1.95 (m, 4 H), 2.02 (m, 1 H), 2.23 (s, 3 H), 2.62 (t, *J* = 7.0 Hz, 2 H), 2.83-3.02 (m, 6 H), 3.34-3.46 (m, 2 H), 3.76 (m, 2 H), 4.19 (dd, *J* = 8.9, 8.9 Hz, 1 H), 4.38 (d, *J* = 14.8 Hz, 1 H), 4.46 (d, *J* = 14.8 Hz, 1 H), 5.12 (bs, 1 H), 5.90 (s, 2 H), 6.44 (dd, *J* = 1.6, 8.0 Hz, 1 H), 6.53 (d, *J* = 1.6 Hz, 1 H), 6.70 (d, *J* = 7.8 Hz, 1 H), 6.78 (dd, *J* = 2.0, 8.0 Hz, 1 H), 6.91 (t, *J* = 2.2 Hz, 1 H), 6.99 (d, *J* = 6.0 Hz, 2 H), 7.16-7.27 (m, 6 H). Anal. ($C_{34}H_{40}N_3O_6Cl$) C, H, N.

Compound 10. $_1$H NMR (400 MHz, $CDCl_3$) δ (ppm) 1.35 (m, 3 H), 1.52 (m, 1 H), 1.68 (bq, 2 H), 1.84 (m, 2 H), 2.27 (bs, 1 H), 2.64 (m, 2 H), 2.82-2.95 (m, 5 H), 3.04 (q, *J* = 7.4 Hz, 1 H), 3.39-3.47 (m, 2 H), 3.75 (s, 3 H), 3.77 (m, 1 H), 4.20 (dd, *J* = 8.7, 8.7 Hz, 1 H), 4.38 (d, *J* = 14.8 Hz, 1 H), 4.46 (d, *J* = 14.8 Hz, 1 H), 5.11 (bs, 1 H), 6.80 (m, 3 H), 6.93 (m, 3 H), 6.99 (d, *J* = 8.8 Hz, 2 H), 7.18-7.28 (m, 5 H). Anal.

($C_{34}H_{42}N_3O_5Cl•2F_3CCO_2H$) C, H, N.

Compound 11. $_1$H NMR (400 MHz, $CDCl_3$) δ (ppm) 1.18 (m, 1 H), 1.38 (m, 1 H), 1.50-1.85 (m, 2 H), 2.17 (m, 1 H), 2.54-2.70 (m, 4 H), 2.87-2.98 (m, 4 H), 3.19 (bd, *J* = 13.0 Hz, 1 H), 3.42-3.60 (m, 2 H), 3.72 (s, 3 H), 3.73-3.80 (m, 1 H), 4.19 (dd, *J* = 8.6, 8.6 Hz, 1 H), 4.41 (d, *J* = 15.0 Hz, 1 H), 4.48 (d, *J* = 15.0 Hz, 1 H), 5.18 (bs, 1 H), 6.79 (m, 4 H), 6.96 (m, 5 H), 7.15-7.25 (m, 5 H). Anal. ($C_{33}H_{41}N_3O_5Cl_2•0.7F_3CCO_2H$) C, H, N.

Compound 12. $_1$H NMR (400 MHz, $CDCl_3$) δ (ppm) 2.45 (m, 1 H), 2.60 (m, 2 H), 3.01 (d, *J* = 7.0 Hz, 1 H), 3.12 (d, *J* = 8.5 Hz, 2 H), 3.74 (m, 1 H), 3.86 (m, 3 H), 3.93 (s, 3 H), 4.25 (m, 1 H), 5.21 (s, 2 H), 5.91 (d, *J* = 3.8 Hz, 2 H), 6.43 (d, *J* = 7.9 Hz, 1 H), 6.51 (s, 1 H), 6.64 (d, *J* = 7.8 Hz, 2 H), 6.87 (d, *J* = 8.4 Hz, 1 H), 7.17-7.28 (m, 8 H), 7.36 (m, 4 H), 7.70 (m, 2 H), 7.81 (m, 2 H). Anal. ($C_{45}H_{43}N_3O_9$) C, H, N.

References:

(1) Wyler, D. J. Malaria-Overview and Update. *Clin. Infect. Dis.* **1993**, *16*, 449-458.

(2) *Malaria : Obstacles and Opportunities : a Report of the Committee for the Study on Malaria Prevention and Control: Status Review and Alternative Strategies, Division of International Health, Institute of Medicine.* Oaks, Jr., S. C.; Mitchell, V. S.; Pearson, G. W.; Carpenter, C. C. J., Eds. Washington, D.C.: National Academy Press, 1991.

(3) Foote, S. J.; Cowman, A. F. The Mode of Action and the Mechanism of Resistance to Antimalarial Drugs. *Acta Tropica* **1994**, *56*, 157-171.

(4) Murray, M. C.; Perkins, M. E. Chemotherapy of Malaria. *Annu. Rep. Med. Chem.* **1996**, *31*, 141-150.

(5) Francis, S. E.; Sullivan, Jr., D. J.; Goldberg, D. E. Hemoglobin Metabolism in the Malaria Parasite *Plasmodium falciparum*. *Annu. Rev. Microbiol.* **1997**, *51*, 97-123.

(6) Goldberg, D. E. Hemoglobin Degradation in Plasmodium-Infected Red Blood Cells. *Semin. Cell Biol.* **1993**, *4*, 355-361.

(7) Silva, A. M.; Lee, A. Y.; Gulnik, S. V.; Majer, P.; Collins, J.; Bhat, T. N.; Collins, P. J.; Cachau, R. E.; Luker, K. E.; Gluzman, I. Y.; Francis, S. E.; Oksman, A.; Goldberg, D. E.; Erickson, J. W. Structure and Inhibition of plasmepsin II, a Hemoglobin-Degrading Enzyme from *Plasmodium falciparum*. *Proc. Natl. Acad. Sci. USA* **1996**, *93*, 10034-10039.

(8) Luker, K. E.; Francis, S. E.; Gluzman, I. Y.; Goldberg, D. E. Kinetic Analysis of Plasmepsins I and II, Aspartic Proteases of the *Plasmodium falciparum* Digestive Vacuole. *Mol. Biochem. Parasitol.* **1996**, *79*, 71-78.

(9) Kick, E. K.; Roe, D. C.; Skillman, A. G.; Liu, G.; Ewing, T. J. A.; Sun, Y.; Kuntz, I. D.; Ellman, J. A. Structure-Based Design and Combinatorial Chemistry Yield Low Nanomolar Inhibitors of Cathepsin D. *Chem. Biol.* **1997**, *4*, 297-307.

(10) Kuntz, I. D.; Blaney, J. M.; Oatley, S. J.; Langridge, R.; Ferrin, T. E. A Geometric Approach to Macromolecule-Ligand Interactions. *J. Mol. Biol.* **1982**, *161*, 269-288.

(11) Meng, E. C.; Gschwend, D. A.; Blaney, J. M.; Kuntz, I. D. Orientational Sampling and Rigid-Body Minimization in Molecular Docking. *Proteins: Structure, Function & Genetics* **1993**, *17*, 266-278.

(12) Ewing, T. J. A.; Kuntz, I. D. Critical Evaluation of Search Algorithms for Automated Molecular Docking and Database Screening. *J. Comp. Chem.* **1992**, *18*, 1175-1189.

(13) Kick, E. K.; Ellman, J. A. Expedient Method For The Solid-Phase Synthesis Of Aspartic Acid Protease Inhibitors Directed Toward The Generation Of Libraries. *J. Med. Chem.* **1995**, *38*, 1427-1430.

(14) Lee, C. E.; Kick, E. K.; Ellman, J. A. General Solid-phase Synthesis Approach To

Prepare Mechanism-Based Aspartyl Protease Inhibitor Libraries. Identification of Potent Cathepsin D Inhibitors. *J. Am. Chem. Soc.* **1998,** *120,* 9735-9747.

(15) Side chains for both the 1000 compound Cat D library and the 39 compound Cat D library are in Chapter 7(Figures 4 and 6).(9)

(16) Available Chemicals Database, version 95.1, Molecular Design Limited: San Leandro, CA, USA.

(17) Silva, A., personal communication.

(18) Gluzman, I. Y.; Francis, S. E.; Oksman, A.; Smith, C. E.; Duffin, K. L.; Goldberg, D. E. Order and Specificity of the *Plasmodium falciparum* Hemoglobin Degradation Pathway. *J. Clin. Invest.* **1994,** *93,* 1602-1608.

(19) Olson, R. E.; Christ, D. D. Plasma Protein Binding of Drugs. *Annu. Rep. Med. Chem.* **1996,** *31,* 327-336.

(20) Lipinski, C. A.; Lombardo, F.; Dominy, B. W.; Feeney, P. J. Experimental and Computational Approaches to estimate Solubility and Permeability in Drug Discovery and Development Settings. *Adv. Drug Deliv. Rev.* **1997,** *23,* 3-25.

(21) Kaldor, S. W.; Kalish, V. J.; Davies, J. F.; Shetty, B. V.; Fritz, J. E.; Appelt, K.; Burgess, J. A.; Campanale, K. M.; Chirgadze, N. Y.; Clawson, D. K.; Dressman, B. A.; Hatch, S. D.; Khalil, D. A.; Kosa, M. B.; Lubbehusen, P. P.; Muesing, M. A.; Patick, A. K.; Reich, S. H.; Su, K. S.; Tatlock, J. H. Viracept (Nelfinavir Mesylate, AG1343): A Potent, Orally Bioavailable, Inhibitor of HIV-1 Protease. *J. Med. Chem.* **1997,** *40,* 3979-3985 and references therein.

(22) Carroll, C. D.; Patel, H.; Johnson, T. O.; Guo, T.; Orlowski, M.; He, M.; Cavallaro, C. L.; Guo, J.; Oksman, A.; Gluzman, I. Y.; Connelly, J.; Chelsky, D.; Goldberg, D. E.; Dolle, R. E. Identification of Potent Inhibitors of *Plasmodium falciparium* Plasmepsin II

from an Encoded Statine Combinatorial Library. *Bioorg. Med. Chem. Lett.* **1998**, *8*, 2315-2320

(23) Erickson, J. W., personal communication.

(24) S. J. Weiner; P. A. Kollman; D. A. Case; U. C. Singh; C. Ghio; G. Alagona; S. Profeta; P. A. Weiner A New Force Field for Molecular Mechanical Simulation of Nucleic Acids and Proteins. *J. Am. Chem. Soc.* **1984**, *106*, 765-784.

(25) Sybyl, version 6.3, TRIPOS Associates: St. Louis, MO.

(26) Skillman, A. G.; Kuntz, I. D., unpublished results.

(27) ClogP, Fingerprint, Nearneighbors, Merlin, Prado, Rubicon, and Toolkit Software, version 4.51, Daylight Chemical Information Systems, Inc.: Santa Fe, NM.

(28) Ewing, T. J. A.; Kuntz, I. D., unpublished results.

(29) Matayoshi, E. D.; Wang, G. T.; Krafft, G. A.; Erickson, J. Novel Fluorogenic Substrates for Assaying Retroviral Proteases by Resonance Energy Transfer. *Science*, **1990**, *247*, 954-958.

(30) Substrate was initally synthesized as described for similar fluorogenic substrates in Krafft, G. A.; Wang, G. T. Synthetic Approaches to Continuous Assays of Retroviral Proteases. *Methods Enzymol.* **1994**, *241*, 70-86, and later purchased (Bachem California Inc.: Torrance, CA).

(31) Williams, J. W.; Morrison, J. F. The Kinetics of Reversible Tight-Binding Inhibition. *Methods Enzymol.* **1979**, *63*, 437-467.

Epilogue to Chapter 8

Several interesting results related to our work on the plasmepsin II protein have recently developed. First, the complete genome of *Plasmodium falciparum* has recently been sequenced and several new proteases were identified which may be involved in hemoglobin degradation (Daniel Goldberg, personal communication). Eventually, these proteases may help clear up some of the confusing cell-culture data we have seen with our inhibitors. Second, during this compound, a compound which is not a protease inhibitor, yet which is extremely potent against malaria parasites in cell culture was identified. Work to optimize this compound and identify its molecular target is underway. Finally, during this project, we also developed several potent cathepsin D inhibitors which also fulfill the Lipinski criteria. Although these compounds are not significantly more potent than those previously identified, they work much better in cell culture. Our collaborator, Gary Lynch, has recently shown that several of these potent, Lipinski compatible cathepsin D inhibitors are capable of preventing neurofibrillary tangle development in a rat brain-culture model of Alzheimer's disease (see appendix 1, part 4).

Prologue to Chapter 9

I ask that my reader now indulge me as I summarize, criticize, second guess, and speculate about each of the projects in my thesis and the lessons I learned participating in them. I have attempted, however unsuccessfully, to keep my summaries to a minimum. In each case I attempt to offer both scientific and procedural criticisms which I hope would lend aid to those who might find themselves working on similar projects. Finally, for each project, I attempt to point out specific technical advances which would improve the design tools, or specific experiments which would yield decisive results. I took some latitude in expressing these opinions (without preliminary data) and one should consider them a personal wish-list rather than a specific directive on how to proceed.

Chapter 9: Conclusions and Future Directions

Combinatorial chemistry makes it possible to efficiently synthesize large libraries of ligands. Merging combinatorial chemistry strategies with structure-based design principles allows the exploration of virtual libraries containing billions of compounds. We have developed three tools for the efficient construction and design of combinatorial libraries. UC_Select is an internet-based tool that allows synthetic chemists to select reagents for a virtual library (Chapter 2). Diversify is a program that allows a computational chemist to generate virtual libraries for molecular docking (Chapter 4). CombiDock is a variation of the structure-based design program DOCK, which has been optimized for designing combinatorial libraries (Chapter 3). We described these tools and demonstrated the application of structure-based design and combinatorial chemistry to the inhibition of HIV-1 Reverse Transcriptase (Chapter 5), *T. foetus* Hypoxanthine-Guanine-Xanthine-Phosphoribosyltransferase (Chapter 6), Cathepsin D (Chapter 7), and *P. falciparum* Plasmepsin II (Chapter 8). The tools discussed here can be used to rapidly design potent inhibitors of medicinally important enzymes.

Chapter 2 described UC_Select, an internet-based common nomenclature interface to help synthetic chemists specifically and sensitively select reagents for combinatorial synthesis. UC_Select is an excellent basic tool and has been well received by chemists in academia as well as industry. UC_Select allows rapid design of libraries which can actually by synthesized from available reagents. This facilitates direct proposal of testable hypothesis by computational chemists designing combinatorial libraries.

Although its simplicity contributes to the utility of UC_Select, it is also one of its drawbacks. UC_Select would benefit from further development of its post-selection library analysis. Simple histograms and graphs of physical properties such as molecular

weight, log P, or rotatable bonds would help chemists address questions about the composition of the results of their UC_Select search. Further development of hitlist tailoring functions based on these properties could also be beneficial. In the future, a Java applet should be developed to allow the user to actively refine the physical property profiles of their final hitlists. Although search parameters are stored by UC_Select at the server site, and although the user can save and re-read compound hitlists, one flaw in UC_Select is that the user cannot save and recall search parameters on their local disk. This function will be essential for UC_Select to be used in the context of generating and updating a database of combinatorial libraries. Finally integrating UC_Select with structure generation algorithms, such as CONCORD or Daylight's Rubicon, would allow MOL2 or PDB format structures to be downloaded directly. UC_Select is a useful tool for chemists to select reagents for combinatorial libraries. Additional features would help advanced users, however, a basic non-intimidating version of the interface should always be available for novice or computationally-wary users.

CombiDock, an efficient algorithm for structure-based design of combinatorial libraries was described in chapter 3. CombiDock was validated by demonstrating greater than 12 fold enrichment of cathepsin D inhibitors among a hydroxyethyl(amine) combinatorial library. CombiDock's efficiency has allowed us to carry out the first example of screening a database of large combinatorial libraries. These initial experiments have shown the dramatic effect of Lipinski's "rule of five" on scaffold selection. Additionally, we have shown that some two component libraries score as well as three component libraries even though they contain orders of magnitude fewer members. CombiDock was designed primarily to screen multiple large libraries. For small or single component librar-

ies, particularly when the scaffold orientation is known (see chapters 7 and 8), a more detailed treatment of conformational flexibility, including torsion minimization is computationally feasible and can be achieved with Dock 4.0. Although it is tempting to try to include detailed conformational minimization in a program for library database screening, such as CombiDock, this is currently not computationally feasible.

In the future, additional experiments to investigate the relative importance of side chain versus scaffold selection in library design should be carried out. This endeavor would be greatly enhanced by implementation of a scoring function without the dependence on size seen with force-field scoring. The GBSA method of Zou(*1*) fulfils this role, but unfortunately, it is not separable into molecular fragment terms. In addition, because of the need to reproduce and expand individual library dockings from a database screening, further advancement will also necessitate implementation of a minimization method not dependent on random number generation. Furthermore, as was first stated by Brian Shoichet nearly nine years ago, the Dock algorithm, and the CombiDock algorithm as well, are inherently parallel, and could easily benefit from even minimal parallelization.

Finally, the body of published synthesis of combinatorial libraries has grown dramatically(*2, 3*) and construction of a database of available combinatorial libraries (ACL) from the ACD using UC_Select and Diversify could greatly enhance the utility of CombiDock. I believe the culmination of these improvements should be an effort to use CombiDock to generate the small molecule equivalent of structural genomics(*4, 5*) (*e.g.* an ongoing screening all available combinatorial libraries against targets from every structural class). This endeavor will complement the efforts to describe and characterize all

binding sites(*6*, *7*, *8*), and will provide a starting place for ligand design efforts against newly identified gene targets based on their homology to targets which have already been screened with CombiDock against the continuously-expanding ACL.

A critical bottleneck in any combinatorial library design is manipulation of the reagents and scaffolds to form whole molecules or fragments appropriate for screening. Diversify, the program presented in chapter 4, is a Daylight toolkit(*9*) solution to library construction, but it is also a general chemical engine capable of modeling both synthetic and biosynthetic chemistries. Diversify benefits and is limited by its Daylight dependencies. The two most significant flaws in Diversify are that it modifies molecules in a graph based rather than a conformation based representation and that it has a limited number of transforms. Diversify is currently a helpful tool, but could be made invaluable by implementation of 3D structure manipulation. The underlying datastructures are capable of supporting cartesian coordinates so this would not be an insurmountable task. Although generation and validation of an individual Diversify transform takes only a few minutes for someone familiar with SMILES and SMARTS, encoding the thousands of reactions available to synthetic chemists remains a daunting project. This could be alleviated by acquisition of one of the available reaction databases, such as SPRESI(*9*). As part of a larger suite of programs, Diversify can allow a scientist to efficiently address questions about combinatorial libraries. Finally, out of obligation to my original vision and the inspiration for Diversify's implementation, I must mention that Diversify could be incorporated into a *de novo* design program. Diversify could act as the mutation function, modifying or joining molecular fragments according to the rules of organic synthesis.

The second portion of my thesis described application of structure-based design to identify novel inhibitors of four medicinally important targets; HIV-1 RT, *Tritrichomonas foetus* HGXPRTase, Human Cathepsin D, and *Plasmodium falciparum* Plasmepsin II. Three of these targets (RT, HGXPRTase, and Plasmepsin II) are relevant to infectious diseases (viral or parasitic), while the fourth (Cathepsin D) has been implicated in Alzheimer's disease and may also have a role in metastatic melanoma.

In the first application, we identified a series of naphthyl sulfonic acid urea inhibitors of HIV-1 RT. This project was a "classic" application of DOCK to screen a rigid representation of the ACD, with only subsequent use of combinatorial libraries for optimization. Through the work of several experimental collaborators, we showed that this class of molecules inhibits RT by a novel mechanism of action, preventing nucleic acid binding and RNase H activity in addition to the polymerase activity inhibited by all currently approved RT inhibitors.

This project was hampered by a lack of meaningful structural information about our compounds binding to RT and by the poor medicinal and solubility properties of our lead compound. If this project was repeated today, it is unlikely that the same series of compounds would be selected for development naphthyl sulfonic acids are not drug-like molecules. Structural information could still provide great benefits to this project, since we only have hypothetical models of compound binding. For the future, we proposed RT mutants which should differentiate among our hypothetical binding modes. In addition, the recent development of GBSA scoring(*1*) may assist binding mode analysis of these highly charged compounds. Knowledge of the binding mode and binding site would allow

investigation of novel classes of compounds which bound to the same site and thus might share some of the same-very interesting-inhibitory qualities. Despite starting with a crystal structure of only 3.0A resolution, we successfully identified a class of molecules with a novel mechanism of inhibition of HIV-1 RT. These compounds may provide a starting point for medicinally interesting new sets of HIV-1 RT inhibitors.

The *T. Foetus* HGXPRTase is an model system for protozoal parasites, including *G. lamblia* and *S. mansonii* and inhibiting it may lead to new treatments for the prevalent human diseases caused by these organisms. In the first phase of identifying HGXPRTase inhibitors we used DOCK 3.5 to screen a rigid representation of the ACD. In the second phase of our approach, we used two procedures novel to our group. First, we proposed binding mode hypothesis by flexibly redocking the inhibitors. We tested these hypothesis using similarity and substrate searching to identify available analogs which could differentiate between specific binding hypothesis. This strategy, yielded over an order of magnitude improvement over our initial leads as well as narrowing our possible binding hypothesis. Second, we selected compounds to be assayed based partially upon the ability to generate combinatorial libraries from the compound. This resulted in straightforward design of two follow-up libraries (one presented in chapter 4), resulting in more than a ten fold improvement in binding. Despite these advances, the best compounds are only moderately potent. Our best binding models predicted the inhibitors bind to the guanine site, and indeed our compounds are competitive inhibitors with respect to guanine in the forward reaction and GMP in the reverse reaction. However, they are *not* competitive inhibitors with respect to PRPP as predicted by the same models. Like the HIV-1 RT project, this project is in need of validation by experimental structural data. Tragically, the data for

one of the compounds bound to the homologous giardial enzyme was collected to high resolution, but never solved. This lack of structural validation makes designing optimization libraries (*vida supra*) highly speculative. It is unfortunate that there is not a more accessible, less committing experiment to validate computationally predicted binding modes than protein x-ray crystallography and protein NMR. In the future, multiple binding mode hypothesis creation and cross validation should be integrated into structure-based design software such as DOCK. Not only should this aid in selecting initial compounds, but it may also make strong predictions about which chemical modifications might elucidate a structure-activity relationship.

The final two application projects targeted a human and a malarial enzyme respectively. They are very similar and represent a progression of methods in structure-based library design. Both targets are homologous aspartyl proteases and both inhibitor libraries were based on the hydroxyethyl(amine) scaffold. The first paper was a landmark work showing that a structure-based design (work of Diana C. Roe) identified more potent inhibitors than diverse design (my work). This result came at a time when some felt that combinatorial chemistry, high-throughput screening and diverse library design would render structure-based design obsolete. Although it has subsequently been shown that the Jarvis-Patrick clustering algorithm I used is not the most effective method(*10*), the diverse library achieved astoundingly good results (<300nM inhibitor in the first round library) which only served to emphasize just how well the structure-based library design performed.

All 1000 compounds in the structure-based library were assayed against human cathepsin D, yielding 23 potent inhibitors. Identifying the 23 inhibitors from among the 1000 designed compounds has been an excellent test case for novel library design algorithms. However, some significant biases of this test case should be acknowledged including: first, the side chain binding sites, upon which the 1000 compounds were differentiated, was quite hydrophobic, while the scaffold binding site, which was common to all ligands, was quite polar, giving undue emphasis to modeling of hydrophobic effects while electrostatic information could be misleading; second, based on a requirement of the CombiBuild protocol, each side-chain has four or fewer rotatable bonds, and thus all of the molecules are relatively rigid; finally, all 30 side-chains in the 1000 compounds scored well in Diana's structure-based screening, so ranking these 1000 may indicate the additional discriminating power relative to the original method, rather than the absolute discriminating power of a new method.

For the second project in this series, we started by assaying the 1000 compounds in the designed library against a related aspartyl protease, plasmepsin II. This project came later chronologically and benefited from more mature library design tools and we developed potent and selective inhibitors of both cathepsin D and plasmepsin II which also passed the Lipinski criteria for crossing biological membranes. Although both of these projects were successful, they both emphasized a frustrating weakness in our screening protocol. After the initial design, we did not have the means to contribute very much additional information. The best subsequent structural insight came by hand building analogs in a crystal structure of our initial lead. Although higher level computational methods could be used to calculate detailed binding energies, these calculations all take much too

long to be practical in an ongoing design setting. In the future, in order to give further insight into molecular optimization, DOCK would benefit from a more detailed scoring function as well as automated binding mode SAR analysis. The GBSA scoring method of Zou *et. al.* (*1*) may address a portion of this issue, but it remains unproven in this arena.

We have implemented a suite of software tools which allow efficient design and execution of experiments involving combinatorial libraries. UC_Select helps one design virtual compounds which can be expediently synthesized. Diversify reliably generates the modified library representation necessary for screening. Finally, DOCK and CombiDock screen libraries of compounds to identify those most complimentary to the binding site. Furthermore, they generate binding mode hypothesis for each compound. Together, these tools can be used to design synthetically accessible, potent, and selective inhibitors of medicinally important targets. These library design experiments and the inhibitors they identify yield insights into our understanding of how small molecules bind to macromolecular targets and how modifications of these small molecules may alter binding.

Bibliography

1. Zou, X. Q., Sun, Y. X., and Kuntz, I. D. (1999) *Journal Of the American Chemical Society 121*, 8033-8043.

2. Dolle, R. E. (1998) *Molecular Diversity 3*, 199-233.

3. Dolle, R. E., and Nelson, K. H. (1999) *Journal Of Combinatorial Chemistry 1*, 235-282.

4. Kim, S. H. (1998) *Nature Structural Biology 5*, 643-645.

5. Burley, S. K., Almo, S. C., Bonanno, J. B., Capel, M., Chance, M. R., Gaasterland, T., Lin, D. W., Sali, A., Studier, F. W., and Swaminathan, S. (1999) *Nature Genetics 23*, 151-157.

6. Briem, H., and Kuntz, I. D. (1996) *Journal of Medicinal Chemistry 39*, 3401-3408.

7. Oshiro, C. M., personal communication (1999).

8. Arnold, J., personal communication (1999).

9. Daylight Chemical Information Systems, Inc., Daylight Ver. 4.6, Santa Fe, N. M.

10. Matter, H. (1997) *Journal of Medicinal Chemistry 40*, 1219-1229.

Appendix 1

Other Publications Contributing Toward the Doctor of Philosophy

A Rapid Method for Exploring the Protein Structure Universe

Published with Malin M. Young and Irwin D. Kuntz in Proteins, February 15, 1999, 34(3):317-332.

We have developed an automatic protein fingerprinting method for the evaluation of protein structural similarities based on secondary structure element compositions, spatial arrangements, lengths, and topologies. This method can rapidly identify proteins sharing structural homologies as we demonstrate with five test cases: the globins, the mammalian trypsinlike serine proteases, the immunoglobulins, the cupredoxins, and the actinlike ATPase domain-containing proteins. Principal components analysis of the similarity distance matrix calculated from an all-by-all comparison of 1,031 unique chains in the Protein Data Bank has produced a distribution of structures within a high-dimensional structural space. Fifty percent of the variance observed for this distribution is bounded by six axes, two of which encode structural variability within two large families, the immunoglobulins and the trypsinlike serine proteases. Many aspects of the spatial distribution remain stable upon reduction of the database to 140 proteins with minimal family overlap. The axes correlated with specific structural families are no longer observed. A clear hierarchy of organization is seen in the arrangement of protein structures in the universe. At the highest level, protein structures populate regions corresponding to the all-alpha, all-beta, and alpha/beta superfamilies. Large protein families are arranged along family-specific axes, forming local densely populated regions within the space. The lowest level of

Novel Cathepsin D Inhibitors Block the Formation of Hyperphosphorylated Tau Fragments in Hippocampus

Submitted to The Journal of Chemical Physics with Xiaoning Bi, Tasir S. Haque, Jun Zhou, Bin Lin, Tina Lee, Irwin D. Kuntz, Jonathan A. Ellman, and Gary Lynch.

Lysosomal disturbances may be a contributing factor to Alzheimer's disease. We used novel compounds to test if suppression of the lysosomal protease cathepsin D blocks production of known precursors to neurofibrillary tangles. Partial lysosomal dysfunction was induced in cultured hippocampal slices with a selective inhibitor of cathepsins B and L. This led within 48 hours to hyperphosphorylated tau protein fragments recognized by antibodies against human tangles. Potent, nonpeptide cathepsin D inhibitors developed using combinatorial chemistry and structure-based design blocked production of the fragments in a dose dependent fashion. Threshold was in the submicromolar range with higher concentrations producing complete suppression. The effects were selective and not accompanied by pathophysiology. Comparable results were obtained with three structurally distinct inhibitors. These results support the hypothesis that cathepsin D links lysosomal dysfunction to the etiology of Alzheimer's and suggest a new approach to treating the disease.

Docking Flexible Molecules with DOCK4.0

Submitted to the Journal of Computer-Aided Molecular Design
with Todd J. A. Ewing and Irwin D. Kuntz.

An incremental construction strategy for docking flexible molecules to a macromolecular site is developed. The search algorithm is developed to optimize speed and accuracy while maintaining simplicity of control. It is tested with a panel of 15 testcases, created from 12 unique crystallographic complexes whose ligands vary in size and flexibility. For all testcases, at least one docked position is generated within 2 Angstroms of the crystallographic position. For 7 of 15 testcases, the top scoring position is also within 2 Angstroms of the crystallographic position. The algorithm is fast enough to successfully dock a few testcases within seconds and most within 100 seconds. The search algorithm is also tested with a database of 51 molecules docked to two of the crystallographic testcases. The software is fast enough to reliably rank the database of compounds within 15 seconds/molecule. The search algorithm is incorporated into version 4.0 of the DOCK suite of molecular modeling software.

Appendix 2: UC_Select Program

UC_Select Perl 5.003 Code:

```
#!/usr/bin/perl
#
# DIVERSIFY: UC_Select 1.01
#
# 9/98 - fixed SETHITLIST, ADDSUPPLIER, and SUPPLIERS so they work
# together properly. includes a separate hitlist for suppliers
# followed by a union of the hitlists
#
# perl merlin client for the design of combinatorial libraries
#
# 12/96 ags
#
# Copyright 1999 A.G. Skillman, I.D. Kuntz, Regents of the University of California
#
# This is a prototype program to demonstrate reagent searching using a
# cgi script to parse common chemical nomenclature into daylight
# search commands. ALTHOUGH COMMON NOMENCLATURE IS A VERY USEFUL WAY TO
# COMMUNICATE CHEMICAL INFORMATION, IT IS NEITHER ENTIRELY SYSTEMATIC
NOR
# UNIQUE. Subtle problems with the ambiguities of common nomenclature can be
# found in the definitions used here. However, we feel that the utility of
# common nomenclature in the majority of cases justifies its use.

# Programming Notes
# 0. This is basically a C program written in Perl to take advantage of
#the great string functions
# 1. Most of the variable are held in a $global hash. This is both
#convienient and burdensome.
# 2. This program has overgrown its initial design and little foresight
#was used in limiting the scope of variables (sorry)
# 3. The $url_path variable defines the directory where reagent and
#order can be executed
# 4. All of the complex parsing unique to this program can be found in
#the subroutine GETINPUT. Most of the other routines are generic
#Merlin-client routines or CGI i/o routines
# 5. This program runs under Perl 5.003 with DayPerl installed
# 6. The graphical output is dependent on having the standard "daycgi"
#directory in the server's path
# 7. The functional groups in the form and those in the vbindings are not
#      linked in a direct way. This is unfortunate because these will
#      probably be changed often, and are critically dependent on one
#      another
# 8. vbindings don't work with merlin, so they are implemented here with
#      string manipulations (see readme daylight for more info).
# 9. in PRINTFORM there are some widths of 2000%. it is not clear why
#      this number needs to be so hight, but it is necessary for the
#      browsers i've tried. it would be reasonable if this caused problems
#      for some other systems, but can be adjusted easily
#10. Thanks to Jeremy Yang for getting me started with Merlin clients &
#      Daylight for use of their software at UCSF
#
# 6/98 ags.

use DayPerl;

$| = 1,
$TRUE = 1,
$FALSE = 0;
$url_path = "http://yorick.ucsf.edu:8000/cgi-bin",

##########################main program control##########################

###if no input send form###

if($ENV{'CONTENT_LENGTH'} eq ""){
  &PRINTFORM,
  exit 1,
}

###set things up & read parameters   ###

&INITIALIZE,

&GETINPUT,

###if simple save hitlist, do it and quit###

if($global{"save_hit"}){
  &SAVEHITLIST(@molecules);
  exit 1,
}

###primary searching sequence  ###

&CHECKINPUT($global{"essential_smiles"}, $global{"essential_smarts"},
  $global{"bad_double_smarts"}).

&OPENDATABASE;

###fill the hitlist either from file or using suppliers###

&SETHITLIST(@molecules);

&SUPPLIERS($global{"supplier_type"}),

###do fast searches first sorted > name > fcd###

&SORTEDCULLS;

&NAMESEARCH($global{"essential_name"});

&FCDSEARCH($global{"fcd_number"});

###do slow smiles & smarts searches   ###

&PRIMARYCHEMICAL($global{"essential_smiles"}, $global{"essential_smarts"},
  $global{"essential_smarts_button1"}, $global{"smiles1"},
  $global{"essential_smarts_button2"}, $global{"smiles2"},
  $global{"essential_smarts_button3"}, $global{"smiles3"});

###more slow smarts searches  ###

&REMOVEGOOD;

&REMOVEBAD;

###output to user's browser   ###

&WRITEHITLIST($global{"output_type"});

quit:
  dt_dealloc($pool),
  DBLPRINT ("\n\n***\tFinished Library Design 0.0\t***\n\n");
  printf ("</BODY></HTML>"),
  exit 0;

############################end of main########################################

##########################
#set hitlist with molecules read from file
##########################
sub SETHITLIST {
  local(@molecules) = @_;

  $max_hit = dt_mer_clear($hitlist);

  $seq = dt_alloc_seq();
  foreach $molecule (@molecules){
$molstring = dt_alloc_string($molecule);
dt_append($seq, $molstring);
  }

  $max_hit = dt_mer_sethits($hitlist, $col_smi, $seq);

  dt_reset($seq);
  while(NULL_OB != ($molstring = dt_next($seq))){
dt_dealloc($molstring),
  }
  dt_dealloc($seq);

  DBLPRINT("\nSet $max_hit molecules to hitlist from file.\n\n");

}

##########################
#write hitlist to browser - SMILES 1per line format
##########################
sub SAVEHITLIST {
  local(@molecules) = @_;

  printf ("Content-type: application/octet-stream\n\n");
  foreach $molecule (@molecules){
printf("$molecule\n"),
  }

}

##########################
#add cpds from supplier. if first make new hitlist
##########################
sub ADDSUPPLIER {
  local($list, $name, $action) = @_;

  DBLPRINT ("Doing string search for supplier: $name.\n");

  $ndone = dt_mer_strsearch($list, $col_sup, DX_STRING_ASCII,
$action, -1, $status, $name, $name);
  &PROGRESS($mserver) if ($status == DX_STATUS_IN_PROGRESS);
  if($status == DX_STATUS_NOT_FOUND){
DBLPRINT ("Target not found.\n");
  }
  elsif ($status == DX_STATUS_ERROR){
DBLPRINT ("Error in string search.\n");
&DU_PRINTERRORS,
  }
  $max_hit = dt_mer_length($list);
  DBLPRINT ("Finished Supplier search.  hitlist = $max_hit.\n\n"),
}

##########################
#select fcd's based on substring
##########################
sub FCDSEARCH {
```

```
    local($fcd) = @_;

    if($fcd){
    DBLPRINT ("Doing string search for fcd:  $fcd.\n");

    $ndone = dt_mer_strsearch($hitlist, $col_fcd, DX_STRING_APPROX,
    DX_ACTION_DEL_NONHITS, -1, $status, $fcd, "McS");
    &PROGRESS($mserver) if ($status == DX_STATUS_IN_PROGRESS);
    if($status == DX_STATUS_NOT_FOUND){
    DBLPRINT ("Target not found.\n"),
    }
    elsif ($status == DX_STATUS_ERROR){
    DBLPRINT ("Error in string search \n");
    &DU_PRINTERRORS;
    }
    $max_hit = dt_mer_length($hitlist);
    DBLPRINT ("Finished Name Search.  hitlist = $max_hit.\n\n");
    }
}

##########################
#use name to limit search
##########################
sub NAMESEARCH {
    local($string) = @_;

    if($string){
    DBLPRINT ("Doing string search for name:  $string \n");

    $ndone = dt_mer_strsearch($hitlist, $col_nam, DX_STRING_APPROX,
    DX_ACTION_DEL_NONHITS, -1, $status, $string, "McS");
    &PROGRESS($mserver) if ($status == DX_STATUS_IN_PROGRESS);
    if($status == DX_STATUS_NOT_FOUND){
    DBLPRINT ("Target not found.\n"),
    }
    elsif ($status == DX_STATUS_ERROR){
    DBLPRINT ("Error in string search.\n"),
    &DU_PRINTERRORS;
    }
    $max_hit = dt_mer_length($hitlist);
    DBLPRINT ("Finished Name Search.  hitlist = $max_hit.\n\n");
    }
}

##########################
#use only selected suppliers
##########################
sub SUPPLIERS {
    local($type) = @_.

    $max_hit = dt_mer_length($hitlist);

    if($type eq "all_suppliers"){
if($max_hit == 0){
    $max_hit = dt_mer_reset($hitlist);
    DBLPRINT("All Suppliers. hitlist = $max_hit\n\n");
}
#default do-nothing
    }

###no hitlist set###

    elsif(($type eq "suppliers_here")&&($max_hit == 0)){
$action = DX_ACTION_NEW_LIST;
foreach $supplier (keys %goodsupplier){
    ADDSUPPLIER($hitlist,$goodsupplier{$supplier},$action);
    $action = DX_ACTION_ADD_HITS;
}
    }
    elsif(($type eq "selected_suppliers")&&($max_hit == 0)){
$action = DX_ACTION_NEW_LIST;
foreach $supplier (keys %goodsupplier){
    if($global{"$supplier"}){
ADDSUPPLIER($hitlist,$goodsupplier{$supplier},$action);
$action = DX_ACTION_ADD_HITS,
    }
}
    }

###with a hitlist already set ###

    elsif($type eq "suppliers_here"){
DBLPRINT("Starting independent supplier search.\n\n");
$newlist = dt_mer_alloc_hitlist($pool);
$action = DX_ACTION_NEW_LIST;
foreach $supplier (keys %goodsupplier){
    ADDSUPPLIER($newlist,$goodsupplier{$supplier},$action);
    $action = DX_ACTION_ADD_HITS;
}
DBLPRINT("Finished all suppliers.\n");
DBLPRINT("Taking union of suppliers and original hitlist.\n");
$action = DX_ACTION_DEL_NONHITS,
$max_hit = dt_mer_combinehitlists($hitlist,$newlist,$action),
DBLPRINT("Finished Union. hitlist = $max_hit.\n\n");
    }
    elsif($type eq "selected_suppliers"){
DBLPRINT("Starting independent supplier search.\n\n");
$newlist = dt_mer_alloc_hitlist($pool);
$action = DX_ACTION_NEW_LIST;
foreach $supplier (keys %goodsupplier){
    if($global{"$supplier"}){
ADDSUPPLIER($newlist,$goodsupplier{$supplier},$action);
    $action = DX_ACTION_ADD_HITS;
    }
}
DBLPRINT("Finished all suppliers.\n");
DBLPRINT("Taking union of suppliers and original hitlist.\n");
$action = DX_ACTION_DEL_NONHITS;
```

```
$max_hit = dt_mer_combinehitlists($hitlist,$newlist,$action);
DBLPRINT("Finished Union. hitlist = $max_hit.\n\n");
    }
}

##########################
#print to output and to log
##########################
sub DBLPRINT {
    local($string) = @_;

    printf LOG "$string";
    $string =~ s/\</"&LT;"/eg;
    $string =~ s/\>/"&GT;"/eg;
    $string =~ s/\n/"<BR>\n"/eg;
    printf STDOUT "$string";
}
##########################
#save hitlist
##########################
sub WRITEHITLIST {
    local($format) = @_;

    DBLPRINT ("\nHistlist:\n\n");

    if($format eq "lst"){
printf STDOUT ("*E*<BR><BR>\n");
for($i=0;$i<$max_hit;$i++){
    $smiles = dt_mer_cellvalue($col_smi, $hitlist, $i);
    $picsmiles = join(" ", $smiles, "#w=300 #h=150 #c=20");
    $hexpic = unpack("H*", $picsmiles);
    $fcd = dt_mer_cellvalue($col_fcd, $hitlist, $i);
    printf STDOUT ("<A HREF=\"/daycgi/smi2gif?$hexpic\">\n");
    printf STDOUT ("$fcd</A><BR>\n");
}
    }

    if($format eq "library"){
printf STDOUT ("<TABLE BORDER=0 WIDTH=80%%>");
printf STDOUT ("<TR><TD>Library:</TD><TD>$global{\"reaction\"}</
TD><TD><I>library_reaction</I></TD></TR>\n");
printf STDOUT ("</TABLE>\n");
printf STDOUT ("<TABLE BORDER=0 WIDTH=80%%>");
for($i=0;$i<$max_hit;$i++){
    $smiles = dt_mer_cellvalue($col_smi, $hitlist, $i);
    $fcd = dt_mer_cellvalue($col_fcd, $hitlist, $i);
    $picsmiles = join(" ", $smiles, "#w=300 #h=150 #c=20");
    $hexpic = unpack("H*", $picsmiles);
    $hexsmiles = unpack("H*", $smiles);
    printf STDOUT ("<TR><TD><A HREF=\"/daycgi/smi2gif?$hexpic\">\n");
    printf STDOUT ("$smiles</A></TD>\n");
    printf STDOUT ("<TD ALIGN=right><A HREF=\"$url_path/order?$hexsmiles\">\n");
    printf STDOUT ("$fcd</A></TD></TR>\n");
}
printf STDOUT ("</TABLE>");
    }

    if($format eq "graphic"){
if($max_hit > 2500){
    DBLPRINT ("Hitlist too long for graphical output...please limit search or complain.\n");
    $format = "tdt";
}
else{
    printf STDOUT ("<TABLE BORDER=5 CELLSPACING=8><TR><TH> </TH><TH> </TH></
TR>\n");
    for($i=0;$i<$max_hit;$i+=2){
$smiles = "";
$smiles1 = "";
$smiles = dt_mer_cellvalue($col_smi, $hitlist, $i);
$smiles1 = dt_mer_cellvalue($col_smi, $hitlist, $i+1);
$fcd = dt_mer_cellvalue($col_fcd, $hitlist, $i);
$fcd1 = dt_mer_cellvalue($col_fcd, $hitlist, $i+1);
$picsmiles = join(" ", $smiles, "#w=300 #h=150 #c=20");
$hexpic = unpack("H*", $picsmiles);
$picsmiles1 = join(" ", $smiles1, "#w=300 #h=150 #c=20");
$hexpic1 = unpack("H*", $picsmiles1);
$hexsmiles = unpack("H*", $smiles);
$hexsmiles1 = unpack("H*", $smiles1);
printf STDOUT ("<TR><TD><A HREF=\"$url_path/order?$hexsmiles\">\n");
printf STDOUT ("$fcd</TD>");
printf STDOUT ("<TD><A HREF=\"$url_path/order?$hexsmiles1\">\n");
printf STDOUT ("$fcd1</TD></TR>\n");
printf STDOUT ("<TR>");
printf STDOUT ("<TD><IMG SRC=\"/daycgi/smi2gif?$hexpic\" WIDTH=300 HEIGHT=150></
TD>\n");
printf STDOUT ("<TD><IMG SRC=\"/daycgi/smi2gif?$hexpic1\" WIDTH=300 HEIGHT=150></
TD>\n");
printf STDOUT ("</TR>");
    }
    printf STDOUT ("</TABLE><BR>\n");
}
    }

    if($format eq "tdt"){
for($i=0;$i<$max_hit;$i++){
    $smiles = dt_mer_cellvalue($col_smi, $hitlist, $i);
    $name = dt_mer_cellvalue($col_nam, $hitlist, $i);
    $mw = dt_mer_cellvalue($col_mw, $hitlist, $i),
    $fcd = dt_mer_cellvalue($col_fcd, $hitlist, $i);
    $picsmiles = join(" ", $smiles, "#w=300 #h=150 #c=20");
    $hexsmiles = unpack("H*", $smiles);
    $hexpic = unpack("H*", $picsmiles);
    printf STDOUT ("<A HREF=\"/daycgi/smi2gif?$hexpic\">\n");
    printf STDOUT ("\$SMI&lt;$smiles&gt;</A><BR>\n");
    printf STDOUT ("\$NAM&lt;$name&gt;<BR>\n");
    printf STDOUT ("<A HREF=\"$url_path/order?$hexsmiles\">\n");
    printf STDOUT ("\$FCD&lt;$fcd&gt;</A><BR>\n");
    printf STDOUT ("AMW&lt;$mw&gt;<BR>\n<BR>\n");
```

```
}
   }

###write hidden hitlist data to give ability to save hitlist   ###

   if($TRUE){

printf ("<HR>\n");
printf ("<FORM METHOD = \"POST\" ACTION=\"$url_path/reagent\">\n");
printf ("\n");
printf ("<INPUT type=hidden name=save_hit value=1>\n");
for($i=0;$i<$max_hit;$i++){
   $smiles = dt_mer_cellvalue($col_smi, $hitlist, $i);
   printf ("<INPUT TYPE=hidden name=HIT$i value=\"$smiles\">\n");
}
printf ("<INPUT TYPE=\"submit\" value=\"Save Hitlist\">\n");
printf ("</FORM>\n");
printf ("<HR>\n"),

   }

}

#########################
#remove cpds with smarts passed as arguent
#########################
sub REMOVE {
   local($smarts) = @_;

   &CHECKSMARTS($smarts);
   if($smarts){
      DBLPRINT ("Doing SMARTS search for $smarts.\n");
      $ndone = dt_mer_superselect($hitlist, $col_smi, DX_SUPER_SMARTS,
         DX_ACTION_DEL_HITS, -1, $status, $smarts);
      &PROGRESS($mserver) if ($status == DX_STATUS_IN_PROGRESS);
      if($status == DX_STATUS_NOT_FOUND){
         DBLPRINT ("Target not found.\n");
      }
      elsif ($status == DX_STATUS_ERROR){
         DBLPRINT ("Error in smarts search.\n");
         &DU_PRINTERRORS,
      }
      $max_hit = dt_mer_length($hitlist);
      DBLPRINT ("Finished SMARTS search.  hitlist = $max_hit.\n\n"),
   }
}

#########################
#remove functional groups whose default is to keep if specified
#########################
sub REMOVEGOOD {

   for($i=0;$i<$max_good;$i++){
      if(!$global{$good[$i]}){
         DBLPRINT ("Removing compounds with $good[$i] groups.\n");
         &REMOVE($vb{$good[$i]});
      }
   }

}

#########################
#remove functional groups whose default if remove unless specifically kept.
#########################
sub REMOVEBAD {

   for($i=0;$i<$max_bad;$i++){
      if(!$global{$bad[$i]}){
         DBLPRINT ("Removing compounds with $bad[$i] groups.\n");
         &REMOVE($vb{$bad[$i]}),
      }
   }

}

#########################
#check input - make sure smarts and smiles are ok
#########################
sub CHECKINPUT {
   local($smiles, $smarts, $dblsmarts) = @_;

   &CHECKSMILES($smiles);

   &CHECKSMARTS($smarts);

   &CHECKSMARTS($dblsmarts);

}

#########################
#check smiles
#########################
sub CHECKSMILES{
   local($smiles) = @_;
   if($smiles){
if(NULL_OB == ($mol = dt_smilin($smiles))){
   &DU_PRINTERRORS;
   DBLPRINT ("Trouble making smiles = \"$smiles\" into a molecule.\n");
   goto quit;
}
   }

   dt_dealloc($mol);
}
#########################
#check smarts
#########################
sub CHECKSMARTS{
   local($smarts) = @_;
   if($smarts){
if(NULL_OB == ($pattern = dt_smartin($smarts))){
   &DU_PRINTERRORS;
   DBLPRINT ("Trouble making smarts = \"$smarts\" into a search pattern.\n");
   goto quit;
}
   }

   dt_dealloc($pattern);
}

#########################
#primary chemical - get the functional group you want
#########################
sub PRIMARYCHEMICAL {
   local($smiles,$smarts,$button1,$smiles1,$button2,$smiles2,$button3,$smiles3) = @_;

   if($smiles){

REMOVESMILES($smiles);

   }


   if($smarts){

#$opt_smarts = dt_smarts_opt($smarts,FALSE);

REMOVESMARTS($smarts);

   }


   if($button1){

if($smiles1){
   REMOVESMILES($smiles1),
}

REMOVESMARTS($button1);

if($global{"double1"}){
   REMOVEDBL($button1,$global{"double1"});
}
   }

   if($button2){

if($smiles2){
   REMOVESMILES($smiles2);
}

REMOVESMARTS($button2);

if($global{"double2"}){
   REMOVEDBL($button2,$global{"double2"});
}
   }

   if($button3){

if($smiles3){
   REMOVESMILES($smiles3);
}

REMOVESMARTS($button3);

if($global{"double3"}){
   REMOVEDBL($button3,$global{"double3"});
}
   }
}

#########################
#remove smiles
#########################
sub REMOVESMILES {
   local($smiles) = @_;

   DBLPRINT ("Doing SMILES search for $smiles.\n");

   $ndone = dt_mer_superselect($hitlist, $col_smi, DX_SUPER_SMILES,
DX_ACTION_DEL_NONHITS, -1, $status, $smiles);
   &PROGRESS($mserver) if ($status == DX_STATUS_IN_PROGRESS);
   if($status == DX_STATUS_NOT_FOUND){
DBLPRINT ("Target not found.\n");
goto quit;
   }
   elsif ($status == DX_STATUS_ERROR){
DBLPRINT ("Error in smiles search.\n");
&DU_PRINTERRORS,
goto quit;
   }
   $max_hit = dt_mer_length($hitlist);
   DBLPRINT ("Finished SMILES search.  hitlist = $max_hit.\n\n");
}

#########################
#remove smarts
#########################
sub REMOVESMARTS {
   local($smarts) = @_;

   DBLPRINT ("Doing SMARTS search for $smarts.\n");

   $ndone = dt_mer_superselect($hitlist, $col_smi, DX_SUPER_SMARTS,
DX_ACTION_DEL_NONHITS, -1, $status, $smarts);
```

```
   &PROGRESS($mserver) if ($status == DX_STATUS_IN_PROGRESS);
   if($status == DX_STATUS_NOT_FOUND){
DBLPRINT ("Target not found.\n");
goto quit;
   }
   elsif ($status == DX_STATUS_ERROR){
DBLPRINT ("Error in smarts search.\n");
&DU_PRINTERRORS;
goto quit;
   }
   $max_hit = dt_mer_length($hitlist);
   DBLPRINT ("Finished SMARTS search.  hitlist = $max_hit.\n\n");
}

########################
#remove double - remove double occurances of smarts
########################
sub REMOVEDBL {
   local($halfsmarts, $symmetry) = @_;

   DBLPRINT ("Removing double occurances of $halfsmarts.\n");

   $dblsmarts = join(".", $halfsmarts, $halfsmarts);
   for($i=1;$i<$symmetry;$i++){
$dblsmarts = join(".", $dblsmarts, $halfsmarts);
   }

   $ndone = dt_mer_superselect($hitlist, $col_smi, DX_SUPER_SMARTS,
DX_ACTION_DEL_HITS, -1, $status, $dblsmarts);
   &PROGRESS($mserver) if ($status == DX_STATUS_IN_PROGRESS);
   if($status == DX_STATUS_NOT_FOUND){
DBLPRINT ("Target not found.\n");
   }
   elsif ($status == DX_STATUS_ERROR){
DBLPRINT ("Error in smarts search.\n");
&DU_PRINTERRORS;
goto quit;
   }
   $max_hit = dt_mer_length($hitlist);
   DBLPRINT ("Finished SMARTS search.  hitlist = $max_hit.\n\n");
}

########################
#sorted culls - remove by mw, price, price/gm
########################
sub SORTEDCULLS {

   ########################
   #low and high mw
   ########################

   $ftype_mw = &GET_FTYPE($pool, "AMW");
   $col_mw = dt_mer_alloc_column($pool, $ftype_mw, DX_FUNC_FIRST);
   DBLPRINT ("Doing molecular weight search.\n");

   $ndone = dt_mer_numsearch($hitlist, $col_mw, DX_ACTION_DEL_NONHITS,
-1, $status, $global{"mw_min"}, $global{"mw_max"});
   &PROGRESS($mserver) if ($status == DX_STATUS_IN_PROGRESS);
   if($status == DX_STATUS_NOT_FOUND){
DBLPRINT ("Target not found.\n");
   }
   elsif ($status == DX_STATUS_ERROR){
DBLPRINT ("Error in mw search.\n");
&DU_PRINTERRORS;
goto quit;
   }
   $max_hit = dt_mer_length($hitlist);

   DBLPRINT ("Removed cpds with mw less than $global{\"mw_min\"}.\n");
   DBLPRINT ("Removed cpds with mw more than $global{\"mw_max\"}.  hitlist = $max_hit.\n\n");

   dt_dealloc($ftype_mw);

   ########################
   #low and high clogp
   ########################

   $ftype_logp = &GET_FTYPE($pool, "CP");
   $col_logp = dt_mer_alloc_column($pool, $ftype_logp, DX_FUNC_FIRST);
   DBLPRINT ("Doing clogP search.\n");

   $ndone = dt_mer_numsearch($hitlist, $col_logp, DX_ACTION_DEL_NONHITS,
-1, $status, $global{"logp_min"}, $global{"logp_max"});
   &PROGRESS($mserver) if ($status == DX_STATUS_IN_PROGRESS);
   if($status == DX_STATUS_NOT_FOUND){
DBLPRINT ("Target not found.\n");
   }
   elsif ($status == DX_STATUS_ERROR){
DBLPRINT ("Error in clogP search.\n");
&DU_PRINTERRORS;
goto quit;
   }
   $max_hit = dt_mer_length($hitlist);

   DBLPRINT ("Removed cpds with ClogP less than $global{\"logp_min\"}.\n");
   DBLPRINT ("Removed cpds with ClogP more than $global{\"logp_max\"}.  hitlist =
$max_hit.\n\n");

   dt_dealloc($ftype_logp);

   ########################
   #logp error
   ########################

   $ftype_logp_err = &GET_FTYPEN($pool, "CP", 2);
   $col_logp_err = dt_mer_alloc_column($pool, $ftype_logp_err, DX_FUNC_FIRST);
   DBLPRINT ("Doing clogP error search.\n");

   ($err,$dum) = split(":", $global{"logp_err"});
   $ndone = dt_mer_strsearch($hitlist, $col_logp_err, DX_STRING_ASCII,
DX_ACTION_DEL_NONHITS, -1, $status, "", $err);
   &PROGRESS($mserver) if ($status == DX_STATUS_IN_PROGRESS);
   if($status == DX_STATUS_NOT_FOUND){
DBLPRINT ("Target not found.\n");
   }
   elsif ($status == DX_STATUS_ERROR){
DBLPRINT ("Error in clogP error search.\n");
&DU_PRINTERRORS;
goto quit;
   }
   $max_hit = dt_mer_length($hitlist);

   DBLPRINT ("Removed cpds with ClogP error more than \"$global{\"logp_err\"}.\"\n  hitlist =
$max_hit.\n\n");

   dt_dealloc($ftype_logp_err);

   ########################
   #low and high h-bond donors
   ########################

   $ftype_hbd = &GET_FTYPEN($pool, "COUNTS", 1);
   $col_hbd = dt_mer_alloc_column($pool, $ftype_hbd, DX_FUNC_FIRST);
   DBLPRINT ("Doing H-bond Donor search.\n");

   $ndone = dt_mer_numsearch($hitlist, $col_hbd, DX_ACTION_DEL_NONHITS,
-1, $status, $global{"hbd_min"}, $global{"hbd_max"});
   &PROGRESS($mserver) if ($status == DX_STATUS_IN_PROGRESS);
   if($status == DX_STATUS_NOT_FOUND){
DBLPRINT ("Target not found.\n");
   }
   elsif ($status == DX_STATUS_ERROR){
DBLPRINT ("Error in H-bond Donor search.\n");
&DU_PRINTERRORS;
goto quit;
   }
   $max_hit = dt_mer_length($hitlist);

   DBLPRINT ("Removed cpds with hbd count less than $global{\"hbd_min\"}.\n");
   DBLPRINT ("Removed cpds with hbd count more than $global{\"hbd_max\"}.  hitlist =
$max_hit.\n\n");

   dt_dealloc($ftype_hbd);

   ########################
   #low and high h-bond acceptors
   ########################

   $ftype_hba = &GET_FTYPEN($pool, "COUNTS", 2);
   $col_hba = dt_mer_alloc_column($pool, $ftype_hba, DX_FUNC_FIRST);
   DBLPRINT ("Doing H-bond Acceptor search.\n");

   $ndone = dt_mer_numsearch($hitlist, $col_hba, DX_ACTION_DEL_NONHITS,
-1, $status, $global{"hba_min"}, $global{"hba_max"});
   &PROGRESS($mserver) if ($status == DX_STATUS_IN_PROGRESS);
   if($status == DX_STATUS_NOT_FOUND){
DBLPRINT ("Target not found.\n");
   }
   elsif ($status == DX_STATUS_ERROR){
DBLPRINT ("Error in H-bond Acceptor search.\n");
&DU_PRINTERRORS;
goto quit;
   }
   $max_hit = dt_mer_length($hitlist);

   DBLPRINT ("Removed cpds with hba count less than $global{\"hba_min\"}.\n");
   DBLPRINT ("Removed cpds with hba count more than $global{\"hba_max\"}.  hitlist =
$max_hit.\n\n");

   dt_dealloc($ftype_hba);

   ########################
   #low and high rotatable bonds
   ########################

   $ftype_roth = &GET_FTYPEN($pool, "COUNTS", 3);
   $col_roth = dt_mer_alloc_column($pool, $ftype_roth, DX_FUNC_FIRST);
   DBLPRINT ("Doing rotatable bond search.\n");

   $ndone = dt_mer_numsearch($hitlist, $col_roth, DX_ACTION_DEL_NONHITS,
-1, $status, $global{"roth_min"}, $global{"roth_max"});
   &PROGRESS($mserver) if ($status == DX_STATUS_IN_PROGRESS);
   if($status == DX_STATUS_NOT_FOUND){
DBLPRINT ("Target not found.\n");
   }
   elsif ($status == DX_STATUS_ERROR){
DBLPRINT ("Error in rotatable bond search.\n");
&DU_PRINTERRORS;
goto quit;
   }
   $max_hit = dt_mer_length($hitlist);

   DBLPRINT ("Removed cpds with rotatable bond count less than $global{\"roth_min\"}.\n");
   DBLPRINT ("Removed cpds with rotatable bond count more than $global{\"roth_max\"}. hitlist =
$max_hit.\n\n");

   dt_dealloc($ftype_roth);

   ########################
   #low and high formal charge
   ########################

   $ftype_crg = &GET_FTYPEN($pool, "COUNTS", 4);
   $col_crg = dt_mer_alloc_column($pool, $ftype_crg, DX_FUNC_FIRST);
   DBLPRINT ("Doing formal charge search.\n");
```

```
   $ndone = dt_mer_numsearch($hitlist, $col_crg, DX_ACTION_DEL_NONHITS,
-1, $status, $global{"crg_min"}, $global{"crg_max"});
   &PROGRESS($mserver) if ($status == DX_STATUS_IN_PROGRESS);
   if($status == DX_STATUS_NOT_FOUND){
DBLPRINT ("Target not found.\n");
   }
   elsif ($status == DX_STATUS_ERROR){
DBLPRINT ("Error in formal charge search \n");
&DU_PRINTERRORS;
goto quit,
   }
   $max_hit = dt_mer_length($hitlist);

   DBLPRINT ("Removed cpds with formal crg count less than $global{\"crg_min\"} \n");
   DBLPRINT ("Removed cpds with formal crg count more than $global{\"crg_max\"}. hitlist =
$max_hit \n\n");

   dt_dealloc($ftype_crg);

   ########################
   #high price
   ########################

   $ftype_usdpergm = &GET_FTYPEN($pool, "SUP", 7);
   $col_usdpergm = dt_mer_alloc_column($pool, $ftype_usdpergm, DX_FUNC_MIN);

   DBLPRINT ("Doing price per gram search.\n");

   $ndone = dt_mer_numsearch($hitlist, $col_usdpergm, DX_ACTION_DEL_NONHITS,
-1, $status, 0, $global{"usdpergm"}),
   &PROGRESS($mserver) if ($status == DX_STATUS_IN_PROGRESS);
   if($status == DX_STATUS_NOT_FOUND){
DBLPRINT ("Target not found.\n"),
   }
   elsif ($status == DX_STATUS_ERROR){
DBLPRINT ("Error in price search.\n"),
&DU_PRINTERRORS;
goto quit;
   }
   $max_hit = dt_mer_length($hitlist);

   DBLPRINT ("Removed cpds with price per gram more than \$US $global{\"usdpergm\"}. hitlist =
$max_hit.\n\n");

}

#########################
#progress of a merlin process
#########################
sub PROGRESS {

   local($server) = @_,

   $old_percent_done = 0;
   $done_when = dt_done_when($server);
   while(DX_STATUS_IN_PROGRESS == $status){
$percent_done = int((100 * $ndone)/$done_when),
if ($percent_done != $old_percent_done) {printf("%d%% ", $percent_done);printf LOG ("%d%% ",
$percent_done), }
$old_percent_done = $percent_done;
$ndone = dt_continue($server, $status),
   }
   DBLPRINT ("done.\n"),
}

#########################
#get du_printerrors after jj wang
#########################
sub DU_PRINTERRORS {
   $errs = dt_errors(DX_ERR_ERROR),
   while (NULL_OB != ($err = dt_next($errs))){
$errstring = dt_stringvalue($err);
DBLPRINT ("$errstring\n"),
   }
}

#########################
#get ftype($pool, $tag)
#########################
sub GET_FTYPE {
   local($pool, $tag) = @_;

   $dtype = dt_getdatatype($pool, $tag);
if(NULL_OB == $dtype){printf("no dtype in getftype\n")};
   $ftypes = dt_stream($dtype, TYP_FIELDTYPE);
   if (NULL_OB == ($ftype = dt_next($ftypes))){
DBLPRINT ("Can't get 1st field for \"$tag\" datatype.\n");
return(NULL_OB),
   }
   dt_dealloc($ftypes),

   return($ftype);
}

#########################
#get ftype($pool, $tag, $count)
#########################
sub GET_FTYPEN {
   local($pool, $tag, $count) = @_;

   $dtype = dt_getdatatype($pool, $tag);
   $ftypes = dt_stream($dtype, TYP_FIELDTYPE);
   for($i=0,$i<$count;$i++){
if (NULL_OB == ($ftype = dt_next($ftypes))){
   DBLPRINT ("Can't get field $count for \"$tag\" datatype.\n");
   return(NULL_OB);
   }
   }
```

```
   dt_dealloc($ftypes);

   return($ftype);
}

#########################
#open database
#########################
sub OPENDATABASE {
   local($i) = (0);

   chop($whoami = 'whoami');
#   $host = $hostname || (gethostent())[0];
   $user = $ENV{'USER'} || getlogin || (getpwuid($<))[0] || $whoami;
   $host = $global{"server"};
   $service = "merlin",
   $userpw = $dbpw = "";

   ########################
   #connect to server after example code by jj yang
   ########################

   $poolname = "$global{'pool'}\@$host:$service:$user";
   if (NULL_OB == ($mserver = dt_mer_server($host, $service, $user, $userpw, $isnew))){
DBLPRINT ("Can't connect to server: \"$poolname\"\n");
goto quit;
   }
   elsif (!dt_isopen($mserver, $global{"pool"})){
DBLPRINT ("Pool not loaded: $global{\"pool\"}\n");
goto quit;
   }
   elsif (NULL_OB == ($pool = dt_open($mserver,  $global{"pool"}, "r", $dbpw,  $isnew))){
DBLPRINT ("Can't open pool: $global{\"pool\"}\n");
goto quit;
   }
   else {
DBLPRINT ("Pool opened : $global{\"pool\"}\n");
   }

   ########################
   #allocate columns and hitlist
   ########################

   $ftype_sup = &GET_FTYPEN($pool, "SUP", 1);
   $col_sup = dt_mer_alloc_column($pool, $ftype_sup, DX_FUNC_ALL);
   $ftype_nam = &GET_FTYPE($pool, "\$NAM");
   $col_nam = dt_mer_alloc_column($pool, $ftype_nam, DX_FUNC_ALL);
   $ftype_smi = &GET_FTYPE($pool, "\$SMI");
   $col_smi = dt_mer_alloc_column($pool, $ftype_smi, DX_FUNC_FIRST);
   $ftype_fcd = &GET_FTYPE($pool, "\$FCD");
   $col_fcd = dt_mer_alloc_column($pool, $ftype_fcd, DX_FUNC_FIRST);
   if((NULL_OB == $ftype_smi) || (NULL_OB == $ftype_sup) || (NULL_OB == $ftype_fcd)){
dt_dealloc($mserver);
goto quit,
   }
   $hitlist = dt_mer_alloc_hitlist($pool);

   return(1);
}

#########################
#get attachment string
#########################
sub GET_ATTACHMENT {
   local($mod, $near, $ct) = @_;
   local($tempmod, $firsttime);

   $any = ($new_variables{$buttonnear} eq "any_attachment");
   @linkers = ("","*","**","***","****","*****","******");
   $maxlink = @linkers;
   $modifystring = "";
   $tempmod = "";
   $firsttime = $TRUE;

   if(!$any){
$tempmod = join("", $near, $vb{$mod});
for($i=1;$i < $ct;$i++){
   $modifystring = join("", $modifystring, "(", $tempmod, ")");
}
$modifystring = join("", "(*", $modifystring, $tempmod, ")]");
   }

   else{
if($ct == 1){
   foreach $link (@linkers){
if($firsttime){
   $firsttime = $FALSE;
   $tempmod = join("", "*", $link, $vb{$mod}, ")");
}
else{
   $tempmod = join("", $tempmod, ",\$(*", $link, $vb{$mod},")");
}
   }
   $tempmod = join("", $tempmod, "");

}
elsif($ct == 2){
   for($link1=$maxlink-1,$link1>=0;$link1--){
for($link2=$link1;$link2>=0;$link2--){
   if($firsttime){
$firsttime = $FALSE;
$tempmod = join("", "*(",
$linkers[$link1], $vb{$mod}, ")",
$linkers[$link2], $vb{$mod}, ")");
   }
   else{
$tempmod = join("", $tempmod, ",\$(*(",
$linkers[$link1], $vb{$mod}, ")",
```

```
$linkers[$link2], $vb{$mod}, ")");
    }
}
    }
}
$modifystring = join("", "(", $modifystring, $tempmod, "]");
    }

   return($modifystring);
}
##########################
#get input
##########################
sub GETINPUT {
   local($filename) = @_;
   @keys = keys %global;

   %new_variables = READPOST(%new_variables);

### put hitlist into molecule ###

   foreach $variable (keys %new_variables){
if(substr($variable,0,3) eq 'HIT'){
   $molecules[substr($variable,3)] = $new_variables{$variable};
}
    }

### fix input for easy boolean,  acknowledge input ###

   foreach $variable (keys %new_variables){
if($new_variables{$variable} eq "FALSE"){
   $new_variables{$variable} = 0;
}
if($new_variables{$variable} eq "TRUE"){
   $new_variables{$variable} = 1,
}
if($variable != grep(/$variable/, @keys)){
#   DBLPRINT ("$variable = $global{$variable} was reset, its new value is $new_variables{$vari-
able} \n");
   $global{$variable} = $new_variables{$variable};
}
    }

###parse primary smarts first because other variables are dependent    ###

   foreach $variable (keys %new_variables){

###parse button names to button smarts and set smiles    ###

if(substr($variable, 0, 23) eq "essential_smarts_button"){
#   DBLPRINT ("$variable = $global{$variable} was reset,  its new value is $vb{$global{$vari-
able}} \n");
   $global{$variable} = $vb{$global{$variable}},
   $buttonsmi = join("","smiles",substr($variable,23));
#   DBLPRINT ("$buttonsmi was set to $sb{$new_variables{$variable}} \n"),
   $global{$buttonsmi} = $sb{$new_variables{$variable}},
}
    }

###now parse the rest of the variables###

   foreach $variable (keys %new_variables){

###append secondary stuff to primary button smarts and smiles  ###

if((substr($variable, 0, 6) eq "modify")&&($new_variables{$variable} ne "none")){

# get buttons

   $buttonsma = join("", "essential_smarts_button", substr($variable,6,1)),
   $buttonsmi = join("","smiles",substr($variable,6,1));

   $buttonmod = $variable;
   $buttonext = join("", "extra", substr($variable,6));
   $buttonnear = join("", "near", substr($variable,6)),
   $buttonnearct = join("", "nearct", substr($variable,6)),
printf ("<BR>***b-$buttonnearct*n-$new_variables{$buttonnearct}***\n<BR>"),
$new_variables{$buttonnearct}=1,
# make modifier smiles string

   if($new_variables{$buttonext}){
$global{$buttonext} = $true;
$newsmi = $global{$buttonsmi};
if($sb{$new_variables{$buttonmod}} ne ""){
   for($i=0;$i<$new_variables{$buttonnearct};$i++){
if($newsmi eq ""){
   $newsmi = $sb{$new_variables{$buttonmod}};
}
else{
   $newsmi = join(".",$newsmi,$sb{$new_variables{$buttonmod}});
}
    }
}
$global{$buttonsmi} = $newsmi,
    }

# set joining as true or false

   if($new_variables{$buttonext}){
$joinstring = ";\$",
    }
   else{
$joinstring = ";!\$";
    }

# make modifier smarts string

   $modifystring = &GET_ATTACHMENT($new_variables{$buttonmod},
```

```
   $near{$new_variables{$buttonnear}},
   $new_variables{$buttonnearct});

# modify global string

   $global{$buttonsma} = join($joinstring,
substr($global{$buttonsma},0,length($global{$buttonsma})-1),
$modifystring);

#   DBLPRINT ("$buttonsma = $global{$buttonsma}\n$buttonsmi = $global{$buttonsmi}\n");
}

###put ring requirement at beginning if necessary***

if((substr($variable, 0, 4) eq "ring")&&($new_variables{$variable} ne "ignore")){

   $buttonsma = join("", "essential_smarts_button", substr($variable,4));
   $buttonring = join("", "ring", substr($variable,4));

   if($new_variables{$buttonring}){
$global{$buttonsma} = join("", "[R;",
   substr($global{$buttonsma},1));
   }
   elsif(!$new_variables{$buttonring}){
$global{$buttonsma} = join("", "[R0;",
   substr($global{$buttonsma},1)),
   }

}

###append steric stuff to primary button smarts***

if((substr($variable, 0, 6) eq "steric")&&($new_variables{$variable} ne "ignore")){
   $buttonsma = join("", "essential_smarts_button", substr($variable,6));
   $buttonbranch = join("", "branch", substr($variable,6));
   $modifystring = join("", "(*", $branch{$new_variables{$buttonbranch}}, ")]");
   if($new_variables{$variable} eq "IF"){
$joinstring = ";\$";
   }
   else{
$joinstring = ";!\$";
   }
   $global{$buttonsma} = join($joinstring,
   substr($global{$buttonsma},0,length($global{$buttonsma})-1),
   $modifystring);
#   DBLPRINT ("Steric constraint added to $buttonsma.\n");
#   DBLPRINT ("$buttonsma = $global{$buttonsma}.\n");
}

###replace spaces in "reaction" with underscores (good for library output###
if($variable eq "reaction"){
   $global{$variable} =~ s/ /_/g;
}

###un-esacpe unix saftey characters in smarts input    ###

if($variable eq "essential_smarts"){
   $global{$variable} =~ s/\\!/!/g;
   $global{$variable} =~ s/\\&/&/g;
}

###set symmetry number for removal of multiple occurances of buttons   ###

if((substr($variable,0,6) eq "double")&&($new_variables{$variable})){
   $button = join("", "essential_smarts_button", substr($variable,6));
   if($symmetry{$new_variables{$button}}){
$global{$variable} = $symmetry{$new_variables{$button}};
   }
}
   }
   DBLPRINT ("\n");
}

##########################
#get cgi post input
##########################
sub READPOST {
   local(%variables) = @_;

   if(($ENV{'REQUEST_METHOD'} eq 'POST')&&
     ($ENV{'CONTENT_TYPE'} eq 'application/x-www-form-urlencoded')){
read(STDIN, $buf, $ENV{'CONTENT_LENGTH'});

@pairs = split(/&/, $buf);
foreach $pair (@pairs){
   ($key, $data) = split(/=/, $pair);
   $data =~ tr/+/ /;
   $data =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C", hex($1))/eg;
   $data =~ s/\!/\\!/g;
   $data =~ s/\\/\\\\/g,
   $data =~ s/\&/\\&/g;
   $variables{$key}=$data;
    DBLPRINT("$key = $variables{$key}\n");
}
  DBLPRINT("\n\n");
    }

   elsif(($ENV{'REQUEST_METHOD'} eq 'POST')&&
  (substr($ENV{'CONTENT_TYPE'},0,19) eq 'multipart/form-data')){
read(STDIN, $buf, $ENV{'CONTENT_LENGTH'});

printf ("Content-type: text/html\n\n");
printf ("<HTML><BODY>");
printf ("\n\n***\tStarting Library Design 1.0\t***\n\n");

($dum, $boundary) = split(/boundary=/, $ENV{'CONTENT_TYPE'});
$boundary = join("", "--", $boundary);
@lines = split(/$boundary/, $buf);
```

```
foreach $line (@lines){
   ($dum, $key, $predata, $filename, $molecules) = split(/"/, $line);
   ($dum, $data, $dum) = split(/\s+/, $predata);

###parse variables   ###

   if($key ne "hitlist"){
$data =~ tr/+/ /;
$data =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C", hex($1))/eg;
$data =~ s/\'/\\'/g;
$data =~ s/\@/\\@/g;
$data =~ s/\&/\\&/g;
$variables{$key}=$data;
 printf LOG ("$key = $variables{$key}\n");
   }

###parse histlist file###

   else{
 printf LOG ("hitlist = $filename\n");
($dum, @molecules) = split(/\s+/, $molecules);
foreach $molecule (@molecules){
    printf LOG ("$molecule\n");
}
   }
}
 printf LOG ("\n\n");
   }
   return(%variables);
}
##########################
#initialize
##########################
sub INITIALIZE {

$false = 0;
$true = 1;

###    date information      ###

$min = (localtime)[1];
$hour = (localtime)[2];
$mday = (localtime)[3];
$mon = (localtime)[4];
$year = (localtime)[5];
$wday = (localtime)[6];
@days = (Sunday,Monday,Tuesday,Wednesday,Thursday,Friday,Saturday);
$today = $days[$wday];
@months = (Jan,Feb,Mar,Apr,May,June,July,Aug,Sep,Oct,Nov,Dec);
@months = (January,February,March,April,May,June,July,August,September,
   October,November,December);
@dom = (31,28,31,30,31,30,31,31,30,31,30,31);
if($year%4 == 0){
   $dom[1]++;
}

###write general info to log   ###

open(LOG, ">>design.log")|| die "can't open log";

printf LOG ("\n\n***\tStarting Library Design 0.0\t***\n\n");
printf LOG ("Time: $hour:$min, $mon/$mday/$year\n");
printf LOG ("Remote Host: $ENV{'REMOTE_HOST'}\n");
printf LOG ("Remote Address: $ENV{'REMOTE_ADDR'}\n");
printf LOG ("Remote User: $ENV{'REMOTE_USER'}\n");
printf LOG ("Query String: $ENV{'QUERY_STRING'}\n");

###    general selection     ###

      $global{"mw_max"} = 200;
      $global{"mw_min"} = 100;
      $global{"usdpergm"} = 101;
      $global{"totusd"} = 100;
$global{"logp_max"} = 3;
$global{"logp_min"} = -2;
$global{"logp_err"} = "";
$global{"hbd_max"} = 5;
$global{"hbd_min"} = 1;
$global{"hba_max"} = 10;
$global{"hba_min"} = 1;
$global{"roth_max"} = 9;
$global{"roth_min"} = 0;
$global{"crg_max"} = 2;
$global{"crg_min"} = 0;
$global{"essential_smiles"} = "";
      $global{"essential_smarts_button1"} = "";
$global{"double1"} = $false;
$global{"ring1"} = $false;
$global{"extra1"} = $true;
$global{"near1"} = "";
$global{"modify1"} = "";
      $global{"essential_smarts_button2"} = "";
$global{"double2"} = $false;
$global{"extra2"} = $true;
$global{"near2"} = "";
$global{"modify2"} = "";
      $global{"essential_smarts_button3"} = "";
$global{"double3"} = $false;
$global{"extra3"} = $true;
$global{"near3"} = "";
$global{"modify3"} = "";
      $global{"essential_smarts"} = "";
$global{"bad_double_smarts"} = "";
$global{"server"} = "yorick.ucsf.edu";
$global{"pool"} = "acd.972";
$global{"supplier_type"} = "";
$global{"output_type"} = "tdt";
```

```
$global{"reaction"} = "";
$global{"essential_name"} = "";
$global{"fcd_number"} = "";
$global{"save_hit"} = $false;

###    negative functional groups     ###

      $bad[0] = "acid_halide";
      $bad[1] = "sulfonic_acid";
      $bad[2] = "sulfonic_ester";
      $bad[3] = "anhydride";
      $bad[4] = "peroxide";
      $bad[5] = "nonstandardatom";
      $bad[6] = "azide";
      $bad[7] = "azo";
      $bad[8] = "unbranched_chain";
      $bad[9] = "four_halides";
      $bad[10] = "two_charges";
      $bad[11] = "long_chain";
      $bad[12] = "two_nitros";
      $bad[13] = "dipeptide";
      $bad[14] = "macrocycle";

$max_bad = 15;

$global{"acid_halide"} = $false;
      $global{"sulfonic_acid"} = $false;
      $global{"sulfonic_ester"} = $false;
      $global{"anhydride"} = $false;
      $global{"peroxide"} = $false;
      $global{"nonstandardatom"} = $false;
      $global{"azide"} = $false;
      $global{"azo"} = $false;
      $global{"unbranched_chain"} = $false;
      $global{"four_halides"} = $false;
      $global{"two_charges"} = $false;
      $global{"long_chain"} = $false;
      $global{"two_nitros"} = $false;
      $global{"dipeptide"} = $false;
      $global{"macrocycle"} = $false;

###    positive functional groups     ###

      $good[0] = "alcohol";
      $good[1] = "alkyne";
      $good[2] = "arene";
      $good[3] = "ether";
      $good[4] = "nitrile";
      $good[5] = "sulfide";
      $good[6] = "thiol";
      $good[7] = "aldehyde";
      $good[8] = "amide";
      $good[9] = "carboxylic_acid";
      $good[10] = "halide";
      $good[11] = "nitro";
      $good[12] = "sulfone";
      $good[13] = "aniline";
      $good[14] = "alkene";
      $good[15] = "amine";
      $good[16] = "ester";
      $good[17] = "ketone";
      $good[18] = "phenol";
      $good[19] = "sulfoxide";
      $good[20] = "amino_acid";
      $good[21] = "urea";
      $good[22] = "thiourea";
      $good[23] = "thioamide";
      $good[24] = "carbamate";
      $good[25] = "hydrazone";
      $good[26] = "hydrazine";
      $good[27] = "hydroxylamine";
      $good[28] = "alkyl_halide";
      $good[29] = "aryl_halide";
      $good[30] = "ring";
$good[31] = "primary_amine";
$good[32] = "secondary_amine";
$good[33] = "tertiary_amine";
$good[34] = "phosphonic_acid";
$good[35] = "phosphonic_ester";
$good[36] = "phosphoric_acid";
$good[37] = "phosphoric_ester";
$good[38] = "isocyanate";
$good[39] = "imine";

      $good[40] = "nucleophile";
      $good[41] = "electrophile";

      $max_good = 42;

$global{"alcohol"} = $true;
      $global{"alkyne"} = $true;
      $global{"arene"} = $true;
      $global{"ether"} = $true;
      $global{"nitrile"} = $true;
      $global{"sulfide"} = $true;
      $global{"thiol"} = $true;
      $global{"aldehyde"} = $true;
      $global{"amide"} = $true;
      $global{"carboxylic_acid"} = $true;
      $global{"halide"} = $true;
      $global{"nitro"} = $true;
      $global{"sulfone"} = $true;
      $global{"aniline"} = $true;
      $global{"alkene"} = $true;
      $global{"amine"} = $true;
      $global{"ester"} = $true;
      $global{"ketone"} = $true;
      $global{"phenol"} = $true;
```

```
        $global{"sulfoxide"} = $true;
        $global{"amino_acid"} = $true,
        $global{"urea"} = $true;
        $global{"thiourea"} = $true;
        $global{"thioamide"} = $true;
        $global{"carbamate"} = $true;
        $global{"hydrazone"} = $true;
        $global{"hydrazine"} = $true;
        $global{"hydroxylamine"} = $true;
        $global{"alkyl_halide"} = $true;
        $global{"aryl_halide"} = $true;
        $global{"ring"} = $true,
$global{"combi_linker"} = $true;
$global{"combi_tcn"} = $true;
$global{"combi_any"} = $true,
$global{"alkylating_agent"} = $true,
        $global{"ring"} = $true;
$global{"primary_amine"} = $true,
$global{"secondary_amine"} = $true,
$global{"tertiary_amine"} = $true,
$global{"phosphonic_acid"} = $true;
$global{"phosphonic_ester"} = $true;
$global{"isocyanate"} = $true;
$global{"imine"} = $true,


        $global{"nucleophile"} = $true;
        $global{"electrophile"} = $true;

###vbindings###

%vb = &READVBIND(%vb);
%sb = &READSBIND(%sb),

###vbinding symmetry counts   ###

$symmetry{"alkene"} = 2;
$symmetry{"alkyne"} = 2;
$symmetry{"anhydride"} = 2;
$symmetry{"arene"} = 6,
$symmetry{"aryl"} = 6,
$symmetry{"azo"} = 2,
$symmetry{"triazine"} = 2,
$symmetry{"carbamate"} = 2,
$symmetry{"thiocarbamate"} = 2;
$symmetry{"carbonate"} = 2,
$symmetry{"thiourea"} = 2,
$symmetry{"urea"} = 2,
$symmetry{"epoxide"} = 2,
$symmetry{"disulfide"} = 2,
$symmetry{"oxalyl"} = 2,

###smarts for proximity###

$near{"alpha"} = "~A~",
$near{"beta"} = "~A~A~";
$near{"gamma"} = "~A~A~A~",
$near{"ortho"} = "aa",
$near{"meta"} = "aaa";
$near{"para"} = "aaaa",

###smarts for branching###

$branch{"alpha_branched"} = "~[D3,D4]";
$branch{"alpha_t_branched"} = "~[D4]";
$branch{"beta_branched"} = "~*~[D3,D4]",
$branch{"beta_t_branched"} = "~*~[D4]";
$branch{"gamma_branched"} = "~*~*~[D3,D4]",
$branch{"gamma_t_branched"} = "~*~*~[D4]",

###suppliers###

$global{"aldrich"} = $false;
$global{"fluka"} = $false;
$global{"sigma"} = $false,
$global{"lancaster"} = $false,
$global{"tci_america"} = $false,
$global{"icn"} = $false,
$global{"pfaltz_bauer"} = $false;
$global{"indofine"} = $false;
$global{"acros_organics"} = $false;
$global{"calbiochem"} = $false;
$global{"maybr_int"} = $false,
$global{"maybridge"} = $false,
$global{"salor"} = $false,
$global{"trans_world"} = $false,

$goodsupplier{"aldrich"} = "ALDRICH";
$goodsupplier{"fluka"} = "FLUKA";
$goodsupplier{"sigma"} = "SIGMA",
$goodsupplier{"lancaster"} = "LANCASTER",
$goodsupplier{"tci_america"} = "TCIUS",
$goodsupplier{"icn"} = "ICN";
$goodsupplier{"pfaltz_bauer"} = "PFALTZBAUER",
$goodsupplier{"indofine"} = "INDOFINE",
$goodsupplier{"acros_organics"} = "ACROS",
$goodsupplier{"calbiochem"} = "CALBIO",
$goodsupplier{"maybr_int"} = "MAYBRINT",
$goodsupplier{"maybridge"} = "MAYBRIDGE";
$goodsupplier{"salor"} = "SALOR";
$goodsupplier{"trans_world"} = "TRANSWLD",

}

##########################
#initialize the smiles bindings
##########################
sub READSBIND {
   local(%sbindings) = @_;

   open(SB, "/idk1/people/skillman/sb.srt")|| die "can't open vb";

   while(<SB>){
chop($_);
($name, $smiles) = split(/\s+/, $_);
$sbindings{$name} = $smiles;
   }
   close(SB);
   return(%sbindings);
}

##########################
#initialize the vbindings
##########################
sub READVBIND {
   local(%vbindings) = @_;

   open(VB, "/idk1/people/skillman/vb.use")|| die "can't open vb";

   while(<VB>){
chop($_);
($name, $smarts) = split(/\s+/, $_);
$vbindings{$name} = $smarts;

@smartlist = (values %vbindings);

if( @fix = grep(/[\$]$name/, @smartlist)){
   foreach $key (keys %vbindings){
$vbindings{$key} =~ s/[\$]$name/substr($vbindings{$name},1,length($vbindings{$name})-2)/eg;
   }
}
   }

### fix the logical AND in the combi v-bindings ###

   $vbindings{"combi_linker"} =~ s/;/&/g;
   $vbindings{"combi_tcn"} =~ s/;/&/g;
   $vbindings{"combi_any"} =~ s/;/&/g;

   close(VB);
   return(%vbindings);
}

##########################
#print form for input
##########################
sub PRINTFORM {
   printf ("Content-type: text/html\n\n");
   printf ("<!DOCTYPE HTML PUBLIC \"-//IETF//DTD HTML//EN\">\n");
   printf ("<HTML><HEAD>\n");
   printf ("<TITLE>UC_Select 1.01</TITLE>\n");
   printf ("</HEAD>\n");
   printf ("\n");
   printf ("\n");
   printf ("<BODY BGCOLOR=\"#CFCFEF\">\n");
   printf ("\n"),
   printf ("<!-- end of header -->\n");
   printf ("\n");
   printf ("<CENTER>\n");
   printf ("<H1>UC_Select 1.01</H1>\n");
   printf ("&copy;1997, University of California, San Francisco\n");
   printf ("</CENTER>\n"),
   printf ("\n"),
   printf ("<HR>\n");
   printf ("\n");
   printf ("<FORM enctype=\"multipart/form-data\" METHOD = \"POST\" ACTION=\"$url_path/
reagent\">\n");
   printf ("\n");
   printf ("<H3>Log Information</H3>\n");
   printf ("Name<BR>          \n");
   printf ("<INPUT SIZE=60 NAME=\"user_name\"><P>\n");
   printf ("Project<BR>\n"),
   printf ("<INPUT SIZE=30 NAME=\"project\"><P>\n");
   printf ("Reaction<BR>\n");
   printf ("<INPUT SIZE=60 NAME=\"reaction\"><P>\n");
   printf ("\n");
   printf ("<TABLE WIDTH=2000%><TR><TD><HR></TD><TD><INPUT TYPE=\"submit\"></
TD></TR></TABLE>\n");
   printf ("<H3>Server Selection</H3>\n");
   printf ("Server<BR>\n");
   printf ("<INPUT SIZE=30 VALUE=\"nobel1.ucsf.edu\" NAME=\"server\"><P>\n");
   printf ("<H3>Database Selection</H3>\n"),
   printf ("Pool<BR>\n");
   printf ("<INPUT SIZE=30 VALUE=\"acd 982\" NAME=\"pool\"><P>\n");
   printf ("<TABLE WIDTH=2000%><TR><TD><HR></TD><TD><INPUT TYPE=\"submit\"></
TD></TR></TABLE>\n");
   printf ("<H3>Hitlists</H3>\n");
   printf ("Starting Hitlist File (SMILES file, one line per compound)<BR>\n");
   printf ("<INPUT TYPE=file SIZE=60 NAME=\"hitlist\"><BR>\n");
   printf ("There is an option to save a hitlist at the end of the output.<P>\n");
   printf ("<TABLE WIDTH=2000%><TR><TD><HR></TD><TD><INPUT TYPE=\"submit\"></
TD></TR></TABLE>\n");
   printf ("<H3>Compound Selection</H3>\n");
   printf ("First choose a primary functionality which is desired in every molecule.<P>\n");
   printf ("Essential <A HREF=./smiles.help>Smiles </A>(optional)<BR>\n");
   printf ("<A HREF=./Editor.html>Try this java SMILES editor!</A><BR>          \n");
   printf ("<INPUT SIZE=60 NAME=\"essential_smiles\"><P>\n");
   printf ("\n");
   printf ("Required Functional Group<BR>\n");
   printf ("<SELECT NAME=\"essential_smarts_button1\">\n");
   printf ("<OPTION SELECTED>none\n");
   printf ("<OPTION>acetal\n"),
   printf ("<OPTION>acid_chloride\n");
   printf ("<OPTION>acid_halide\n"),
   printf ("<OPTION>alcohol\n");
   printf ("<OPTION>aldehyde\n"),
```

```
printf ("<OPTION>alkene\n");
printf ("<OPTION>alkyl\n");
printf ("<OPTION>alkyl_halide\n");
printf ("<OPTION>alkylating_agent\n");
printf ("<OPTION>alkyne\n");
printf ("<OPTION>amide\n");
printf ("<OPTION>amino_acid\n");
printf ("<OPTION>anhydride\n");
printf ("<OPTION>aniline\n");
printf ("<OPTION>aniline_unsubstituted\n");
printf ("<OPTION>aryl_halide\n");
printf ("<OPTION>azide\n");
printf ("<OPTION>azo\n");
printf ("<OPTION>carbamate\n");
printf ("<OPTION>carbamic_acid\n");
printf ("<OPTION>carbonate\n");
printf ("<OPTION>carbonyl\n");
printf ("<OPTION>carboxylic_acid\n");
printf ("<OPTION>combi_linker\n");
printf ("<OPTION>combi_fcn\n");
printf ("<OPTION>combi_any\n");
printf ("<OPTION>disulfide\n");
printf ("<OPTION>dithioacetal\n");
printf ("<OPTION>enamine\n");
printf ("<OPTION>enol_ether\n");
printf ("<OPTION>epoxide\n");
printf ("<OPTION>ester\n");
printf ("<OPTION>ether\n");
printf ("<OPTION>hemiacetal\n");
printf ("<OPTION>hemiketal\n");
printf ("<OPTION>hydrazine\n");
printf ("<OPTION>hydrazone\n");
printf ("<OPTION>imino\n");
printf ("<OPTION>isocyanate\n");
printf ("<OPTION>isothiocyanate\n");
printf ("<OPTION>ketal\n");
printf ("<OPTION>ketone\n");
printf ("<OPTION>lactam\n");
printf ("<OPTION>lactone\n");
printf ("<OPTION>nitrile\n");
printf ("<OPTION>nitro\n");
printf ("<OPTION>organometallic\n");
printf ("<OPTION>oxalyl\n");
printf ("<OPTION>oxime\n");
printf ("<OPTION>peroxide\n");
printf ("<OPTION>phenol\n");
printf ("<OPTION>phosphonic_acid\n");
printf ("<OPTION>phosphonic_ester\n");
printf ("<OPTION>primary_amine\n");
printf ("<OPTION>secondary_amine\n");
printf ("<OPTION>sulfide\n");
printf ("<OPTION>sulfonamide\n");
printf ("<OPTION>sulfone\n");
printf ("<OPTION>sulfonic_acid\n");
printf ("<OPTION>sulfonic_ester\n");
printf ("<OPTION>sulfonyl_halide\n");
printf ("<OPTION>sulfoxide\n");
printf ("<OPTION>tertiary_amine\n");
printf ("<OPTION>thiocarbonyl\n");
printf ("<OPTION>thioether\n");
printf ("<OPTION>thiol\n");
printf ("<OPTION>thiourea\n");
printf ("<OPTION>triazine\n");
printf ("<OPTION>urea\n");
printf ("</SELECT>\n");
printf ("Discard Double Occurances?\n");
printf ("<INPUT TYPE=\"radio\" NAME=\"double1\" VALUE=\"TRUE\" CHECKED>Yes\n");
printf ("<INPUT TYPE=\"radio\" NAME=\"double1\" VALUE=\"FALSE\">No<BR>\n");
printf ("\n");
printf ("OK In Ring?\n");
printf ("<INPUT TYPE=\"radio\" NAME=\"ring1\" VALUE=\"ignore\">Don't Care\n");
printf ("<INPUT TYPE=\"radio\" NAME=\"ring1\" VALUE=\"TRUE\">Yes\n");
printf ("<INPUT TYPE=\"radio\" NAME=\"ring1\" VALUE=\"FALSE\">No<BR>\n");
printf ("\n");
printf ("<INPUT TYPE=\"radio\" NAME=\"extra1\" VALUE=\"TRUE\" CHECKED>if \n");
printf ("<INPUT TYPE=\"radio\" NAME=\"extra1\" VALUE=\"FALSE\">not\n");
printf ("\n");
printf ("<SELECT NAME=\"near1\">\n");
printf ("<OPTION SELECTED>alpha\n");
printf ("<OPTION>beta\n");
printf ("<OPTION>gamma\n");
printf ("<OPTION>ortho\n");
printf ("<OPTION>meta\n");
printf ("<OPTION>para\n");
printf ("<OPTION>any_attachment\n");
printf ("</SELECT>\n");
printf ("\n");
printf (" to a <SELECT NAME=\"modify1\">\n");
printf ("<OPTION SELECTED>none\n");
printf ("<OPTION>acetal\n");
printf ("<OPTION>acid_chloride\n");
printf ("<OPTION>acid_halide\n");
printf ("<OPTION>alcohol\n");
printf ("<OPTION>aldehyde\n");
printf ("<OPTION>alkene\n");
printf ("<OPTION>alkyl\n");
printf ("<OPTION>alkyl_halide\n");
printf ("<OPTION>alkyne\n");
printf ("<OPTION>amide\n");
printf ("<OPTION>anhydride\n");
printf ("<OPTION>aniline\n");
printf ("<OPTION>azide\n");
printf ("<OPTION>azo\n");
printf ("<OPTION>carbamate\n");
printf ("<OPTION>carbamic_acid\n");
printf ("<OPTION>carbonate\n");
printf ("<OPTION>carbonyl\n");
printf ("<OPTION>carboxylic_acid\n");
```

```
printf ("<OPTION>combi_linker\n");
printf ("<OPTION>combi_fcn\n");
printf ("<OPTION>combi_any\n");
printf ("<OPTION>disulfide\n");
printf ("<OPTION>dithioacetal\n");
printf ("<OPTION>enamine\n");
printf ("<OPTION>enol_ether\n");
printf ("<OPTION>epoxide\n");
printf ("<OPTION>ester\n");
printf ("<OPTION>ether\n");
printf ("<OPTION>hemiacetal\n");
printf ("<OPTION>hemiketal\n");
printf ("<OPTION>hydrazine\n");
printf ("<OPTION>hydrazone\n");
printf ("<OPTION>imino\n");
printf ("<OPTION>isocyanate\n");
printf ("<OPTION>isothiocyanate\n");
printf ("<OPTION>ketal\n");
printf ("<OPTION>ketone\n");
printf ("<OPTION>lactam\n");
printf ("<OPTION>lactone\n");
printf ("<OPTION>nitrile\n");
printf ("<OPTION>nitro\n");
printf ("<OPTION>organometallic\n");
printf ("<OPTION>oxalyl\n");
printf ("<OPTION>oxime\n");
printf ("<OPTION>peroxide\n");
printf ("<OPTION>phenol\n");
printf ("<OPTION>phosphonic_acid\n");
printf ("<OPTION>phosphonic_ester\n");
printf ("<OPTION>primary_amine\n");
printf ("<OPTION>ring\n");
printf ("<OPTION>secondary_amine\n");
printf ("<OPTION>sulfide\n");
printf ("<OPTION>sulfonamide\n");
printf ("<OPTION>sulfone\n");
printf ("<OPTION>sulfonic_acid\n");
printf ("<OPTION>sulfonic_ester\n");
printf ("<OPTION>sulfonyl_halide\n");
printf ("<OPTION>sulfoxide\n");
printf ("<OPTION>tertiary_amine\n");
printf ("<OPTION>thiocarbonyl\n");
printf ("<OPTION>thioether\n");
printf ("<OPTION>thiol\n");
printf ("<OPTION>thiourea\n");
printf ("<OPTION>triazine\n");
printf ("<OPTION>urea\n");
printf ("</SELECT>\n");
printf ("<BR>\n");
printf ("\n");
printf ("<INPUT TYPE=\"radio\" NAME=\"steric1\" VALUE=\"ignore\" CHECKED>ignore\n");
printf ("<INPUT TYPE=\"radio\" NAME=\"steric1\" VALUE=\"IF\" >if \n");
printf ("<INPUT TYPE=\"radio\" NAME=\"steric1\" VALUE=\"NOT\">not\n");
printf ("\n");
printf ("<SELECT NAME=\"branch1\">\n");
printf ("<OPTION SELECTED>alpha_branched\n");
printf ("<OPTION>alpha_t_branched\n");
printf ("<OPTION>beta_branched\n");
printf ("<OPTION>beta_t_branched\n");
printf ("<OPTION>gamma_branched\n");
printf ("<OPTION>gamma_t_branched\n");
printf ("</SELECT><BR>\n");
printf ("<B>NOTE: Currently has a \"feature\" which takes branching in the functional \n");
printf ("group itself into account.  For example, Ureas will ALWAYS have alpha \n");
printf ("branching because the carbonyl carbon is branched and is alpha to the \n");
printf ("urea nitrogen.  However, it should work fine for selecting amines or \n");
printf ("carboxylic acids for instance.\n");
printf ("</B>\n");
printf ("<P>\n");
printf ("\n");
printf ("<H4>AND</H4><BR>\n");
printf ("Required Functional Group<BR>\n");
printf ("<SELECT NAME=\"essential_smarts_button2\">\n");
printf ("<OPTION SELECTED>none\n");
printf ("<OPTION>acetal\n");
printf ("<OPTION>acid_chloride\n");
printf ("<OPTION>acid_halide\n");
printf ("<OPTION>alcohol\n");
printf ("<OPTION>aldehyde\n");
printf ("<OPTION>alkene\n");
printf ("<OPTION>alkyl\n");
printf ("<OPTION>alkyl_halide\n");
printf ("<OPTION>alkyne\n");
printf ("<OPTION>amide\n");
printf ("<OPTION>anhydride\n");
printf ("<OPTION>aniline\n");
printf ("<OPTION>azide\n");
printf ("<OPTION>azo\n");
printf ("<OPTION>carbamate\n");
printf ("<OPTION>carbamic_acid\n");
printf ("<OPTION>carbonate\n");
printf ("<OPTION>carbonyl\n");
printf ("<OPTION>carboxylic_acid\n");
printf ("<OPTION>disulfide\n");
printf ("<OPTION>dithioacetal\n");
printf ("<OPTION>enamine\n");
printf ("<OPTION>enol_ether\n");
printf ("<OPTION>epoxide\n");
printf ("<OPTION>ester\n");
printf ("<OPTION>ether\n");
printf ("<OPTION>hemiacetal\n");
printf ("<OPTION>hemiketal\n");
printf ("<OPTION>hydrazine\n");
printf ("<OPTION>hydrazone\n");
printf ("<OPTION>imino\n");
printf ("<OPTION>isocyanate\n");
printf ("<OPTION>isothiocyanate\n");
printf ("<OPTION>ketal\n");
```

```
printf ("<OPTION>ketone\n");
printf ("<OPTION>lactam\n");
printf ("<OPTION>lactone\n");
printf ("<OPTION>nitrile\n");
printf ("<OPTION>nitro\n");
printf ("<OPTION>organometallic\n");
printf ("<OPTION>oxalyl\n");
printf ("<OPTION>oxime\n");
printf ("<OPTION>peroxide\n");
printf ("<OPTION>phenol\n");
printf ("<OPTION>phosphonic_acid\n");
printf ("<OPTION>phosphonic_ester\n");
printf ("<OPTION>primary_amine\n");
printf ("<OPTION>secondary_amine\n");
printf ("<OPTION>sulfide\n");
printf ("<OPTION>sulfonamide\n");
printf ("<OPTION>sulfone\n");
printf ("<OPTION>sulfonic_acid\n");
printf ("<OPTION>sulfonic_ester\n");
printf ("<OPTION>sulfonyl_halide\n");
printf ("<OPTION>sulfoxide\n");
printf ("<OPTION>tertiary_amine\n");
printf ("<OPTION>thiocarbonyl\n");
printf ("<OPTION>thioether\n");
printf ("<OPTION>thiol\n");
printf ("<OPTION>thiourea\n");
printf ("<OPTION>triazine\n");
printf ("<OPTION>urea\n");
printf ("</SELECT>\n");
printf ("Discard Double Occurances?\n");
printf ("<INPUT TYPE=\"radio\" NAME=\"double2\" VALUE=\"TRUE\" CHECKED>Yes\n");
printf ("<INPUT TYPE=\"radio\" NAME=\"double2\" VALUE=\"FALSE\">No<BR>\n");
printf ("\n");
printf ("<INPUT TYPE=\"radio\" NAME=\"extra2\" VALUE=\"TRUE\" CHECKED>if \n");
printf ("<INPUT TYPE=\"radio\" NAME=\"extra2\" VALUE=\"FALSE\">not\n");
printf ("\n");
printf ("<SELECT NAME=\"near2\">\n");
printf ("<OPTION SELECTED>alpha\n");
printf ("<OPTION>beta\n");
printf ("<OPTION>gamma\n");
printf ("<OPTION>ortho\n");
printf ("<OPTION>meta\n");
printf ("<OPTION>para\n");
printf ("</SELECT>\n");
printf (" to a <SELECT NAME=\"modify2\">\n");
printf ("<OPTION SELECTED>none\n");
printf ("<OPTION>acetal\n");
printf ("<OPTION>acid_chloride\n");
printf ("<OPTION>acid_halide\n");
printf ("<OPTION>alcohol\n");
printf ("<OPTION>aldehyde\n");
printf ("<OPTION>alkene\n");
printf ("<OPTION>alkyl\n");
printf ("<OPTION>alkyl_halide\n");
printf ("<OPTION>alkyne\n");
printf ("<OPTION>amide\n");
printf ("<OPTION>anhydride\n");
printf ("<OPTION>aniline\n");
printf ("<OPTION>azide\n");
printf ("<OPTION>azo\n");
printf ("<OPTION>carbamate\n");
printf ("<OPTION>carbamic_acid\n");
printf ("<OPTION>carbonate\n");
printf ("<OPTION>carbonyl\n");
printf ("<OPTION>carboxylic_acid\n");
printf ("<OPTION>disulfide\n");
printf ("<OPTION>dithioacetal\n");
printf ("<OPTION>enamine\n");
printf ("<OPTION>enol_ether\n");
printf ("<OPTION>epoxide\n");
printf ("<OPTION>ester\n");
printf ("<OPTION>ether\n");
printf ("<OPTION>hemiacetal\n");
printf ("<OPTION>hemiketal\n");
printf ("<OPTION>hydrazine\n");
printf ("<OPTION>hydrazone\n");
printf ("<OPTION>imino\n");
printf ("<OPTION>isocyanate\n");
printf ("<OPTION>isothiocyanate\n");
printf ("<OPTION>ketal\n");
printf ("<OPTION>ketone\n");
printf ("<OPTION>lactam\n");
printf ("<OPTION>lactone\n");
printf ("<OPTION>nitrile\n");
printf ("<OPTION>nitro\n");
printf ("<OPTION>organometallic\n");
printf ("<OPTION>oxalyl\n");
printf ("<OPTION>oxime\n");
printf ("<OPTION>peroxide\n");
printf ("<OPTION>phenol\n");
printf ("<OPTION>phosphonic_acid\n");
printf ("<OPTION>phosphonic_ester\n");
printf ("<OPTION>primary_amine\n");
printf ("<OPTION>secondary_amine\n");
printf ("<OPTION>sulfide\n");
printf ("<OPTION>sulfonamide\n");
printf ("<OPTION>sulfone\n");
printf ("<OPTION>sulfonic_acid\n");
printf ("<OPTION>sulfonic_ester\n");
printf ("<OPTION>sulfonyl_halide\n");
printf ("<OPTION>sulfoxide\n");
printf ("<OPTION>tertiary_amine\n");
printf ("<OPTION>thiocarbonyl\n");
printf ("<OPTION>thioether\n");
printf ("<OPTION>thiol\n");
printf ("<OPTION>thiourea\n");
printf ("<OPTION>triazine\n");
printf ("<OPTION>urea\n");
```

```
printf ("</SELECT>\n");
printf ("<P>\n");
printf ("\n");
printf ("<INPUT TYPE=\"radio\" NAME=\"steric2\" VALUE=\"ignore\" CHECKED>ignore\n");
printf ("<INPUT TYPE=\"radio\" NAME=\"steric2\" VALUE=\"IF\" >if \n");
printf ("<INPUT TYPE=\"radio\" NAME=\"steric2\" VALUE=\"NOT\">not\n");
printf ("\n");
printf ("<SELECT NAME=\"branch2\">\n");
printf ("<OPTION SELECTED>alpha_branched\n");
printf ("<OPTION>alpha_t_branched\n");
printf ("<OPTION>beta_branched\n");
printf ("<OPTION>beta_t_branched\n");
printf ("<OPTION>gamma_branched\n");
printf ("<OPTION>gamma_t_branched\n");
printf ("</SELECT><BR>\n");
printf ("<P>\n");
printf ("\n");
printf ("<H4>AND</H4><BR>\n");
printf ("Required Functional Group<BR>\n");
printf ("<SELECT NAME=\"essential_smarts_button3\">\n");
printf ("<OPTION SELECTED>none\n");
printf ("<OPTION>acetal\n");
printf ("<OPTION>acid_chloride\n");
printf ("<OPTION>acid_halide\n");
printf ("<OPTION>alcohol\n");
printf ("<OPTION>aldehyde\n");
printf ("<OPTION>alkene\n");
printf ("<OPTION>alkyl\n");
printf ("<OPTION>alkyl_halide\n");
printf ("<OPTION>alkyne\n");
printf ("<OPTION>amide\n");
printf ("<OPTION>anhydride\n");
printf ("<OPTION>aniline\n");
printf ("<OPTION>azide\n");
printf ("<OPTION>azo\n");
printf ("<OPTION>carbamate\n");
printf ("<OPTION>carbamic_acid\n");
printf ("<OPTION>carbonate\n");
printf ("<OPTION>carbonyl\n");
printf ("<OPTION>carboxylic_acid\n");
printf ("<OPTION>disulfide\n");
printf ("<OPTION>dithioacetal\n");
printf ("<OPTION>enamine\n");
printf ("<OPTION>enol_ether\n");
printf ("<OPTION>epoxide\n");
printf ("<OPTION>ester\n");
printf ("<OPTION>ether\n");
printf ("<OPTION>hemiacetal\n");
printf ("<OPTION>hemiketal\n");
printf ("<OPTION>hydrazine\n");
printf ("<OPTION>hydrazone\n");
printf ("<OPTION>imino\n");
printf ("<OPTION>isocyanate\n");
printf ("<OPTION>isothiocyanate\n");
printf ("<OPTION>ketal\n");
printf ("<OPTION>ketone\n");
printf ("<OPTION>lactam\n");
printf ("<OPTION>lactone\n");
printf ("<OPTION>nitrile\n");
printf ("<OPTION>nitro\n");
printf ("<OPTION>organometallic\n");
printf ("<OPTION>oxalyl\n");
printf ("<OPTION>oxime\n");
printf ("<OPTION>peroxide\n");
printf ("<OPTION>phenol\n");
printf ("<OPTION>phosphonic_acid\n");
printf ("<OPTION>phosphonic_ester\n");
printf ("<OPTION>primary_amine\n");
printf ("<OPTION>secondary_amine\n");
printf ("<OPTION>sulfide\n");
printf ("<OPTION>sulfonamide\n");
printf ("<OPTION>sulfone\n");
printf ("<OPTION>sulfonic_acid\n");
printf ("<OPTION>sulfonic_ester\n");
printf ("<OPTION>sulfonyl_halide\n");
printf ("<OPTION>sulfoxide\n");
printf ("<OPTION>tertiary_amine\n");
printf ("<OPTION>thiocarbonyl\n");
printf ("<OPTION>thioether\n");
printf ("<OPTION>thiol\n");
printf ("<OPTION>thiourea\n");
printf ("<OPTION>triazine\n");
printf ("<OPTION>urea\n");
printf ("</SELECT>\n");
printf ("\n");
printf ("Discard Double Occurances?\n");
printf ("<INPUT TYPE=\"radio\" NAME=\"double3\" VALUE=\"TRUE\" CHECKED>Yes\n");
printf ("<INPUT TYPE=\"radio\" NAME=\"double3\" VALUE=\"FALSE\">No<BR>\n");
printf ("\n");
printf ("<INPUT TYPE=\"radio\" NAME=\"extra3\" VALUE=\"TRUE\" CHECKED>if \n");
printf ("<INPUT TYPE=\"radio\" NAME=\"extra3\" VALUE=\"FALSE\">not\n");
printf ("\n");
printf ("<SELECT NAME=\"near3\">\n");
printf ("<OPTION SELECTED>alpha\n");
printf ("<OPTION>beta\n");
printf ("<OPTION>ortho\n");
printf ("<OPTION>meta\n");
printf ("<OPTION>para\n");
printf ("</SELECT>\n");
printf (" to a <SELECT NAME=\"modify3\">\n");
printf ("<OPTION SELECTED>none\n");
printf ("<OPTION>acetal\n");
printf ("<OPTION>acid_chloride\n");
printf ("<OPTION>acid_halide\n");
printf ("<OPTION>alcohol\n");
printf ("<OPTION>aldehyde\n");
printf ("<OPTION>alkene\n");
printf ("<OPTION>alkyl\n");
```

```
    printf (“<OPTION>alkyl_halide\n”);
    printf (“<OPTION>alkyne\n”);
    printf (“<OPTION>amide\n”);
    printf (“<OPTION>anhydride\n”);
    printf (“<OPTION>aniline\n”);
    printf (“<OPTION>azide\n”);
    printf (“<OPTION>azo\n”);
    printf (“<OPTION>carbamate\n”);
    printf (“<OPTION>carbamic acid\n”);
    printf (“<OPTION>carbonate\n”);
    printf (“<OPTION>carbonyl\n”);
    printf (“<OPTION>carboxylic acid\n”);
    printf (“<OPTION>disulfide\n”);
    printf (“<OPTION>dithioacetal\n”);
    printf (“<OPTION>enamine\n”);
    printf (“<OPTION>enol ether\n”);
    printf (“<OPTION>epoxide\n”);
    printf (“<OPTION>ester\n”);
    printf (“<OPTION>ether\n”);
    printf (“<OPTION>hemiacetal\n”);
    printf (“<OPTION>hemiketal\n”);
    printf (“<OPTION>hydrazine\n”);
    printf (“<OPTION>hydrazone\n”);
    printf (“<OPTION>imino\n”);
    printf (“<OPTION>isocyanate\n”);
    printf (“<OPTION>isothiocyanate\n”);
    printf (“<OPTION>ketal\n”);
    printf (“<OPTION>ketone\n”);
    printf (“<OPTION>lactam\n”);
    printf (“<OPTION>lactone\n”);
    printf (“<OPTION>nitrile\n”);
    printf (“<OPTION>nitro\n”);
    printf (“<OPTION>organometallic\n”);
    printf (“<OPTION>oxalyl\n”);
    printf (“<OPTION>oxime\n”);
    printf (“<OPTION>peroxide\n”);
    printf (“<OPTION>phenol\n”);
    printf (“<OPTION>phosphonic acid\n”);
    printf (“<OPTION>phosphonic ester\n”);
    printf (“<OPTION>primary_amine\n”);
    printf (“<OPTION>secondary_amine\n”);
    printf (“<OPTION>sulfide\n”);
    printf (“<OPTION>sulfonamide\n”);
    printf (“<OPTION>sulfone\n”);
    printf (“<OPTION>sulfonic_acid\n”);
    printf (“<OPTION>sulfonic ester\n”);
    printf (“<OPTION>sulfonyl halide\n”);
    printf (“<OPTION>sulfoxide\n”);
    printf (“<OPTION>tertiary_amine\n”);
    printf (“<OPTION>thiocarbonyl\n”);
    printf (“<OPTION>thioether\n”);
    printf (“<OPTION>thiol\n”);
    printf (“<OPTION>thiourea\n”);
    printf (“<OPTION>triazine\n”);
    printf (“<OPTION>urea\n”);
    printf (“</SELECT>\n”);
    printf (“\n”);
    printf (“<INPUT TYPE=\"radio\" NAME=\"steric3\" VALUE=\"ignore\" CHECKED>ignore\n”);
    printf (“<INPUT TYPE=\"radio\" NAME=\"steric3\" VALUE=\"IF\" >if \n”);
    printf (“<INPUT TYPE=\"radio\" NAME=\"steric3\" VALUE=\"NOT\">not\n”);
    printf (“\n”);
    printf (“<SELECT NAME=\"branch3\">\n”);
    printf (“<OPTION SELECTED>alpha branched\n”);
    printf (“<OPTION>alpha t branched\n”);
    printf (“<OPTION>beta_branched\n”);
    printf (“<OPTION>beta t_branched\n”);
    printf (“<OPTION>gamma branched\n”);
    printf (“<OPTION>gamma t branched\n”);
    printf (“</SELECT><BR>\n”);
    printf (“\n”);
    printf (“<H4>AND</H4><BR>\n”);
    printf (“Other essential smarts\n”);
    printf (“<INPUT SIZE=60 NAME=\"essential_smarts\">\n”);
    printf (“\n”);
    printf (“<H4>AND</H4><BR>\n”);
    printf (“Compound name substring (optional) \n”);
    printf (“<INPUT SIZE=60 NAME=\"essential_name\">\n”);
    printf (“\n”);
    printf (“<H4>AND</H4><BR>\n”);
    printf (“Compound lcd substring (optional) \n”);
    printf (“<INPUT SIZE=60 NAME=\"lcd_number\">\n”);
    printf (“\n”);
    printf (“<TABLE WIDTH=2000% ><TR><TD><HR></TD><TD><INPUT TYPE=\"submit\"></
TD></TR></TABLE>\n”);
    printf (“<H3>Compound Property Profile</H3>\n”);
    printf (“Limit the properties of compounds selected.<P>\n”);
    printf (“<TABLE BORDER=2>\n”);
    printf (“<TR><TD>\n”);
    printf (“<INPUT SIZE=5 VALUE=\"450\" NAME=\"mw_max\">   Maximum Molecular
Weight<BR>\n”);
    printf (“<INPUT SIZE=5 VALUE=\"100\" NAME=\"mw_min\">   Minimum Molecular
Weight<BR>\n”);
    printf (“</TD></TR>\n”);
    printf (“<TR><TD>\n”);
    printf (“<INPUT SIZE=5 VALUE=\"5\" NAME=\"logp_max\">   Maximum ClogP<BR>\n”);
    printf (“<INPUT SIZE=5 VALUE=\"-2\" NAME=\"logp_min\">   Minimum ClogP<BR>\n”);
    printf (“<SELECT NAME=\"logp_err\">\n”);
    printf (“<OPTION> -0P;All fragments measured\n”);
    printf (“<OPTION> -10P;Valid estimate for difficult structure\n”);
    printf (“<OPTION> -20P;Estimated fragment value used\n”);
    printf (“<OPTION> -30P;Approximated fragment value used\n”);
    printf (“<OPTION> -31P;Benzyl approximation used\n”);
    printf (“<OPTION> -32P;Vinyl approximation used\n”);
    printf (“<OPTION> -33P;New approximated fragment value used\n”);
    printf (“<OPTION> -40P;A priori fragment value used\n”);
    printf (“<OPTION> -41P;Benzyl approx. based on a priori value\n”);
    printf (“<OPTION> -42P;Vinyl approx. based on a priori value\n”);
    printf (“<OPTION> -43P;Other a priori fragment value used\n”);
    printf (“<OPTION> -50P;Warning, hard-to-estimate structure\n”);
    printf (“<OPTION SELECTED> -51P;Very high LogP unrealistic in nature\n”);
    printf (“<OPTION> -52P;\"X/Y-C-C-Y\" excessive structure\n”);
    printf (“<OPTION> -54P;Possibly low due to hydrophilic overlap\n”);
    printf (“<OPTION> -55P;Possibly anomalous steroid\n”);
    printf (“<OPTION> -56P;Possibly anomalous alkaloid\n”);
    printf (“<OPTION> -57P;Error uncertain for charged structures\n”);
    printf (“<OPTION> -58P;(NEW) STRUCTURE CAUGHT IN SCREEN\n”);
    printf (“<OPTION> -60P;INVALID due to missing fragment value\n”);
    printf (“<OPTION> -70P;Apparent impossible structure\n”);
    printf (“<OPTION> -71P;CLOGP3 can't do disconnected structures\n”);
    printf (“<OPTION> -72P;Apparent impossible structure\n”);
    printf (“<OPTION> -80P;INVALID smiles input (fatal)\n”);
    printf (“<OPTION> -90P;NULL smiles input (fatal)\n”);
    printf (“</SELECT> Maximum ClogP Error<BR>\n”);
    printf (“</TD></TR>\n”);
    printf (“<TR><TD>\n”);
    printf (“<INPUT SIZE=5 VALUE=\"5\" NAME=\"hbd_max\">   Maximum Number of H-Bond
Donors<BR>\n”);
    printf (“<INPUT SIZE=5 VALUE=\"0\" NAME=\"hbd_min\">   Minimum Number of H-Bond
Donors<BR>\n”);
    printf (“</TD></TR>\n”);
    printf (“<TR><TD>\n”);
    printf (“<INPUT SIZE=5 VALUE=\"10\" NAME=\"hba_max\">   Maximum Number of H-Bond
Acceptors<BR>\n”);
    printf (“<INPUT SIZE=5 VALUE=\"0\" NAME=\"hba_min\">   Minimum Number of H-Bond
Acceptors<BR>\n”);
    printf (“</TD></TR>\n”);
    printf (“<TR><TD>\n”);
    printf (“<INPUT SIZE=5 VALUE=\"9\" NAME=\"rotb_max\">   Maximum Number of rotatable
bonds<BR>\n”);
    printf (“<INPUT SIZE=5 VALUE=\"0\" NAME=\"rotb_min\">   Minimum Number of rotatable
bonds<BR>\n”);
    printf (“<TR><TD>\n”);
    printf (“<TR><TD>\n”);
    printf (“<INPUT SIZE=5 VALUE=\"2\" NAME=\"crg_max\">   Maximum Number of formal
charges<BR>\n”);
    printf (“<INPUT SIZE=5 VALUE=\"0\" NAME=\"crg_min\">   Minimum Number of formal
charges<BR>\n”);
    printf (“<TR><TD>\n”);
    printf (“<INPUT SIZE=5 VALUE=\"101\" NAME=\"usdpergm\">   Maximum Cost Per Gram (in
USD)<BR>\n”);
    printf (“</TD></TR>\n”);
    printf (“</TABLE>\n”);
    printf (“<TABLE WIDTH=2000% ><TR><TD><HR></TD><TD><INPUT TYPE=\"submit\"></
TD></TR></TABLE>\n”);
    printf (“<H3>Functional Group Eliminations</H3>\n”);
    printf (“\n”);
    printf (“<TABLE BORDER=5 WIDTH=90%>\n”);
    printf (“<CAPTION><H4>These functional groups will be eliminated by default unless you select
them.</H4>\n”);
    printf (“</CAPTION>\n”);
    printf (“<TR>\n”);
    printf (“<TH>Keep?</TH><TH NOWRAP>Functional Group</TH>\n”);
    printf (“<TH>Keep?</TH><TH NOWRAP>Functional Group</TH>\n”);
    printf (“<TH>Keep?</TH><TH NOWRAP>Functional Group</TH>\n”);
    printf (“</TR>\n”);
    printf (“\n”);
    printf (“<TR>\n”);
    printf (“<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"TRUE\"
NAME=\"acid halide\"></TD><TD><FONT COLOR=\"#770000\"><FONT
COLOR=\"#770000\">Acid Halide</TD>\n”);
    printf (“<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"TRUE\" NAME=\"anhy-
dride\"></TD><TD><FONT COLOR=\"#770000\">Anhydride</TD>\n”);
    printf (“<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"TRUE\"
NAME=\"azide\"></TD><TD><FONT COLOR=\"#770000\">Azide</TD>\n”);
    printf (“</TR>\n”);
    printf (“<TR>\n”);
    printf (“<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"TRUE\" NAME=\"azo\"></
TD><TD><FONT COLOR=\"#770000\">Azo</TD>\n”);
    printf (“<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"TRUE\" NAME=\"dipep-
tide\"></TD><TD><FONT COLOR=\"#770000\">Di-Peptide</TD>\n”);
    printf (“<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"TRUE\"
NAME=\"four_halides\"></TD><TD><FONT COLOR=\"#770000\">Four Halides</TD>\n”);
    printf (“</TR>\n”);
    printf (“<TR>\n”);
    printf (“<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"TRUE\"
NAME=\"long_chain\"></TD><TD><FONT COLOR=\"#770000\">Long Chain (>7 atoms)</
TD>\n”);
    printf (“<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"TRUE\" NAME=\"non-
standardatom\"></TD><TD><FONT COLOR=\"#770000\">Non-Standard Atom</TD>\n”);
    printf (“<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"TRUE\" NAME=\"perox-
ide\"></TD><TD><FONT COLOR=\"#770000\">Peroxide</TD>\n”);
    printf (“</TR>\n”);
    printf (“<TR>\n”);
    printf (“<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"TRUE\"
NAME=\"sulfonic_acid\"></TD><TD><FONT COLOR=\"#770000\">Sulfonic Acid</TD>\n”);
    printf (“<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"TRUE\"
NAME=\"sulfonic_ester\"></TD><TD><FONT COLOR=\"#770000\">Sulfonic Ester</TD>\n”);
    printf (“<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"TRUE\"
NAME=\"two_charges\"></TD><TD><FONT COLOR=\"#770000\">>1 Formal Charge</TD>\n”);
    printf (“</TR>\n”);
    printf (“<TR>\n”);
    printf (“<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"TRUE\"
NAME=\"two_nitros\"></TD><TD><FONT COLOR=\"#770000\">>1 Nitros</TD>\n”);
    printf (“<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"TRUE\"
NAME=\"unbranched_chain\"></TD><TD><FONT COLOR=\"#770000\">Unbranched Chain (>= 4
atoms)</TD>\n”);
    printf (“<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"TRUE\" NAME=\"macrocy-
cle\"></TD><TD><FONT COLOR=\"#770000\">Macrocycles (>7 atoms)</TD>\n”);
    printf (“</TR>\n”);
    printf (“<TR>\n”);
    printf (“<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"TRUE\"
NAME=\"epoxide\"></TD><TD><FONT COLOR=\"#770000\">Epoxides</TD>\n”);
    printf (“<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"TRUE\"
NAME=\"alpha_elimination\"></TD><TD><FONT COLOR=\"#770000\"><I>Alpha Elimination
Problems</I></TD>\n”);
    printf (“<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"TRUE\"
```

```
NAME=\"beta_elimination\"></TD><TD><FONT COLOR=\"#770000\"><I>Beta Elimination Problems</I></TD>\n");
   printf ("</TR>\n");
   printf ("</TABLE>\n");
   printf ("<I>Italic indicates groups under evaluation</I>\n");
   printf ("<BR><BR>\n");
   printf ("\n");
   printf ("<TABLE BORDER=5 WIDTH=90%>\n");
   printf ("<CAPTION>\n");
   printf ("<H4>These functional groups will be kept by default unless you select them.</H4>\n");
   printf ("</CAPTION>\n");
   printf ("<TR>\n");
   printf ("<TH>Discard?</TH><TH NOWRAP>Functional Group</TH>\n");
   printf ("<TH>Discard?</TH><TH NOWRAP>Functional Group</TH>\n");
   printf ("<TH>Discard?</TH><TH NOWRAP>Functional Group</TH>\n");
   printf ("</TR>\n");
   printf ("\n");
   printf ("<TR>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"alcohol\"></TD><TD><FONT COLOR=\"#007700\">alcohol</TD>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"aldehyde\"></TD><TD><FONT COLOR=\"#007700\">aldehyde</TD>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"alkene\"></TD><TD><FONT COLOR=\"#007700\">alkene</TD>\n");
   printf ("</TR>\n");
   printf ("<TR>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"alkyl_halide\"></TD><TD><FONT COLOR=\"#007700\">alkyl_halide</TD>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"alkyne\"></TD><TD><FONT COLOR=\"#007700\">alkyne</TD>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"amide\"></TD><TD><FONT COLOR=\"#007700\">amide</TD>\n");
   printf ("</TR>\n");
   printf ("<TR>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"amine\"></TD><TD><FONT COLOR=\"#007700\">amine</TD>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"amino_acid\"></TD><TD><FONT COLOR=\"#007700\">amino_acid</TD>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"aniline\"></TD><TD><FONT COLOR=\"#007700\">aniline</TD>\n");
   printf ("</TR>\n");
   printf ("<TR>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"arene\"></TD><TD><FONT COLOR=\"#007700\">arene</TD>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"aryl_halide\"></TD><TD><FONT COLOR=\"#007700\">aryl_halide</TD>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"carbamate\"></TD><TD><FONT COLOR=\"#007700\">carbamate</TD>\n");
   printf ("</TR>\n");
   printf ("<TR>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"carboxylic_acid\"></TD><TD><FONT COLOR=\"#007700\">carboxylic_acid</TD>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"electrophile\"></TD><TD><FONT COLOR=\"#007700\"><I>electrophile</I></TD>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"ester\"></TD><TD><FONT COLOR=\"#007700\">ester</TD>\n");
   printf ("</TR>\n");
   printf ("<TR>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"ether\"></TD><TD><FONT COLOR=\"#007700\">ether</TD>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"halide\"></TD><TD><FONT COLOR=\"#007700\">halide</TD>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"hydrazine\"></TD><TD><FONT COLOR=\"#007700\">hydrazine</TD>\n");
   printf ("</TR>\n");
   printf ("<TR>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"hydroxylamine\"></TD><TD><FONT COLOR=\"#007700\">hydroxylamine</TD>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"hydrazone\"></TD><TD><FONT COLOR=\"#007700\">hydrazone</TD>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"ketone\"></TD><TD><FONT COLOR=\"#007700\">ketone</TD>\n");
   printf ("</TR>\n");
   printf ("<TR>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"nitrile\"></TD><TD><FONT COLOR=\"#007700\">nitrile</TD>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"nitro\"></TD><TD><FONT COLOR=\"#007700\">nitro</TD>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"nucleophile\"></TD><TD><FONT COLOR=\"#007700\"><I>nucleophile</I></TD>\n");
   printf ("</TR>\n");
   printf ("<TR>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"phenol\"></TD><TD><FONT COLOR=\"#007700\">phenol</TD>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"ring\"></TD><TD><FONT COLOR=\"#007700\">ring</TD>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"sulfide\"></TD><TD><FONT COLOR=\"#007700\">sulfide</TD>\n");
   printf ("</TR>\n");
   printf ("<TR>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"sulfone\"></TD><TD><FONT COLOR=\"#007700\">sulfone</TD>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"sulfoxide\"></TD><TD><FONT COLOR=\"#007700\">sulfoxide</TD>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"thioamide\"></TD><TD><FONT COLOR=\"#007700\">thioamide</TD>\n");
   printf ("</TR>\n");
   printf ("<TR>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"thiol\"></TD><TD><FONT COLOR=\"#007700\">thiol</TD>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"thiourea\"></TD><TD><FONT COLOR=\"#007700\">thiourea</TD>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"urea\"></TD><TD><FONT COLOR=\"#007700\">urea</TD>\n");
   printf ("</TR>\n");
   printf ("<TR>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"primary_amine\"></TD><TD><FONT COLOR=\"#007700\">primary amine</TD>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"secondary_amine\"></TD><TD><FONT COLOR=\"#007700\">secondary amine</TD>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"tertiary_amine\"></TD><TD><FONT COLOR=\"#007700\">tertiary amine</TD>\n");
   printf ("</TR>\n");
   printf ("<TR>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"phosphoric_acid\"></TD><TD><FONT COLOR=\"#007700\">phosphoric acid</TD>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"phosphoric_ester\"></TD><TD><FONT COLOR=\"#007700\">phosphoric ester</TD>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"phosphonic_acid\"></TD><TD><FONT COLOR=\"#007700\">phosphonic_acid</TD>\n");
   printf ("</TR>\n");
   printf ("<TR>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"phosphonic_ester\"></TD><TD><FONT COLOR=\"#007700\">phosphonic ester</TD>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"isocyanate\"></TD><TD><FONT COLOR=\"#007700\">isocyanate</TD>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"imine\"></TD><TD><FONT COLOR=\"#007700\">imine</TD>\n");
   printf ("</TR>\n");
   printf ("</TABLE>\n");
   printf ("<I>Italic indicates groups under evaluation</I>\n");
   printf ("<BR><BR>\n");
   printf ("\n");
   printf ("<TABLE WIDTH=2000%><TR><TD><HR></TD><TD><INPUT TYPE=\"submit\"></TD></TR></TABLE>\n");
###Currently there are no smarts strings for the unstable fcns###
#   printf ("<H3>Compound Stability</H3>\n");
#   printf ("<TABLE BORDER=5 WIDTH=90%>\n");
#   printf ("<CAPTION>\n");
#   printf ("<H4>Some of these groups may be unstable.  Check the groups you wish to discard.</H4>\n");
#   printf ("</CAPTION>\n");
#   printf ("<TR>\n");
#   printf ("<TH>Discard?</TH><TH NOWRAP>Functional Group</TH>\n");
#   printf ("<TH>Discard?</TH><TH NOWRAP>Functional Group</TH>\n");
#   printf ("<TH>Discard?</TH><TH NOWRAP>Functional Group</TH>\n");
#   printf ("</TR>\n");
#   printf ("\n");
#   printf ("<TR>\n");
#   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"malonic_acid\"></TD><TD><FONT COLOR=\"#000077\">malonic acid derivatives</TD>\n");
#   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"alpha_cyano_ca\"></TD><TD><FONT COLOR=\"#000077\">alpha-cyano carboxylic acid</TD>\n");
#   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"alpha_nitro_ca\"></TD><TD><FONT COLOR=\"#000077\">alpha-nitro carboxylic acid</TD>\n");
#   printf ("</TR>\n");
#   printf ("<TR>\n");
#   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"alpha_aryl_ca\"></TD><TD><FONT COLOR=\"#000077\">alpha-aryl carboxylic acid</TD>\n");
#   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"alpha_trihalo_ca\"></TD><TD><FONT COLOR=\"#000077\">alpha-trihalo carboxylic acid</TD>\n");
#   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"alpha_keto_ca\"></TD><TD><FONT COLOR=\"#000077\">alpha-keto carboxylic acid</TD>\n");
#   printf ("</TR>\n");
#   printf ("<TR>\n");
#   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"beta_keto_ca\"></TD><TD><FONT COLOR=\"#000077\">beta-keto carboxylic acid</TD>\n");
#   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"beta_keto_ca\"></TD><TD><FONT COLOR=\"#000077\">beta-keto ester</TD>\n");
#   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"alpha_beta_unsat_ca\"></TD><TD><FONT COLOR=\"#000077\">alpha,beta-unsaturated carboxylic acid</TD>\n");
#   printf ("</TR>\n");
#   printf ("<TR>\n");
#   printf (" <TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"beta_gamma_unsat_ca\"></TD><TD><FONT COLOR=\"#000077\">beta,gamma-unsaturated carboxylic acid</TD>\n");
#   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"beta_hydroxy_ca\"></TD><TD><FONT COLOR=\"#000077\">beta-hydroxy carboxylic acid</TD>\n");
#   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"beta_halo_ca\"></TD><TD><FONT COLOR=\"#000077\">beta-halo carboxylic acid</TD>\n");
#   printf ("</TR>\n");
#   printf ("<TR>\n");
#   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"alpha_amino_ket_or_ald\"></TD><TD><FONT COLOR=\"#000077\">alpha-amino ketone/aldehyde</TD>\n");
#   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"alpha_hydroxy_ket_or_ald\"></TD><TD><FONT COLOR=\"#000077\">alpha-hydroxy ketone/aldehyde</TD>\n");
#   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"alpha_beta_ket_or_ald\"></TD><TD><FONT COLOR=\"#000077\">alpha-beta ketone/aldehyde</TD>\n");
#   printf ("</TR>\n");
#   printf ("<TR>\n");
#   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"alpha_halo_ket_or_ald\"></TD><TD><FONT COLOR=\"#000077\">alpha-halo ketone/aldehyde (Cl,Br,I only)</TD>\n");
#   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"alpha_halo_hydroxyl\"></TD><TD><FONT COLOR=\"#000077\">alpha-halo hydroxyl</TD>\n");
#   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"alpha_amino_hydroxyl\"></TD><TD><FONT COLOR=\"#000077\">alpha-amino hydroxyl</TD>\n");
```

```
#   printf ("</TR>\n");
#   printf ("<TR>\n");
#   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\"
NAME=\"beta_halo_hydroxyl\"></TD><TD><FONT COLOR=\"#000077\">beta-halo hydroxyl</
TD>\n");
#   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\"
NAME=\"beta_amino_hydroxyl\"></TD><TD><FONT COLOR=\"#000077\">beta-amino
hydroxyl</TD>\n");
#   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"FALSE\" NAME=\"\"></
TD><TD><FONT COLOR=\"#000077\">alpha-halo ketone/aldehyde (Cl,Br,I only)</TD>\n");
#   printf ("</TR>\n");
#   printf ("</TABLE>\n");
#   printf ("<BR><BR>\n");
#   printf ("\n");
#   printf ("<TABLE WIDTH=200%><TR><TD><HR></TD><TD><INPUT TYPE=\"submit\"></
TD></TR></TABLE>\n");
   printf ("<H3>Suppliers</H3>\n");
   printf ("<TABLE BORDER=5>\n");
   printf ("<TR>\n");
   printf ("<TH ALIGN=center COLSPAN=2>Supplier Class</TH>\n");
   printf ("</TR>\n");
   printf ("<TR>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"radio\" NAME=\"supplier_type\"
VALUE=\"all_suppliers\"></TD>\n");
   printf ("<TD ALIGN=center><FONT COLOR=\"#007777\">All Suppliers</TD>\n");
   printf ("</TR>\n");
   printf ("<TR>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"radio\" NAME=\"supplier_type\"
VALUE=\"suppliers_here\" CHECKED></TD>\n");
   printf ("<TD ALIGN=center><FONT COLOR=\"#007777\">All Suppliers Below</TD>\n");
   printf ("</TR>\n");
   printf ("<TR>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"radio\" NAME=\"supplier_type\"
VALUE=\"selected_suppliers\"></TD>\n");
   printf ("<TD ALIGN=center><FONT COLOR=\"#007777\">Selected Suppliers Below</TD>\n");
   printf ("</TR>\n");
   printf ("<TR>\n");
   printf ("<TD></TD><TD></TD>\n");
   printf ("</TR>\n");
   printf ("<TR>\n");
   printf ("<TD></TD><TD></TD>\n");
   printf ("</TR>\n");
   printf ("<TR>\n");
   printf ("<TD></TD><TD></TD>\n");
   printf ("</TR>\n");
   printf ("<TR>\n");
   printf ("<TH>OK?</TH><TH>Company</TH>\n");
   printf ("</TR>\n");
   printf ("<TR>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"TRUE\" NAME=\"ald-
rich\"></TD>\n");
   printf ("<TD ALIGN=center><FONT COLOR=\"#770077\">Aldrich</TD>\n");
   printf ("</TR>\n");
   printf ("<TR>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"TRUE\"
NAME=\"fluka\"></TD>\n");
   printf ("<TD ALIGN=center><FONT COLOR=\"#770077\">Fluka</TD>\n");
   printf ("</TR>\n");
   printf ("<TR>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"TRUE\"
NAME=\"sigma\"></TD>\n");
   printf ("<TD ALIGN=center><FONT COLOR=\"#770077\">Sigma</TD>\n");
   printf ("</TR>\n");
   printf ("<TR>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"TRUE\" NAME=\"calbio-
chem\"></TD>\n");
   printf ("<TD ALIGN=center><FONT COLOR=\"#770077\">Calbiochem</TD>\n");
   printf ("</TR>\n");
   printf ("<TR>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"TRUE\" NAME=\"ind-
ofine\"></TD>\n");
   printf ("<TD ALIGN=center><FONT COLOR=\"#770077\">Indofine</TD>\n");
   printf ("</TR>\n");
   printf ("<TR>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"TRUE\" NAME=\"icn\"></
TD>\n");
   printf ("<TD ALIGN=center><FONT COLOR=\"#770077\">ICN</TD>\n");
   printf ("</TR>\n");
   printf ("<TR>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"TRUE\"
NAME=\"pfaltz_bauer\"></TD>\n");
   printf ("<TD ALIGN=center><FONT COLOR=\"#770077\">Pfaltz &amp; Bauer Inc.</TD>\n");
   printf ("</TR>\n");
   printf ("<TR>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"TRUE\"
NAME=\"tci_america\"></TD>\n");
   printf ("<TD ALIGN=center><FONT COLOR=\"#770077\">TCI America</TD>\n");
   printf ("</TR>\n");
   printf ("<TR>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"TRUE\" NAME=\"lan-
caster\"></TD>\n");
   printf ("<TD ALIGN=center><FONT COLOR=\"#770077\">Lancaster</TD>\n");
   printf ("</TR>\n");
   printf ("<TR>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"TRUE\"
NAME=\"acros_organics\"></TD>\n");
   printf ("<TD ALIGN=center><FONT COLOR=\"#770077\">Acros Organics</TD>\n");
   printf ("</TR>\n");
   printf ("<TR>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"TRUE\"
NAME=\"maybr_int\"></TD>\n");
   printf ("<TD ALIGN=center><FONT COLOR=\"#770077\">Maybridge-INT</TD>\n");
   printf ("</TR>\n");
   printf ("<TR>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"TRUE\" NAME=\"may-
bridge\"></TD>\n");
   printf ("<TD ALIGN=center><FONT COLOR=\"#770077\">Maybridge</TD>\n");
   printf ("</TR>\n");
   printf ("<TR>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"TRUE\"
NAME=\"salor\"></TD>\n");
   printf ("<TD ALIGN=center><FONT COLOR=\"#770077\">Salor</TD>\n");
   printf ("</TR>\n");
   printf ("<TR>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"checkbox\" VALUE=\"TRUE\"
NAME=\"trans_world\"></TD>\n");
   printf ("<TD ALIGN=center><FONT COLOR=\"#770077\">Trans World</TD>\n");
   printf ("</TR>\n");
   printf ("</TABLE>\n");
   printf ("<BR><BR>\n");
   printf ("\n");
   printf ("<TABLE WIDTH=200%><TR><TD><HR></TD><TD><INPUT TYPE=\"submit\"></
TD></TR></TABLE>\n");
   printf ("<TABLE WIDTH = 30%>\n");
   printf ("<H3>Select Output Format</H3>\n");
   printf ("<TR>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"radio\" NAME=\"output_type\" VALUE=\"tdt\"
CHECKED></TD>\n");
   printf ("<TD>SMILES</TD>\n");
   printf ("</TR>\n");
   printf ("<TR>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"radio\" NAME=\"output_type\" VALUE=\"lst\"></
TD>\n");
   printf ("<TD NOWRAP>MDL .lst</TD>\n");
   printf ("</TR>\n");
   printf ("<TR>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"radio\" NAME=\"output_type\"
VALUE=\"graphic\"></TD>\n");
   printf ("<TD NOWRAP>Inline Graphic (takes longer, not recommended for lists >100)</TD>\n");
   printf ("</TR>\n");
   printf ("<TR>\n");
   printf ("<TD ALIGN=center><INPUT TYPE=\"radio\" NAME=\"output_type\"
VALUE=\"library\"></TD>\n");
   printf ("<TD NOWRAP>Reaction Library</TD>\n");
   printf ("</TR>\n");
   printf ("</TABLE>\n");
   printf ("\n");
   printf ("<TABLE WIDTH=200%><TR><TD><HR></TD><TD><INPUT TYPE=\"submit\"></
TD></TR></TABLE>\n");
   printf ("<H3>Submit Search</H3>\n");
   printf ("<INPUT TYPE=\"submit\"><INPUT TYPE=\"reset\">\n");
   printf ("<P>When form is complete submit it for a database search.  The search may take a couple
minutes.\n");
   printf ("</FORM>\n");
   printf ("<HR>\n");
   printf ("<BR>Created by: Geoff Skillman\n");
   printf ("<A HREF=mailto:skillman\@francisco.ucsf.edu>\n");
   printf ("<EM>skillman\@francisco.ucsf.edu</A></EM>\n");
   printf (", <A TARGET=_TOP HREF=http://www.cmpharm.ucsf.edu/kuntz>Kuntz Group,
UCSF\n");
   printf ("</BR>\n");
   printf ("</CENTER>\n");
   printf ("</BODY>\n");
   printf ("</HTML>\n");

}
```

Functional Group Smiles Codes:

Functional Group	SMILES
aminoacid	NCC=O
dipeptide	O=CCNC(=O)CN
two_nitros	N(=O)[O-].N(=O)[O-]
malonic	O=CCC=O
urea	NC(=O)N
alcohol	CO
thiol	S
alkene	C=C
alkyl	C
alkyne	C#C
lactam	O=CN
amide	O=CN
anhydride	O=COC=O
aniline	N
aniline_unsubstituted	N
azide	N
triazine	N=NN
azo	N=N
carbamate	NC(=O)O
carbamic_acid	NC(=O)O
carbonate	OC(=O)O
thiourea	NC(=S)N
carbonyl	O=C
thiocarbonyl	S=C
carboxylic_acid	O=CO
acid_halide	O=C
acid_chloride	O=CCl

Functional Group	SMILES
lactone	O=CO
ester	O=CO
aldehyde	O=C
ketone	O=C
sulfonic_acid	S(=O)(=O)O
sulfonic_ester	S(=O)(=O)O*
phosphonic_acid	P(=O)(=O)O
phosphonic_ester	P(=O)(=O)O*
epoxide	C1OC1
hydrazine	NN
hydrozone	NN=C
isocyanate	O=C=N
isothiocyanate	S=C=N
nitrile	N#C
nitro	N(=O)[O-]
peroxide	OO
phenol	O
amine	N
primary_amine	N
secondary_amine	N
tertiary_amine	N
sulfide	S
sulfone	S(=O)=O
sulfoxide	S=O
disulfide	SS
alkyl_halide	C
ether	O
thioether	S

Functional Group	SMILES
acetal	OCO
ketal	OCO
hemiacetal	OCO
hemiketal	OCO
sulfonamide	S(=O)(=O)N
sulfonyl_halide	S(=O)(=O)
imino	N=C
oxime	ON=C
dithioacetal	SCS
oxalyl	O=CC=O
enamine	C=CN
enol_ether	C=CO
hydroxylamine	NO

Functional Group Rooted SMARTS Codes:

Functional Group	Rooted SMARTS
amino_acid	[N][C;!$(C=*)][C;$(C=O);$(C[!#6])]
dipeptide	O=CCNC(=O)CN
two_charges	[$charge].[$charge]
two_nitros	[$nitro].[$nitro]
unbranched_chain	[R0;D2][R0;D2][R0;D2][R0;D2]
charge	[$acid,$base]
acid	[*&$(*=*)&$(*[$hydroxyl]),$malonic]
base	[n,N&!D3&!$(N*=[!#6])]
malonic	[C;H1,H2;$(C([$Ccarbonyl])[$Ccarbonyl])]
four_halides	[$halide].[$halide].[$halide].[$halide]
long_chain	[A;R0][A;R0][A;R0][A;R0][A;R0][A;R0][A;R0][A;R0]

Functional Group	Rooted SMARTS
macrocycle	[r8,r9,r10,r11,r12,r13,r14,r15,r16,r17,r18]
nonstandardatom	[!#1;!#2;!#3;!#5;!#6;!#7;!#8;!#9;!#11;!#12;!#15;!#16;!#17;!#19;!#20;!#35;!#53]
nucleophile	[$alcohol,$primary_amine,$secondary_amine,$aniline,$phenol,$azide,$hydrazine,$hydroxylamine,$peroxide,$thiol,$oxime]
alkyl	[$Calkyl]
combi_any	[$combi_fcn,$combi_linker]
combi_linker	[$amide,$secondary_amine,$sulfonamide,$urea,$ester,$ether,$tertiary_amine,$carbamate,$imino,$hydrazone,$thioether,$thioamide,$thiourea,$thiocarbamate,$thioester]
combi_fcn	[$ketone,$aldehyde,$primary_amine,$secondary_amine,$amide,$alkylating_agent,$aniline,$alcohol,$phenol,$thiol,$isocyanate,$isothiocyanate,$carboxylic_acid,$acid_halide,$hydrazine,$aryl_mono_BrI]
mono_alkene	[$alkene;$!($alkene.$alkene)]
mono_alkyne	[$alkyne;$!($alkyne.$alkyne)]
aryl_mono_BrI	[c;$(c[Br,I])]
urea	[$Nurea]
alcohol	[$Oalcohol]
thiol	[$Sthiol]
alkene	[$Calkene]
alkyne	[$Calkyne]
lactam	[$Clactam]
amide	[$Camide]
thioamide	[$Cthioamide]
anhydride	[$Canhydride]
aniline	[$pseudo_amine;$(N[$aryl]);!$(N~[!#6])]
aniline_unsubstituted	[$pseudo_amine;D1;$(N[$aryl]);!$(N~[!#6])]
azide	[$Nazide]
triazine	[$N1triazine,$N12triazine]
azo	[$Nazo]

Functional Group	Rooted SMARTS
thiocarbamate	[$Othiocarbamate,$Nthiocarbamate]
carbamate	[$Ocarbamate,$Ncarbamate]
carbamic_acid	[$Ccarbamic_acid]
carbonate	[$Ocarbonate]
thiourea	[$Nthiourea]
carbonyl	[$Ccarbonyl]
thiocarbonyl	[$Cthiocarbonyl]
carboxylic_acid	[$Ccarboxylic_acid]
acid_halide	[$Cacid_halide]
acid_chloride	[$Cacid_chloride]
thioester	[$Cthioester]
ester	[$Cester]
lactone	[$Clactone]
aldehyde	[$Caldehyde]
ketone	[$Cketone]
sulfonic_acid	[$Ssulfonic_acid]
sulfonic_ester	[$Ssulfonic_ester]
phosphonic_acid	[$Pphosphonic_acid]
phosphonic_ester	[$Pphosphonic_ester]
phosphoric_acid	[$Pphosphoric_acid]
phosphoric_ester	[$Pphosphoric_ester]
epoxide	[$Cepoxide]
hydrazine	[$Nhydrazine]
hydrazone	[$Nhydrazone]
isocyanate	[$Nisocyanate]
isothiocyanate	[$Nisothiocyanate]
nitrile	[$Cnitrile]
nitro	[$Nnitro]

Functional Group	Rooted SMARTS
peroxide	[$Operoxide]
phenol	[$Ophenol]
primary_amine	[$Nprimary_amine]
secondary_amine	[$Nsecondary_amine]
tertiary_amine	[$Ntertiary_amine]
sulfide	[$Ssulfide]
sulfone	[$Ssulfone]
sulfoxide	[$Ssulfoxide]
disulfide	[$Sdisulfide]
alkylating_agent	[$Xalkylating_agent]
alkyl_halide	[$Xalkyl_halide]
aryl_halide	[$Xaryl_halide]
ether	[$Oether]
thioether	[$Sthioether]
acetal	[$Cacetal]
ketal	[$Cketal]
hemiacetal	[$Chemiacetal]
hemiketal	[$Chemiketal]
sulfonamide	[$Ssulfonamide]
sulfonyl_halide	[$Ssulfonyl_halide]
imino	[$Cimino]
oxime	[$Coxime]
dithioacetal	[$Cdithioacetal]
organometallic	[$Corganometallic]
oxalyl	[$Coxalyl]
enamine	[$Cenamine]
enol_ether	[$Cenol_ether]
Sdisulfide	[S;$(S([$Calkyl])S[$Calkyl])]

Functional Group	Rooted SMARTS
Xalkylating_agent	[$lg_halide;$(*[$Calkyl])]
Xalkyl_halide	[$halide;$(*[$Calkyl])]
Xaryl_halide	[$halide;$(*[$aryl])]
Sthioether	[S;$(S([$Calkyl])[$Calkyl])]
Oacetal	[O;$(O[$Cacetal])]
Cacetal	[C;H1,H2;$(C(O[$Calkyl])O[$Calkyl])]
Oketal	[O;$(O[$Cketal])]
Cketal	[C;H0;$(C(O[$Calkyl])O[$Calkyl])]
OEhemiacetal	[O;$Oether;$(O[$Chemiacetal])]
OHhemiketal	[O;$hydroxyl;$(O[$Chemiketal])]
OEhemiacetal	[O;$Oether;$(O[$Chemiacetal])]
OHhemiketal	[O;$hydroxyl;$(O[$Chemiketal])]
Chemiacetal	[C;H1,H2;$(C(O[$Calkyl])[$hydroxyl])]
Chemiketal	[C;H0;$(C(O[$Calkyl])[$hydroxyl])]
Ssulfonamide	[S;$(S(=O)(=O)N)]
Ssulfonyl_halide	[S;$(S(=O)(=O)[$halide])]
Nimino	[N;$(N=[$Cimino])]
Cimino	[C;$(C=[N;!$(N~[$hetatm])])]
Ooxime	[O;$(O[$Noxime])]
Noxime	[N;$(N=[$Coxime])]
Coxime	[C;$(C=N[$hydroxyl])]
Sdithioacetal	[S;$(S[$Cdithioacetal])]
Cdithioacetal	[C;$(C1SCCCS1)]
Corganometallic	[C;$(CB),$(C[Mg][$halide]),$(C[Li]),$(C[Cu][Li]),$(C([Ag])#C)]
Ooxalyl	[O;$(O=[$Coxalyl])]
Coxalyl	[$Ccarbonyl;$(C[$Ccarbonyl])]
Cenamine	[C;$(C=C[N;!$Nnitro])]

Functional Group	Rooted SMARTS
Oenol_ether	[O;$(OC=[$Cenol_ether])]
Cenol_ether	[C;$(C=C[$Oether])]
Oether	[O;$(O([$Cstd])[$Cstd])]
Nurea	[N;$(N[$Curea])]
Curea	[$Ccarbonyl;$(C(=O)(N)N)]
Oalcohol	[$hydroxyl;$(O[C;!$(C=[!#6])])]
Sthiol	[$mercapto;$(S[#6;!$(C=[!#6])])]
Nlactam	[Namide;R]
Clactam	[Camide;R]
Nthioamide	[N;$(N[$Cthioamide])]
Cthioamide	[$Cthiocarbonyl;$(CN);!$(C(N)(=S)[!#6])]
Namide	[N;$(N[$Camide])]
Camide	[$Ccarbonyl;$(CN);!$(C(N)(=O)[!#6])]
Canhydride	[$Ccarbonyl;$(CO[$Ccarbonyl])]
Nazide	[N;$(N=[N+]=[N-])]
N1triazine	[N;$(N=N-N);D2]
N12triazine	[N;$(N-N=N);D2,D3]
Nazo	[N;D2;$(N=[N;D2]);!$(N[$hetatm]);!$(N=N[$hetatm])]
Ccarbamic_acid	[$Ccarbamate;$(C[$hydroxyl])]
Othiocarbamate	[O;$(O[$Cthiocarbamate])]
Nthiocarbamate	[N;$(N[$Cthiocarbamate])]
Cthiocarbamate	[$Cthiocarbonyl;$(C(=S)(O)N)]
Ocarbamate	[O;$(O[$Ccarbamate])]
Ncarbamate	[N;$(N[$Ccarbamate])]
Ccarbamate	[$Ccarbonyl;$(C(=O)(O)N)]
Ocarbonate	[O;$(O[$Ccarbonate])]
Ccarbonate	[$Ccarbonyl;$(C(=O)(O)O)]
Nthiourea	[N;$(N[$Cthiourea])]

Functional Group	Rooted SMARTS
Cthiourea	[$Cthiocarbonyl;$(C(=S)(N)N)]
Ocarboxylic_acid	[$hydroxyl;$(O[$Ccarboxylic_acid])]
Ccarboxylic_acid	[$Ccarbonyl;$(C[$hydroxyl]);$(C[#6,#1])]
Cacid_chloride	[$Cacid_halide;$(CCl)]
Cacid_halide	[$Ccarbonyl;$(C[$halide]);$(C[#6,#1])]
Clactone	[$Cester;R]
Cthioester	[$Cthiocarbonyl;$(C(=S)O[#6]);$(C[#6,#1])]
Cester	[$Ccarbonyl;$(C(=O)O[#6]);$(C[#6,#1])]
Caldehyde	[$Ccarbonyl;$([H1,H2]);!$(C-[$hetatm])]
Cketone	[$Ccarbonyl;$(C([#6])[#6])]
Ssulfonic_acid	[S;$(S(=O)(=O)[$hydroxyl])]
Ssulfonic_ester	[S;$(S(=O)(=O)O*)]
Pphosphonic_acid	[P;$(P(=O)(=O)[$hydroxyl])]
Pphosphonic_ester	[P;$(P(=O)(=O)O*)]
Pphosphoric_acid	[P;$(P(=O)(O)[$hydroxyl])]
Pphosphoric_ester	[P;$(P(=O)(O)O*)]
Oepoxide	[O;$(O([$Cepoxide])[$Cepoxide])]
Cepoxide	[C;$(C1CO1)]
Nhydrazine	[N;$(N-[N;D1]);!$(N=C)]
hydroxylamine	[$pseudo_amine;$(N[$hydroxyl]);!$(N=*)]
Nhydrazone	[N;$(N[N;D2]=C)]
Cisocyanate	[C;$(C=[$Nisocyanate])]
Nisocyanate	[N;$(N(=C=O)*)]
Cisothiocyanate	[C;$(C=[$Nisothiocyanate])]
Nisothiocyanate	[N;$(N(=C=S)*)]
Nnitrile	[N;$(N#[$Cnitrile])]
Cnitrile	[C;$(C#[N;D1])]
Nnitro	[N;+0,+1;$(N(=O)~[O;H0;-0,-1])]

Functional Group	Rooted SMARTS
Operoxide	[O;$(O[$hydroxyl])]
Ophenol	[$hydroxyl;$(Oc)]
Nprimary_amine	[$amine;D1]
Nsecondary_amine	[$amine;D2]
Ntertiary_amine	[$amine;D3]
ring	[R]
amine	[N;!$(N*=[!#6]);!$(N~[!#6]);!$(Na);!$(N#C);!$(N=C)]
pseudo_amine	[N;!$(N*=[!#6])]
Ssulfide	[S;D2;$(S([#6])[#6])]
Ssulfone	[S;$(S(=O)(=O)([#6])[#6])]
Ssulfoxide	[S;D3;$(S(=O)([#6])[#6])]
Ccarbonyl	[C;$(C=[$Ocarbonyl])]
Ocarbonyl	[O;D1;$(O=C)]
Cthiocarbonyl	[C;$(C=[$Scarbonyl])]
Scarbonyl	[S;D1;$(S=C)]
hetatm	[!#6;$([N,O,S,F,Cl,Br,I,P])]
halide	[!#6;$([F,Cl,Br,I])]
lg_halide	[!#6;$([Br,I])]
mercapto	[S;$([H1&-0,H0&-1])]
hydroxyl	[O;$([H1&-0,H0&-1])]
Cstd	[#6;!$(*=[!#6])]
Calkyl	[C;!$(C=[!#6])]
Calkene	[C;$(C=C)]
Calkyne	[C;$(C#C)]
Caryl	[#6;a]
arene	[c]
aryl	[a]

Appendix 3: Diversify Program

Abstract

The complete code in C for Diversify is included below. Diversify was discussed above in Chapter 4. It includes extensive use of Daylight Chemical Information System's toolkit C libraries. Although I must admit to writing the entirety of this code. It would not have been possible without the extensive assistance of Connie Oshiro and Donna Hendrix, to whom I am grateful. I've included both the datastructures (in header files) and the subroutines. The code does include some notes for future development.

Included Files

Datastructure Files:

mol.h
trans.h
div.h
rep.h
sep.h
con.h
parsesmiles.h
io.h
screen.h
select.h
vbind.h
debug.h

Code Files:

div.c
rep.c
sep.c
con.c
parsesmiles.c
io.c
screen.c
select.c
vbind.c
debug.c

```
Copyright 1999, A.G. Skillman, I.D. Kuntz, Regents of the University of California
 * rxn.h
 *
 * this is the header file for the syn rxn routines.
 *
 ***********************************************************************/


/***definitions***/


/***structs/typedefs   ***/


/***function prototypes***/

voidconnect(molequeue *Reactant, transform *Transform, molequeue **Lastptr),
/***********************************************************************
 * debug.h
 *
 * this is the header file for the debuging routines for use with daylight.
 *
 ***********************************************************************/

/***definitions***/

#define TRUE1
#define FALSE0


/***structs/typedefs   ***/


/***function prototypes***/

void debug_sequence(dt_Handle seq);
void debug_atoms(dt_Handle atoms);
void debug_bonds(dt_Handle bonds);
void debug_pattern(dt_Handle pattern);
void debug_bond(dt_Handle bond);
void debug_path(dt_Handle path);
void debug_pathset(dt_Handle pathset);
void debug_molecule(dt_Handle mol),
void debug_atom(dt_Handle atom),
/***********************************************************************
 * diversity.h
 *
 * this is the header file for the main routine for the program diversity.
 *
 ***********************************************************************/


/***includes   ***/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <ctype.h>

#include "dt_smiles.h"
#include "dt_smarts.h"

#include "mol.h"
#include "trans.h"

#include "io.h"
#include "rep.h"
#include "sep.h"
#include "con.h"
#include "screen.h"
#include "vbind.h"
#include "debug.h"

/***definitions***/

#define VERSION 1.10

#define TRUE 1
#define FALSE 0
#define PSTOP 750.0
#define PSBOTTOM 50.0
#define PSLEFT 50.0
#define PSRIGHT 550.0

/***structs/typedefs   ***/

typedef struct{
   int    membercount;
   int    zerocount;
   int    member[MAXMOLEQUEUE+100];
   int    parent[MAXMOLEQUEUE+100];
   int    nochildren[MAXMOLEQUEUE+100].
   int    nozero[MAXMOLEQUEUE+100].
}level;

typedef struct{
   int    numlevels.
   level   level[MAXDEPTH+1];
}dendogram;

/***function prototypes***/

dendogram *fill_dendogram(molequeue *Add, dendogram *Tree).
int write_dendogram(dendogram *Tree);
/***********************************************************************
 * getconf.h
 *
```

```
 * adapted from getmatrix.h
 *
 * this is the header file for the main routine for the program getmatrix.
 *
 ***********************************************************************/


/***includes   ***/


/***definitions***/

#define FALSE 0
#define TRUE 1
#define MAXSMILES 200
#define MAXBUF 200

/***structs/typedefs   ***/

typedef struct dg_minimizer{
   void   *minimizer;
   float   convergence;
   int   eval;
   struct dg_minimizer   *Next;
}dg_minimizer;

typedef struct dg_method{
   void   *method;
   char   name[MAXBUF];
   int   nconfs;
   int   trials;
   float   mxdv;
   float   mxvv;
   char   rulefile[MAXBUF];
   int   hflag;
   int   bump14;
   int   debug;
   int   savebounds;
   dg_minimizer   *Min;
}dg_method;

typedef struct volatoms{
   int   iatom[4];
}volatoms;

typedef struct volbounds{
   float   minvol;
   float   maxvol;
}volbounds;

typedef struct iofiles{
   char   infilename[MAXBUF];
   char   method1name[MAXBUF];
   char   method2name[MAXBUF];
}iofiles;


/***function prototypes***/

dt_Handle getconf(*molequeue *Molecule);
int getin(FILE *Infile, dg_method *Method, iofiles *Filenames);
int openmethod(dg_method *Method);
int getmol(char *Smiles, mol *Molecule);
int fixH(mol *Molecule, int molct);
int getstruct(mol *Molecule, dg_method *Method);
/***********************************************************************
 * io.h
 *
 * this is the header file for the div io routines.
 *
 ***********************************************************************/

/***definitions***/

#define MAXBUF 2000
#define MAXKEY 20
#define TRUE 1
#define FALSE 0

/***structs/typedefs   ***/

typedef struct{
   char   input[MAXBUF];
   char   vbind[MAXBUF];
   char   rxn_levels[MAXBUF];
   char   smiles[MAXBUF];
   char   transform[MAXBUF];
   char   library[MAXBUF];
   char   output[MAXBUF];
} filenames;

/***function prototypes***/

molequeue *readinmols(FILE *Smilesfile, molequeue **Lastptr);
molequeue *retire(FILE *Outfile, molequeue *Done);
library *readlibs(FILE *Libraryfile, library *Libraries, int *libtot, transform *Transforms, int transtot);
int readtrans(FILE *Reactionfile, transform *Transforms);
int getsmart(transform *Transform);
transform *parserep(transform *Transform);
transform *parsesep(transform *Transform);
transform *parsecon(transform *Transform);
molequeue *re_read(FILE *Outread, molequeue *Active);
molequeue *tempwrite(FILE *Outfile, molequeue *Done);
int get_input(FILE *Infile, filenames *Filenames, molparms *Molecule_parameters, transparms
*Transform_parameters);
int **readlevels(FILE *Rxnlevelsfile, transform *Transforms, int transcount, int *rxnlevels, int
maxdepth);
```

```
/*********************************************************************
 * mol.h
 *
 * this is the header file which contains molecular structures.
 *
 *********************************************************************/


/***includes    ***/

#include "dt_smiles.h"
#include "dt_finger.h"


/***definitions***/

#define MAXSMILES200
#define MAXCHILDREN5000
#defineMAXMOLEQUEUE2000000
#define MAXDEPTH13
#defineMAXMOLWT1000
#define MINMOLWT50
#define MAXNAME100
#define MAXTAG2000


/***structs/typedefs    ***/

typedef struct  molequeue{
   dt_Handlemolecule;
   dt_Handlecont;
   charsmiles[MAXSMILES];
   charexternalid[MAXTAG];
   charparent[MAXSMILES+8];
   int identical;
   chartransform[MAXNAME];
   intchildren[MAXCHILDREN];
   struct  molequeue*Next;
   struct  molequeue*Previous;
   intdepth;
   floattanimoto;
   intsynno;
   intnochildren;
} molequeue;

typedef struct{
   int   maxdepth;
   int   maxmolwt;
   int   minmolwt;
   int   maxmolecule;
} molparms;

/***function prototypes***/

/*********************************************************************
 * parsesmiles.h
 *
 * this is the header file for the div transform-smiles parsing routines
 *
 *********************************************************************/

/***definitions***/


#define TRUE 1
#define FALSE 0
#define DIV_CHI_FORWARD 3
#define DIV_CHI_BACKWARD 4
#define DIV_CHI_UP 5
#define DIV_CHI_DOWN 6
#define DIV_BCLS_IMPLICIT 0
#define DIV_BCLS_EXPLICIT 1
#define DIV_BCLS_RING 2
#define DIV_BCLS_EXTERNAL 3
#define DIV_CHI_AL 5
#define DIV2DX_AL(X) ((DX_CHI_TH))

#defineFIRST_POSITION(JOINT)   ((((JOINT) >> 16) & 0xffff) - 1)
#define SECOND_POSITION(JOINT)   ((JOINT) & 0xffff)
#define JOIN_POSITIONS(FIRST,SECOND)   ((((FIRST) + 1) << 16) | (SECOND))

#define FOR_TO_UPDOWN(BONDPOS, ATOMPOS)   (((BONDPOS) < (ATOMPOS)) ?
(DIV_CHI_DOWN) : (DIV_CHI_UP))
#define BACK_TO_UPDOWN(BONDPOS, ATOMPOS)  (((BONDPOS) < (ATOMPOS)) ?
(DIV_CHI_UP) : (DIV_CHI_DOWN))
#defineFORBACK_TO_UPDOWN(CHI_FB, BONDPOS, ATOMPOS)(((CHI_FB) ==
(DIV_CHI_FORWARD)) ? (FOR_TO_UPDOWN(BONDPOS, ATOMPOS))
(BACK_TO_UPDOWN(BONDPOS, ATOMPOS)))

/***structs/typedefs    ***/


/***function prototypes***/

int parsesmiles(char *string, atoms *newatoms, int *atomnumber, bonds *fragbonds, int *bondnumber);
int parseatoms(char *string, atoms *atoms);
int atno(char *id);
int parsebonds(char *string, atoms *atoms, int maxatoms, bonds *bonds);
int makebond(int bondclass, int bondcount, atoms *atoms, int maxatoms, bonds *bonds, char* string,
int position, int bondtype, int chival);
int preatom(atoms *atomlist, int maxatoms, int current, int branchlevel);
int postatom(atoms *atomlist, int maxatoms, int current);
int setchiral(char *string, atoms *atoms, int maxatoms, bonds *bonds, int maxbonds);
int chibondc(int value);
int otheratom(int iatom, int ibond, atoms *atoms, bonds *bonds);
int bondposition(int ibond, int iatom, atoms *atoms, bonds *bonds);

/*********************************************************************
```

```
 * rxn.h
 *
 * this is the header file for the syn rxn routines.
 *
 *********************************************************************/


/***definitions***/

#define MAXEXTERNAL5

/***structs/typedefs    ***/


/***function prototypes***/

voidreplace(molequeue *Reactant, transform *Transform, molequeue **Lastptr);
inthcount(dt_Handle *Atom);
intscreen(molequeue *Reactant, transform *Transform);
/*********************************************************************
 * debug.h
 *
 * this is the header file for the debuging routines for use with daylight.
 *
 *********************************************************************/

/***definitions***/


/***structs/typedefs    ***/


/***function prototypes***/

int get_molwt(dt_Handle mol);
int tanimoto(molequeue *Testmol, float min, float max);
int prune(molequeue *Testmol,molparms *Parameters);
/*********************************************************************
 * select.h
 *
 * this is the header file for the selection routines for figuring out
 * where a reaction should take place on a molecule.
 *
 *********************************************************************/

/***definitions***/

#define DIV_RANK_ALL 0
#define DIV_RANK_THRESHOLD 1
#define DIV_ONE 2
#define DIV_ONE_THRESHOLD 3

#define DIV_STERIC_1A 1
#define DIV_STERIC_1a 1
#define DIV_STERIC_2A 1
#define DIV_STERIC_2a 1
#define DIV_STERIC_NEOPENTENE 2
#define DIV_STERIC_DBL 1
#define DIV_STERIC_ALPHAAROMATIC 2
#define DIV_STERIC_FLUORINE 9
#define DIV_STERIC_FLUORINE_SCR 1

/***structs/typedefs    ***/


/***function prototypes***/

dt_Handle *screen(molequeue *Reactant, transform *Transform);
dt_Handle *steric(dt_Handle *pathset, dt_Handle *mol, int threshold, int rank);
/*********************************************************************
 * sep.h
 *
 * this is the header file for the div separation transfrom routines.
 *
 *********************************************************************/


/***definitions***/


/***structs/typedefs    ***/


/***function prototypes***/

voidseparate(molequeue *Reactant, transform *Transform, molequeue **Lastptr);
molequeue *subproduct(molequeue *Product, transform *Transform, int count);
intdeletefragment(dt_Handle bond, dt_Handle *atom, dt_Handle *delete, int *incount);
/*********************************************************************
 * trans.h
 *
 * this is the header file for the reaction structures.
 *
 *********************************************************************/


/***includes    ***/

#include "dt_smiles.h"
#include "dt_smarts.h"

/***definitions***/

#define MAXTRANS 100
#define MAXTEMPLATE 200
#define MAXLIBRARY 50
#define MAXNAME 100
#define MAXCHIRAL 8
```

```
#define MAXIBOND 6
#define MAXCONNECTIONS 6
#define MAXATTACHATOMS 10
#define MAXVBINDINGS 100

#define DIV_IMP_HYDROGEN 999

extern dt_Handle vbinding[MAXVBINDINGS];

/***structs/typedefs   ***/

typedef struct {
   charsymbol;
   intatomicno,
   inthydrogens,
   intformalcharge,
   intaromatic;
   intibond[MAXIBOND];
   charbranchlevel,
   intposition,
   unsigned char   inbracket,
   unsigned char   bracketsize;
   intchival;
   intchiseq[MAXCHIRAL].
} atoms,

typedef struct {
   intiatom1,
   intiatom2,
   intbondtype;
   charbranchlevel;
   intposition,
   intchival;
   intchiseq[2];
} bonds,

typedef struct{
   charname[MAXNAME],
   intsize;
   dt_Handle*Fragments;
   char**Externalids;
   dt_Handle**Attachatoms;
} library,

typedef struct{
   charproductset[MAXTEMPLATE],
   atomsproductatom[MAXTEMPLATE];
   intatomnumber,
   bondsproductbond[MAXTEMPLATE];
   intbondnumber,
} replacement;

typedef struct{
   charproductset[2][MAXTEMPLATE],
   atomsproductatom[2][MAXTEMPLATE];
   intatomnumber[2];
   bondsproductbond[2][MAXTEMPLATE],
   intbondnumber[2],
} separation;

typedef struct{
   charlibraryname[MAXNAME];
   library*Library,
   charproductset[MAXTEMPLATE],
   atomsproductatom[MAXTEMPLATE],
   intatomnumber,
   bondsproductbond[MAXTEMPLATE];
   intbondnumber,
}connection,

typedef struct{
   char       name[MAXNAME],

   char       inclusion[MAXTEMPLATE];
   dt_Handlescreenin,
   intpatternatoms,

   char       exclusion[MAXTEMPLATE],
   dt_Handlescreenout;

   char       marked[MAXTEMPLATE],
   dt_Handle   markscreen;

   char       flag,
   union{
replacement rep;
connection  con;
separation  sep;
   } type,
} transform,

typedef struct{
   int nothing;
}transparms,

/***function prototypes***/

/**********************************************************************
 * vbind h
 *
 * this is the header file for the vbinding initialization routines
 *
 **********************************************************************/

/***includes***/

#include "dt_smarts h"
#include "dt_smiles.h"
```

```
/***definitions***/

#define TRUE 1
#define FALSE 0
#define MAXBUF 2000

/***structs/typedefs   ***/


/***function prototypes***/

int vbindinit(FILE *Bindingfile);




#include "div.h"

extern int syncount;

main(int argc, char *argv[])
{
   char   buf[MAXBUF];
   FILE   *Infile,
   *Vbindingfile,
   *Smilesfile,
   *Transformfile,
   *Rxnlevelsfile,
   *Libraryfile,
   *Outfile,
   *Tempout,
   *Tempin;
   int   rxnscheme = FALSE,
   rxnlevels = FALSE,
   **levels,
   count,
   count1,
   mw,
   index,
   index1,
   end,
   idcount = 0,
   uniquecount = 0,
   uniquelist[MAXSMILES*('z'-'#')] = {-1},
   transcount,
   vbindcount,
   libtot,
   transtot,
   goodmol,
   molequeue*First,
*Retire,
*Active,
*Last,
*Key,
*Going,
*Coming,
*Tofree,
**Lastptr = &Last;
/* dendogrammaintree;*/
   transformtransforms[MAXTRANS];
   library*Libraries;
   molparms*Molecule_parameters;
   transparms*Transform_parameters;
   filenames*Filenames;
   time_tStarttime,
Endtime;


/***initialize time   ***/

   Starttime = time(NULL),

/***initialize structures***/

   if(NULL == (Molecule_parameters = (molparms *)malloc(sizeof(molparms)))){
fprintf(stderr, "Unable to malloc Molecule_parameters in div.c.\n");
exit(1);
   }
   if(NULL == (Transform_parameters = (transparms *)malloc(sizeof(transparms)))){
fprintf(stderr, "Unable to malloc Transform_parameters in div.c.\n");
exit(1);
   }
   if(NULL == (Filenames = (filenames *)malloc(sizeof(filenames)))){
fprintf(stderr, "Unable to malloc Filenames in div.c.\n");
exit(1);
   }

/***   open input file for reading   ***/

   if (argc > 3){
      fprintf(stderr,"usage div <inputfile> -sl\n");
fprintf(stderr,"-s: reaction scheme rather than exhaustive application\n");
fprintf(stderr,"-l: reaction levels\n"),
      exit(1);
   }
   if ((argc == 3)&&(0 == strcmp(argv[2], "-s"))){
fprintf(stderr, "Using rxn scheme format.\n");
rxnscheme = TRUE;
   }
   if ((argc == 3)&&(0 == strcmp(argv[2], "-l"))){
fprintf(stderr, "Using rxn levels format.\n");
rxnlevels = TRUE,
   }
   if (argc == 2){
if ((argv[1][0] == '-') && (tolower(argv[1][1]) == 'h')){
   fprintf(stderr,"usage div <smilesfile> -sl\n");
```

```
    fprintf(stderr,"-s: reaction scheme rather than exhaustive application\n");
    fprintf(stderr,"-l: reaction level control for scheme.\n");
    exit(1);
}
   }
   if (argc < 2){
      fprintf(stdout,"Input file? >>");
      fflush(stdout);
      fgets(buf,MAXBUF,stdin);
      buf[strlen(buf)-1]='\0';
strcpy(Filenames->input, buf);
      if(NULL == (Infile = fopen(buf,"r"))){
         fprintf(stderr, "Unable to open input file:  %s\n",buf);
         exit(1);
      }
   }
   else{
strcpy(Filenames->input, argv[1]);
      if(NULL == (Infile = fopen(argv[1],"r"))){
         fprintf(stderr, "Unable to open input file:  %s\n",argv[1]);
         exit(1);
      }
   }

/***read input file   ***/

   if(FALSE == get_input(Infile, Filenames, Molecule_parameters, Transform_parameters)){
fprintf(stderr, "Error reading input file \n");
exit(1);
   }

/*** get output filename and open ***/

   if (!strcmp(Filenames->output, "")){
      fprintf(stdout, "Output file?>>");
fflush(stdout);
fgets(buf, MAXBUF, stdin);
buf[strlen(buf)-1]='\0';
strcpy(Filenames->output, buf);
if(NULL == (Outfile = fopen(buf, "w"))){
   fprintf(stderr, "Unable to open outputfile:  %s\n", buf);
   exit(1);
}
   }
   else{
if(NULL == (Outfile = fopen(Filenames->output,"w"))){
   fprintf(stderr, "Unable to open outputfile:  %s\n", Filenames->output);
   exit(1);
}
   }

/*** get library filename and open for input***/

   if (!strcmp(Filenames->library, "")){
      fprintf(stdout, "Library file?>>");
fflush(stdout);
fgets(buf, MAXBUF, stdin);
buf[strlen(buf)-1]='\0';
strcpy(Filenames->library, buf);
if(NULL == (Libraryfile = fopen(buf, "r"))){
   fprintf(stderr, "Unable to open libraryfile   %s\n", buf);
   exit(1);
}
   }
   else{
if(NULL == (Libraryfile = fopen(Filenames->library,"r"))){
   fprintf(stderr,"Unable to open libraryfile:  %s\n", Filenames->library);
   exit(1);
}
   }

/*** get transform filename and open for input***/

   if (!strcmp(Filenames->transform, "")){
fprintf(stdout, "Transform file?>>");
fflush(stdout);
fgets(buf, MAXBUF, stdin);
buf[strlen(buf)-1]='\0';
strcpy(Filenames->transform, buf);
if(NULL == (Transformfile = fopen(buf, "r"))){
   fprintf(stderr, "Unable to open transform file: %s\n", buf);
   exit(1);
}
   }
   else{
if(NULL == (Transformfile = fopen(Filenames->transform, "r"))){
   fprintf(stderr, "Unable to open transformfile: %s\n", Filenames->transform);
   exit(1);
}
   }

/*** get smiles filename and open for input ***/

   if (!strcmp(Filenames->smiles, "")){
      fprintf(stdout,"SMILES file? >>");
      fflush(stdout);
      fgets(buf,MAXBUF,stdin);
      buf[strlen(buf)-1]='\0';
strcpy(Filenames->smiles, buf);
      if(NULL == (Smilesfile = fopen(buf,"r"))){
         fprintf(stderr, "Unable to open smilesfile:  %s\n",buf);
         exit(1);
      }
   }
   else{
      if(NULL == (Smilesfile = fopen(Filenames->smiles,"r"))){
         fprintf(stderr, "Unable to open smilesfile:  %s\n",Filenames->smiles);
         exit(1);
      }
   }

/*** get vbinding filename and open for input ***/

   if (!strcmp(Filenames->vbind, "")){
      fprintf(stdout,"VBINDING file? >>");
      fflush(stdout);
      fgets(buf,MAXBUF,stdin);
      buf[strlen(buf)-1]='\0';
strcpy(Filenames->vbind, buf);
      if(NULL == (Vbindingfile = fopen(buf,"r"))){
         fprintf(stderr, "Unable to open vbindingfile:  %s\n",buf);
         exit(1);
      }
   }
   else{
      if(NULL == (Vbindingfile = fopen(Filenames->vbind,"r"))){
         fprintf(stderr, "Unable to open vbindingfile:  %s\n",Filenames->vbind);
         exit(1);
      }
   }

/*** get rxn levels filename and open for input ***/

   if(rxnlevels){
      if (!strcmp(Filenames->rxn_levels, "")){
         fprintf(stdout,"RXN LEVEL file? >>");
         fflush(stdout);
         fgets(buf,MAXBUF,stdin);
         buf[strlen(buf)-1]='\0';
   strcpy(Filenames->rxn_levels, buf);
         if(NULL == (Rxnlevelsfile = fopen(buf,"r"))){
            fprintf(stderr, "Unable to open rxn levels file:  %s\n",buf);
            exit(1);
         }
      }
      else{
         if(NULL == (Rxnlevelsfile = fopen(Filenames->rxn_levels,"r"))){
            fprintf(stderr, "Unable to open rxn levels file:  %s\n",Filenames->rxn_levels);
            exit(1);
         }
      }
   } /* end rxnlevels */

/*** open tempfile for reading and writing  ***/

   strftime(buf, MAXBUF, "temp.%H%M%S", localtime(&Starttime));
   if(NULL == (Tempout = fopen(buf, "w"))){
fprintf(stderr, "Unable to open tempfile for writing");
exit(1);
   }
   if(NULL == (Tempin = fopen(buf, "r"))){
fprintf(stderr, "Unable to open tempfile for reading");
exit(1);
   }

/*** write start-up info to Outfile***/

   fprintf(Outfile, "Diversify Version:  %.2f\n", VERSION);
   fprintf(Outfile, "Start:  %s\n\n", ctime(&Starttime));
   fprintf(Outfile, "Input file:\t\t%s\n", Filenames->input);
   fprintf(Outfile, "Smiles file:\t\t%s\n", Filenames->smiles);
   fprintf(Outfile, "Vbinding file:\t\t%s\n", Filenames->vbind);
   fprintf(Outfile, "Transform file:\t\t%s\n", Filenames->transform);
   fprintf(Outfile, "Library file:\t\t%s\n", Filenames->library);
   fprintf(Outfile, "Output file:\t\t%s\n", Filenames->output);
   if(rxnlevels){
      fprintf((Outfile, "Rxn levels file:\t\t%s\n", Filenames->rxn_levels);
   }

/***read in and initialize vbindings   ***/

   vbindcount = vbindinit(Vbindingfile);
   fprintf(Outfile, "\n%d vbinding(s) initialized.\n\n", vbindcount);

/***read in starting molecule   ***/

   if(NULL == (Active = readinmols(Smilesfile, Lastptr))){
fprintf(stderr, "Unable to read starting molecules.\n");
exit(1);
   }
   fprintf(Outfile, "%d Molecule(s) read.\n\n", syncount);
   fclose(Smilesfile);
   First = Active;

/***read in transforms   ***/

   if(NULL == (transtot = readtrans(Transformfile, transforms))){
fprintf(stderr, "Unable to read in transforms.\n\n");
exit(1);
   }
   fprintf(Outfile, "%d Transforms read.\n\n", transtot);
   fclose(Transformfile);

/***read in libraries   ***/

   if(NULL == (Libraries = readlibs(Libraryfile, Libraries, &libtot, transforms, transtot))){
fprintf(stderr, "Error reading in libraries.\n\n");
exit(1);
   }
   fprintf((Outfile, "%d Libraries read.\n\n", libtot);
   fclose(Libraryfile);

/***attach libraries to transforms***/

   for(count = 0;count < transtot;count++){
if(transforms[count].flag == 'c'){
```

```
   for(count1 = 0;count1 < libtot;count1++){
if(0 == strcmp(transforms[count].type.con.libraryname,   \
Libraries[count1].name)){
   transforms[count].type.con.Library = &(Libraries[count1]);
   break;
}
   }
}
   }

/***read in reaction levels***/


   if(rxnlevels){
levels = readlevels(Rxnlevelsfile, transforms, transtot, &rxnlevels, Molecule_parameters->maxdepth);
fprintf(Outfile, "\n%d reaction levels.\n\n", rxnlevels);
for(count = 0;count <rxnlevels;count++){
   fprintf(Outfile, "Level %d \n", count);
   for(count1 = 0;count1 < transtot;count1++){
if(levels[count][count1]){
   fprintf(Outfile, "\t%s\n", transforms[count1].name);
}
   }
   fprintf(Outfile, "\n");
}
fprintf(Outfile, "\n");
   }
   fflush(Outfile);

/***write all molecules to beginning of Outfile   ***/

   Going = tempwrite(Tempout, Active);
   while(Going != NULL){
Tofree = Going;
Going = tempwrite(Tempout, Going);
free(Tofree);
   }

/***re-read starting molecule for active***/

   Active = re_read(Tempin, Active);
   Last = Active;

/******************************MAIN LOOP**************************************/
/******************************MAIN LOOP**************************************/
/******************************MAIN LOOP**************************************/
/***cycle through queue attempt to transform each molecule   ***/

   while(Active != NULL){

goodmol = TRUE;

/***test to see if the molequeue is ok for transformation***/

goodmol = prune(Active, Molecule_parameters);

/***screen for identity to previous molecules   ***/
/*
index = 0,
index1 = 0;
end = strlen(Active->smiles);
strcpy(buf, Active->smiles);
for(count=0;count<end,count++){
   index += MAXSMILES*count + buf[count] - '#';
   index1 += MAXSMILES*count + buf[count].
}
index1 -= '#'*count;
fprintf(stderr, "%d %d\n", index, index1);
if(uniquelist[index] == -1){
   uniquelist[index] = Active->synno;
   uniquecount++,
}
else{
   Active->identical = uniquelist[index];
   goodmol = FALSE;
   idcount++;
}
*/
/***apply transforms to active molecule if it's good   ***/

if(goodmol){
if((transcount != Active->depth)||(Active->depth == 0))&&(rxnscheme)){
   fprintf(stderr, "\nDepth = %d Transform = %s\n", Active->depth, transforms[Active->depth].name);
}
   for(transcount =0; transcount < transtot , transcount++){
if(rxnscheme){ /* if rxn scheme only apply 1 transform */
   transcount = Active->depth;
fprintf(stderr, " "),
}
if((rxnlevels)&&(!levels[Active->depth][transcount])){
   continue;
}
switch(transforms[transcount].flag){
   case 'r':
replace(Active, &transforms[transcount], Lastptr),
break;
   case 'c':
connect(Active, &transforms[transcount], Lastptr);
break;
   case 's':
if(Active->transform[0] != 'c'){
   separate(Active, &transforms[transcount], Lastptr);
}
break;
   case 'l':
/* library transform  don't apply to molecules */
break;
   default:
```

```
fprintf(stderr, "Reaction %d (%s) undefined type.", \
   transcount, transforms[transcount].name);
exit(1);
break;
} /* end switch */

/***while products are left, retire   ***/

Going = Active->Next;
Tofree = Active->Next;
   while(Going != NULL){
   Going = tempwrite(Tempout, Going);
}
free(Tofree);
Last = Active;

if(rxnscheme){
   break; /* if rxn scheme only apply 1 transform */
}

   } /* end for transcount */

} /* end if goodmol */

/***writeout transformed molequeue with children***/

/*
if(NULL == fill_dendogram(Active, &maintree)){
   fprintf(stderr, "Unable to add %s to dendogram \n", Going->smiles);
}
*/
Active = retire(Outfile, Active);


/***get next molequeue  ***/

Active = re_read(Tempin, First);
Last = Active;

   } /* end while Active isn't NULL */

/**************************END MAIN LOOP**************************************/
/**************************END MAIN LOOP**************************************/
/**************************END MAIN LOOP**************************************/

/***write out psfile of dendogram***/
/*
if(FALSE == write_dendogram(&maintree)){
   fprintf(stderr, "Error writing dendogram to psfile.\n");
}
*/
/***number of identicals***/

   fprintf(Outfile, "\n%d of %d (or %2d%%) were unique.\n",   \
syncount-idcount, syncount, (int)((float)(syncount-idcount)*100.0/(float)syncount));
   fprintf(stdout, "\n%d of %d (or %2d%%) were unique.\n",   \
syncount-idcount, syncount, (int)((float)(syncount-idcount)*100.0/(float)syncount));
/*
   fprintf(Outfile, "\n%d of %d (or %2d%%) were unique.\n",   \
uniquecount, syncount, (int)((float)(uniquecount)*100.0/(float)syncount));
   fprintf(stdout, "\n%d of %d (or %2d%%) were unique.\n",   \
uniquecount, syncount, (int)((float)(uniquecount)*100.0/(float)syncount));
*/

/***clean up output   ***/

   Endtime = time(NULL);
   fprintf(Outfile, "\nFinished at: %s\n", ctime(&Endtime));
   fprintf(Outfile, "Elapsed time. %.0f seconds\n", difftime(Endtime, Starttime));
   fprintf(Outfile, "Bye.\n");
   fflush(Outfile);

   fclose(Outfile);
   fclose(Tempin);
   fclose(Tempout);
   strftime(buf, MAXBUF, "temp.%H%M%S", localtime(&Starttime));

   if(FALSE){
if(FALSE != remove(buf)){
   fprintf(stderr, "Unable to remove temp file %s.\n\n", buf);
}
   }

} /* end main */

/***************************************************************
* fill in dendogram before retireing molequeue's
***************************************************************/
dendogram  *fill_dendogram(molequeue *Add, dendogram *Tree)
{
   int   levelno = Add->depth,
   memberno = Tree->level[levelno].membercount;

/***set tree depth   ***/

   if(levelno > Tree->numlevels){
Tree->numlevels = levelno;
memberno = 0;
Tree->level[levelno].zerocount=Tree->level[levelno-1].zerocount;
   }

/***set member, parent, and children properties***/

   Tree->level[levelno].member[memberno] = Add->synno;
   sscanf(Add->parent, "%d: ", &(Tree->level[levelno].parent[memberno]));
   Tree->level[levelno].nochildren[memberno] = Add->nochildren;
   if(Add->nochildren == 0){
Tree->level[levelno].zerocount++;
```

```
Tree->level[levelno].nozero[memberno+1] =   \
Tree->level[levelno].nozero[memberno] + 1;
   }

/***increment membercount***/

   Tree->level[levelno].membercount++;

   return(Tree);
}
/************************************************************
* make Post-Script file of dendogram
************************************************************/
int write_dendogram(dendogram *Tree)
{
   FILE   *Psfile;
   typedef struct point{
float x;
float y;
   }point;
   point   parent[MAXMOLEQUEUE];
   float   vspace,
   hspace,
   vsum;
   int    zerokids,
   kids,
   levelcount,
   membercount,
   parentoffset,

   if(NULL == (Psfile = fopen("ps.out","w"))){
fprintf(stderr,"Unable to open Psfile \n");
exit(1);
   }

/***write ps header    ***/

   fprintf(Psfile, "%%%!PS-Adobe-3.0\n");
   fprintf(Psfile, "%%%%BoundingBox: 0.00 0.00 612.00 792.00\n");
   fprintf(Psfile, "%%%%Creator: div\n");
   fprintf(Psfile, "%%%%Title: dendogram\n");
   fprintf(Psfile, "%%%%PageOrder: Ascend\n");
   fprintf(Psfile, "%%%%Orientation: Portrait\n");
   fprintf(Psfile, "%%%%DocumentData: Clean7Bit\n");
   fprintf(Psfile, "%%%%EndComments\n");
   fprintf(Psfile, "%%%%BeginProlog\n");
   fprintf(Psfile, "%%%%\n");
   fprintf(Psfile, "/c  { gsave newpath 0 360 arc stroke grestore } def\n");
   fprintf(Psfile, "/C  { gsave newpath 0 360 arc fill grestore   } def\n");
   fprintf(Psfile, "/d  { lineto } def\n");
   fprintf(Psfile, "/f  { gsave stroke grestore } def\n");
   fprintf(Psfile, "/m  { moveto } def\n");
   fprintf(Psfile, "/o  { gsave newpath 0 360 arc stroke grestore } def\n");
   fprintf(Psfile, "/O  { gsave newpath 0 360 arc fill grestore   } def\n");
   fprintf(Psfile, "/rp { newpath 4 2 roll moveto\n");
   fprintf(Psfile, "      dup 0 exch rlineto exch dup 0 rlineto\n");
   fprintf(Psfile, "      exch -1 mul 0 exch rlineto -1 mul 0 rlineto\n");
   fprintf(Psfile, "      closepath\n");
   fprintf(Psfile, "    } def\n");
   fprintf(Psfile, "/r  { gsave rp stroke grestore } def\n");
   fprintf(Psfile, "/R  { gsave rp fill   grestore } def\n");
   fprintf(Psfile, "/s  { stroke } def\n");
   fprintf(Psfile, "/bg { 1 setgray } def\n");
   fprintf(Psfile, "/fg { 0 setgray } def\n");
   fprintf(Psfile, "/w  { setlinewidth } def\n");
   fprintf(Psfile, "%%%%EndProlog\n");
   fprintf(Psfile, "%%%%Page: 1\n");
   fprintf(Psfile, "gsave\n");
   fprintf(Psfile, "fg\n");
   fprintf(Psfile, "1 setlinecap\n");
   fprintf(Psfile, "1 setlinejoin\n");
   fprintf(Psfile, "3 setmiterlimit\n");
   fprintf(Psfile, "0.10 w\n");
   fprintf(Psfile, "/bg { 1 setgray } def /fg { 0 setgray } def\n");
   fprintf(Psfile, "fg\n");
   fprintf(Psfile, "0 setgray\n");
   fprintf(Psfile, "%.2f %.2f m\n", PSLEFT, PSBOTTOM);
   fprintf(Psfile, "%.2f %.2f d\n", PSLEFT, PSTOP);
   fprintf(Psfile, "%.2f %.2f d\n", PSRIGHT, PSTOP);
   fprintf(Psfile, "%.2f %.2f d\n", PSRIGHT, PSBOTTOM);
   fprintf(Psfile, "%.2f %.2f d\n", PSLEFT, PSBOTTOM);

/***push nochildren indicators into the children level  ***/

/***set number of zero's in lowest level to zero (all have no children) ***/

   levelcount = Tree->numlevels;
   for(membercount = 0;membercount < Tree->level[levelcount].membercount;membercount++){
Tree->level[levelcount].nozero[membercount]=0;
   }

/***pass number of zero's on to first child from each parent    ***/

   for(levelcount = 1;levelcount <= Tree->numlevels;levelcount++){
parentoffset = Tree->level[levelcount-1].member[0];
for(membercount = 0;membercount < Tree->level[levelcount].membercount;membercount++){
   if(Tree->level[levelcount].parent[membercount]!=Tree->level[levelcount].parent[membercount-1]){
Tree->level[levelcount].nozero[membercount] =   \
   Tree->level[levelcount].nozero[membercount] + \
   Tree->level[levelcount-1].nozero[Tree->level[levelcount].parent[membercount]-parentoffset];
   }
}
   }

/***reset current row zero's to only reflect those from parent rows***/

   for(levelcount = 0;levelcount < Tree->numlevels;levelcount++){
for(membercount = (Tree->level[levelcount].membercount-1);membercount >= 0;membercount--){
   if(Tree->level[levelcount].nochildren[membercount] == 0){
      Tree->level[levelcount].nozero[membercount+1] =   \
      Tree->level[levelcount].nozero[membercount+1] -   \
      (Tree->level[levelcount].nozero[membercount]+1);
   }
      }
   }

/***write out levels    ***/

   hspace = (float)(PSRIGHT-PSLEFT) / Tree->numlevels;
   vspace = (float)(PSTOP-PSBOTTOM) / (1+Tree->level[Tree->numlevels].membercount \
 +Tree->level[Tree->numlevels-1].zerocount);
   for(levelcount=Tree->numlevels;levelcount > 0;levelcount--){
parentoffset = Tree->level[levelcount-1].member[0];

/***if not first level, need to sum offspring and put in children***/

if(levelcount < Tree->numlevels){

/***reset number of children counters for level above   ***/

   for(membercount=0;membercount < Tree->level[levelcount-1].membercount;membercount++){
Tree->level[levelcount-1].nochildren[membercount]=0;
   }

/***sum over children from this level to give number of children at next level  ***/

   for(membercount=0;membercount < Tree->level[levelcount].membercount;membercount++){
if(Tree->level[levelcount].nochildren[membercount] == 0){
   Tree->level[levelcount].nochildren[membercount] = 1;
}
Tree->level[levelcount-1].nochildren[Tree->level[levelcount].parent[membercount]-parentoffset] =   \
     (Tree->level[levelcount-1].nochildren[Tree->level[levelcount].parent[membercount]-parentoffset]
+   \
Tree->level[levelcount].nochildren[membercount]);
   }
}

/***calculate where parents will be    ***/

vsum = 0.0;
for(membercount=0;membercount < Tree->level[levelcount-1].membercount;membercount++){
   parent[membercount].x = PSLEFT + hspace*(levelcount-1);
   kids = (Tree->level[levelcount-1].nochildren[membercount]);
   if(kids==0)
kids=1;
   zerokids = (Tree->level[levelcount-1].nozero[membercount]);
   parent[membercount].y = PSBOTTOM + vsum + ((kids+1)*vspace/2.0  \
    + (zerokids*vspace));
   vsum += (kids*vspace + zerokids*vspace);
}

/***calculate where children will be and connect children to parents    ***/

vsum = 0.0;
for(membercount=0;membercount < Tree->level[levelcount].membercount;membercount++){
   zerokids = Tree->level[levelcount].nozero[membercount];
   kids = Tree->level[levelcount].nochildren[membercount];
   if(kids==0)
kids=1;
   fprintf(Psfile, "%.2f %.2f m\n",\
(PSLEFT + hspace*levelcount),\
(PSBOTTOM + vsum + ((kids+1)*vspace/2.0) + (zerokids*vspace)));
   vsum += (kids*vspace + zerokids*vspace);
   fprintf(Psfile, "%.2f %.2f d\n",\
parent[Tree->level[levelcount].parent[membercount]-parentoffset].x, \
parent[Tree->level[levelcount].parent[membercount]-parentoffset].y);
   fflush(Psfile);
}
   }



/***write ps footer    ***/

   fprintf(Psfile, "s\n");
   fprintf(Psfile, "grestore showpage\n");
   fprintf(Psfile, "%%%%Trailer\n");

   return(TRUE);
}
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "trans.h"
#include "mol.h"
#include "rep.h"
#include "debug.h"

extern int syncount;

/*********************************************************************
 * replace
 *
 * this is a transform to replace the inclusion with the target.
 *********************************************************************/
voidreplace(molequeue *Reactant, transform *Transform, molequeue **Lastptr)
{
   charparent[MAXSMILES];
   intcopyct,
reactatom,
reactbond,
atomcount,
interbondct,
bondcount,
seqcount,
xatomct,
```

```
count, count1,
length,
ibond,
realbond,
chival[MAXCONNECTIONS],
wildchival[MAXTEMPLATE];
   dt_Handleerror,
errors,
pathset,
paths,
path,
atoms,
atom,
atom1,
bonds, bonds1,
bond,
bond1, bond2, bond3,
copybonds[MAXCONNECTIONS],
copyatoms[MAXCONNECTIONS],
delatoms[MAXTEMPLATE],
delbonds[MAXTEMPLATE],
newatoms[MAXTEMPLATE],
newbonds[MAXTEMPLATE],
chiseq[MAXCONNECTIONS],
wildchiseq[MAXTEMPLATE],
   dt_Stringstr,
   molequeue*Product;


   if(screen(Reactant, Transform) == FALSE)
return;

/***malloc new molecule (Product)***/

   if(NULL == (Product = (molequeue *)malloc(sizeof(molequeue)))){
fprintf(stderr, "Out of memory in: replace \n");
fprintf(stderr, "Unable to open molecule %d \n", syncount);
exit(1);
   }

/*********************************************************************
 * In the following section the product will be created.
 * 1. duplicate reactant->product
 * 2. find reactant set
 * 3. loop through reactant set
 *if appropriate add new atom
 *    loop through reactant bonds and add to product
 * 4. add rest of product atoms.
 * 5. add product bonds.
 * 6. delete reactant set.
 * 7. set hydrogens in new molecule
 * this should result in the product molecule
 *********************************************************************/

/***copy reactant molecule to product molecule   ***/

   Product->molecule = dt_copy(Reactant->molecule);

/***make product molecule modifiable   ***/

   if(FALSE == dt_mod_on(Product->molecule)){
fprintf(stderr, "Can't modify %s in rep \n", Product->smiles);
dt_dealloc(Product->molecule);
free(Product);
return;
   }

/***find reactant path in product molecule   ***/

   if(NULL_OB == (pathset = dt_umatch(Transform->screenin, Product->molecule, FALSE))){
fprintf(stderr, "Unable to find remove. %s of %s in %s\n", \
   Transform->inclusion, \
   Transform->name, \
   Product->smiles);
/***this should NEVER occur but is left in for historical reasons ***/
fprintf(stderr, "BIG problem, a reactant conatined screenin, but the product didn't \n");
dt_dealloc(Product->molecule);
free(Product);
return;
   }

/***get atoms in screenin   ***/

   paths = dt_stream(pathset, TYP_PATH);
   path = dt_next(paths);
   atoms = dt_stream(path, TYP_ATOM);

/***loop through reactant set get atoms, bonds  ***/

   reactatom = 0;
   reactbond = 0;
   while(NULL_OB != (delatoms[reactatom] = dt_next(atoms))){
if(NULL_OB != (bonds = dt_stream(delatoms[reactatom], TYP_BOND))){
   while(NULL_OB != (delbonds[reactbond] = dt_next(bonds))){
reactbond++;
   }
}
dt_dealloc(bonds);
reactatom++;
   }
   dt_dealloc(atoms);
   dt_dealloc(pathset);
   dt_dealloc(paths);
   dt_dealloc(path);

   atomcount = 0;
   bondcount = 0;
   while(atomcount < reactatom){
```

```
/***if appropriate add product atom***/

if(Transform->type.rep.productatom[atomcount].atomicno >= 0){

   if(Transform->type.rep.productatom[atomcount].atomicno > 0){
newatoms[atomcount] = dt_addatom(Product->molecule,\
   Transform->type.rep.productatom[atomcount].atomicno, 0);
dt_setaromatic(newatoms[atomcount],\
   Transform->type.rep.productatom[atomcount].aromatic);
dt_setcharge(newatoms[atomcount],\
   Transform->type.rep.productatom[atomcount].formalcharge);
   }

/***atomicno 0 is for wildcard, copy reactant atom***/

   wildchival[atomcount] = DX_CHI_NONE;
   if(Transform->type.rep.productatom[atomcount].atomicno == 0){
newatoms[atomcount] = dt_addatom(Product->molecule,\
   dt_number(delatoms[atomcount]), dt_imp_hcount(delatoms[atomcount]));
dt_setaromatic(newatoms[atomcount],\
   dt_aromatic(delatoms[atomcount]));
dt_setcharge(newatoms[atomcount],\
   dt_charge(delatoms[atomcount]));

/***copy wildcard atom chirality if it exists   ***/

wildchiseq[atomcount] = dt_alloc_seq();
if(TRUE == dt_imp_hcount(delatoms[atomcount])){
   atom1 = dt_isohydro();
   wildchiseq[atomcount] = dt_append(wildchiseq[atomcount], atom1);
}
if(NULL_OB != (bonds = dt_stream(delatoms[atomcount], TYP_BOND))){
   while(NULL_OB != (bond = dt_next(bonds))){
wildchiseq[atomcount] = dt_append(wildchiseq[atomcount], bond);
   } /* end while wildcard's bonds */
} /* end if bonds */
dt_dealloc(bonds);
dt_reset(wildchiseq[atomcount]);
wildchival[atomcount] = dt_chival(delatoms[atomcount], wildchiseq[atomcount]);

   } /* end if wildcard */

/***loop through reactant atom bonds and add to product ***/

   if(NULL_OB != (bonds = dt_stream(delatoms[atomcount], TYP_BOND))){
copyct = 0;
while(NULL_OB != (copybonds[copyct] = dt_next(bonds))){

/***only count bonds to unchanged atoms ***/

   realbond = TRUE;
   chival[copyct] = DX_CHI_NONE;
   copyatoms[copyct] = dt_xatom(delatoms[atomcount], copybonds[copyct]);
   for(count=0;count<reactatom;count++){
if(copyatoms[copyct] == delatoms[count]){
realbond = FALSE;
break;
}
   }

/***if bond is to non-delatom check for chirality in xatom   ***/

   if(realbond){
chiseq[copyct] = dt_alloc_seq();
if(TRUE == dt_imp_hcount(copyatoms[copyct])){
   atom1 = dt_isohydro();
   chiseq[copyct] = dt_append(chiseq[copyct], atom1);
}
if(NULL_OB != (bonds1 = dt_stream(copyatoms[copyct], TYP_BOND))){
   while(NULL_OB != (bond1 = dt_next(bonds1))){
chiseq[copyct] = dt_append(chiseq[copyct], bond1);
   } /* end while non-delatom's bonds */
} /* end if bonds */
dt_dealloc(bonds1);
dt_reset(chiseq[copyct]);
chival[copyct] = dt_chival(copyatoms[copyct], chiseq[copyct]);
copyct++;
   } /* end if bond is real (ie-to a non-delatom) */

/***if atom on either end of copybond is chiral substutute newbond.   ***/

   else{
for(count=0;count<atomcount;count++){
   if(copyatoms[copyct] == delatoms[count]){
break;
   }
}
if((count<atomcount)&&((wildchival[atomcount] > DX_CHI_NONE)|| \
   (Transform->type.rep.productatom[count].atomicno == 0))){
copyatoms[copyct] = newatoms[count];
    copyct++;
} /* end if wildchival */
   } /* end else */

} /* while there are copybonds */

/***make new bond, substituting chirality if necessary  ***/

for(count=0;count<copyct;count++){
   newbonds[bondcount] = dt_addbond(newatoms[atomcount],\
   copyatoms[count], dt_bondtype(copybonds[count]));

/***if atom across bond is chiral reset chirality***/

   if(chival[count] > DX_CHI_NONE){
dt_reset(chiseq[count]);
```

```
while(NULL_OB != (bond = dt_next(chiseq[count]))){
   if(bond == copybonds[count]){
if(FALSE == dt_delete(chiseq[count])){
   fprintf(stderr, "trouble repairing non-delatom chirality in %s \n\n", Transform->name);
   }
dt_next(chiseq[count]);
chiseq[count] = dt_insert(chiseq[count], newbonds[bondcount]);
dt_reset(chiseq[count]);
dt_dealloc(copybonds[count]);
if(FALSE == dt_setchival(copyatoms[count], chiseq[count], chival[count])){
/*   fprintf(stderr, "trouble resetting non-delatom chirality in %s.\n\n", Transform->name);*/
}
break;
   }
}
   } /* end if xatom is chiral */

/***if atom across bond is wildcard reset chirality***/

   xatomct = -1;
   for(count1=0,count1<reactatom;count1++){
if(copyatoms[count] == newatoms[count1]){
   xatomct = count1,
   break,
}
   }
   if((xatomct >= 0)&&(wildchival[count1] > DX_CHI_NONE)){
dt_reset(wildchiseq[count1]),
while(NULL_OB != (bond = dt_next(wildchiseq[count1]))){
   if(bond == copybonds[count]){
if(FALSE == dt_delete(wildchiseq[count1])){
/*   fprintf(stderr, "trouble repairing non-delatom chirality in %s.\n\n", Transform->name);*/
   }
dt_next(wildchiseq[count1]);
wildchiseq[count1] = dt_insert(wildchiseq[count1], newbonds[bondcount]);
dt_reset(wildchiseq[count1]);
dt_dealloc(copybonds[count]);
if(FALSE == dt_setchival(copyatoms[count], wildchiseq[count1], wildchival[count1])){
/*   fprintf(stderr, "trouble resetting non-delatom chirality in %s \n\n", Transform->name);*/
}
break;
   }
}
   } /* end if xatom is chiral */

/***if current atom is a chiral wildcard reset chirality***/

   if(wildchival[atomcount] > DX_CHI_NONE){
dt_reset(wildchiseq[atomcount]);
while(NULL_OB != (bond = dt_next(wildchiseq[atomcount]))){
   if(bond == copybonds[count]){
if(FALSE == dt_delete(wildchiseq[atomcount])){
/*   fprintf(stderr, "trouble repairing wildcard chirality in %s.\n\n", Transform->name);*/
   }
dt_next(wildchiseq[atomcount]);
wildchiseq[atomcount] = dt_insert(wildchiseq[atomcount], newbonds[bondcount]);
dt_reset(wildchiseq[atomcount]).
if(FALSE == dt_setchival(newatoms[atomcount], wildchiseq[atomcount], wildchival[atomcount])){
/*   fprintf(stderr, "Trouble resetting wildcard atom chirality in %s.\n\n", Transform->name); */
}
else{
}
break;
   }
}

   } /* end if wildcard is chiral */

/***if newbond was made, increment bondcount    ***/

   if(NULL_OB != newbonds[bondcount]){
bondcount++;
   }
} /* end for make newbonds */
for(count=0;count<copyct;count++){
   dt_dealloc(chiseq[count]),
}

   } /* end loop through reactant atom's bonds */
   dt_dealloc(bonds),

} /* end if atomicno >0 */

atomcount++;
   } /* end while reactant set atom */

/***add rest of the product atoms***/

   for(atomcount = atomcount;atomcount < Transform->type.rep.atomnumber;atomcount++){
newatoms[atomcount] = dt_addatom(Product->molecule,\
Transform->type.rep.productatom[atomcount].atomicno, 0);
dt_setaromatic(newatoms[atomcount],\
Transform->type.rep.productatom[atomcount].aromatic);
dt_setcharge(newatoms[atomcount],\
Transform->type.rep.productatom[atomcount].formalcharge);
   }

/***add product bonds   ***/

   interbondct = bondcount;
   for(bondcount = 0;bondcount < Transform->type.rep.bondnumber;bondcount++){

if(NULL_OB == (newbonds[bondcount+interbondct] = dt_bond(newatoms[Transform->type.rep.productbond[bondcount].iatom1],\
 newatoms[Transform->type.rep.productbond[bondcount].iatom2]))){
   newbonds[bondcount+interbondct] = dt_addbond(newatoms[Transform->type.rep.productbond[bondcount].iatom1],   \
 newatoms[Transform->type.rep.productbond[bondcount].iatom2],   \
 Transform->type.rep.productbond[bondcount].bondtype);
}
else{
   dt_setbondtype(newbonds[bondcount+interbondct], Transform->type.rep.productbond[bondcount].bondtype);
}
   }

/***delete reactant atoms and bonds***/

   for(count = 0;count < reactatom;count++){
dt_dealloc(delatoms[count]);
   }
   for(count = 0;count < reactbond;count++){
dt_dealloc(delbonds[count]);
   }

/***   set number of implicit hydrogens for each atom     ***/

   for(count=0;count < Transform->type.rep.atomnumber;count++){
      dt_setimp_hcount(newatoms[count], hcount(&(newatoms[count])));
   }

/***set chirality on chiral transform atoms    ***/

   for(count=0;count < Transform->type.rep.atomnumber;count++){
if(Transform->type.rep.productatom[count].chival != DX_CHI_NONE){

/***get chiral sequence of bonds***/

   chiseq[0] = dt_alloc_seq();
   for(seqcount=0;seqcount < MAXCHIRAL;seqcount++){
if(Transform->type.rep.productatom[count].chiseq[seqcount] == DIV_IMP_HYDROGEN){
   atom = dt_isohydro();
   chiseq[0] = dt_append(chiseq[0], atom);
}
else if(Transform->type.rep.productatom[count].chiseq[seqcount] >= 0){
   chiseq[0] = dt_append(chiseq[0],\
newbonds[(Transform->type.rep.productatom[count].chiseq[seqcount])+interbondct]);
}
else{
   break;
}

   }

/***set chirality of atom***/

   if(FALSE == dt_setchival(newatoms[count], chiseq[0],\
   Transform->type.rep.productatom[count].chival)){
fprintf(stderr, "Error setting chirality of atom in replace.\n");
fprintf(stderr, "Transform: %s, Productatom: %d\n", Transform->name, count);
fprintf(stderr, "chival = %d.\n", Transform->type.rep.productatom[count].chival);
   }
   dt_dealloc(chiseq[0]);
   dt_dealloc(atom);
}
   }

/***set chirality of bonds***/

   for(count=0;count<Transform->type.rep.bondnumber;count++){
if((Transform->type.rep.productbond[count].chival == DX_CHI_CIS)||   \
  (Transform->type.rep.productbond[count].chival == DX_CHI_TRANS)){
  if(FALSE == dt_setdbo(newbonds[count+interbondct],   \
newbonds[(Transform->type.rep.productbond[count].chiseq[0])+interbondct], \
newbonds[(Transform->type.rep.productbond[count].chiseq[1])+interbondct], \
Transform->type.rep.productbond[count].chival)){
fprintf(stderr, "Error setting double bond chirality in replace.\n");
fprintf(stderr, "Transform: %s, Productbond: %d\n", Transform->name, count);
fprintf(stderr, "chival = %d.\n", Transform->type.rep.productbond[count].chival);
errors = dt_errors(DX_ERR_NOTE);
while(NULL_OB != (error = dt_next(errors))){
   str = dt_stringvalue(&length, error);
   fprintf(stderr, "%s.\n", str);
}
   }
  }
   }

/***initialize product mmolequeue and reorder queue***/

   if(FALSE == (dt_mod_off(Product->molecule))){
fprintf(stderr, "Unable to turn modify off in replace.\n");

/***clean up after reaction    ***/

      dt_dealloc(pathset);
      dt_dealloc(paths);
      dt_dealloc(path);
      dt_dealloc(atoms);
      dt_dealloc(bonds);
      for(count=0;count < Transform->type.rep.atomnumber;count++){
      dt_dealloc(newatoms[count]);
      }
for(count=0;count<reactatom;count++){
   if(Transform->type.con.productatom[count].atomicno == 0){
dt_dealloc(wildchiseq[count]);
   }
}

return;
   } /* end if FALSE */

   strncpy(Product->smiles, dt_cansmiles(&length, Product->molecule, TRUE), length);
   Product->smiles[length] = '\0';
   strcpy(Product->externalid, Reactant->externalid),
   if(strlen(Product->externalid) + strlen(Transform->name) < MAXSMILES){
```

```
        strcat(Product->externalid, ":");
        strcat(Product->externalid, Transform->name);
    }
/*fprintf(stderr, "%d %.*s\n", syncount, strlen(Product->smiles), Product->smiles);*/
    sprintf(parent, "%d", Reactant->synno);
    strcpy(Product->parent, parent);
    strcat(Product->parent, ":");
    strcat(Product->parent, Reactant->smiles);
    Product->identical = -1;
    strcpy(Product->transform, Transform->name);
    Product->Next = NULL;
    Product->Previous = *Lastptr;
    Product->depth = Reactant->depth + 1;
    Product->synno = syncount;
    syncount++,
    Product->nochildren = 0;


    Reactant->children[Reactant->nochildren] = Product->synno;
    Reactant->nochildren++;

    (*Lastptr)->Next = Product;
    *Lastptr = Product,

/***clean up after reaction    ***/

    if(TRUE == dt_dealloc(pathset)){
fprintf(stderr, "true dealloc pathset\n");
    }
    if(TRUE == dt_dealloc(paths)){
fprintf(stderr, "true dealloc paths\n");
    }
    if(TRUE == dt_dealloc(path)){
fprintf(stderr, "true dealloc path\n");
    }
    if(TRUE == dt_dealloc(atoms)){
fprintf(stderr, "true dealloc atoms\n");
    }
    dt_dealloc(paths);
    dt_dealloc(path);
    dt_dealloc(atoms);
    dt_dealloc(bonds),
    for(count=0;count<reactatom;count++){
if(Transform->type con productatom[count].atomicno == 0){
    dt_dealloc(wildchiseq[count]);
}
    }

    return;
}

/*********************************************************************
 * hcount
 *
 * takes atom as an argument, and returns the number of implicit
 * hydrogens which the atom should contain
 *********************************************************************/
int hcount(dt_Handle *Atom)
{
    int formalcrg,
valence,
atomicno,
hcount,
count = 0;

    dt_Handlebonds,
bond;

    typedef struct {
int atomicno;
int valence;
    }table,

    table   altable[] = {   /* aliphatic table*/
1, 1,/* Hydrogen has 1 open valence     */
6, 4,/* Carbon has 4 open valences     */
7, 3,/* Nitrogen has 3 open valences    */
8, 2,/* Oxygen has 2 open valences    */
9, 1,/* F,Cl,Br,I have 1 open valence    */
17, 1,
35, 1,
53, 1,
15, 6,/* P has 6 open Valences    */
16, 6/* S has 6 open Valences    */
    };

    table   artable[] = {   /* aromatic table */
6, 3,/* aromatic Carbon have 3 open valence*/
7, 2,/* aromatic Nitrogen have 3 open valence (for now)*/
8, 2,/* aromatic Oxygen have 0 open valence    */
16, 2/* aromatic Sulfur have 0 open valence    */
    };

atomicno = dt_number(*Atom),
formalcrg = dt_charge(*Atom);
if(FALSE == dt_aromatic(*Atom)){
    while(altable[count].atomicno != atomicno){
count++,
    }
    valence = altable[count].valence,
    valence += formalcrg;
    if(NULL_OB != (bonds = dt_stream(*Atom, TYP_BOND))){
while(NULL_OB != (bond = dt_next(bonds))){
    valence -= dt_bondtype(bond),
}
if((atomicno == 16)&&(valence > 2)){
    valence -= 4;/* expanded sulfur d-orbital */
}
dt_dealloc(bonds);
return(valence);
    }

} /* end if aliphatic */
else{
    while(artable[count].atomicno != atomicno){
count++;
    }
    valence = artable[count].valence;
    valence += formalcrg;
    if(NULL_OB !=(bonds = dt_stream(*Atom, TYP_BOND))){
while(NULL_OB != (bond = dt_next(bonds))){
    valence--;
}
if((atomicno == 7)&&(dt_hcount(*Atom)==1)){
    valence = 1;/* [nH] */
}
dt_dealloc(bonds);
return(valence);
    }
} /* end else (aromatic) */
} /* END hcount */

/*********************************************************************
 * screen
 *
 * takes a reactant molecule and a transform and returns TRUE/FALSE
 * for whether the transform can be applied to the molecule.
 *********************************************************************/
int screen(molequeue *Reactant, transform *Transform)
{

    dt_Handlepathset,
paths,
path,
atoms,
atom,
expathset,
expaths,
expath,
exatoms,
exatom;
    intpathcount = 0,
flag;

/***check to see if screen is in molecule, if not return 0.***/

    if(NULL_OB == (pathset = dt_umatch(Transform->screenin, Reactant->molecule, FALSE))){
flag = FALSE;
    }

    else{
flag = TRUE;

/***count number of occurences in molecule, if > 1 return 0.    ***/

paths = dt_stream(pathset, TYP_PATH);
while(NULL_OB != (path = dt_next(paths))){
    atoms = dt_stream(path, TYP_ATOM);
    pathcount++;
    if(pathcount > 1){
flag = TRUE;
    }
}

/***check for exclusion structures which intersect inclusion path.***/

        if(0 != strcmp(Transform->exclusion, "\0")){
    if(NULL_OB != (expathset = dt_match(Transform->screenout, Reactant->molecule, FALSE))){
if(NULL_OB != (expaths = dt_stream(expathset, TYP_PATH))){
    while(NULL_OB != (expath = dt_next(expaths))){
if(NULL_OB != (exatoms = dt_stream(expath, TYP_ATOM))){
    while(NULL_OB != (exatom = dt_next(exatoms))){
while(NULL_OB != (atom = dt_next(atoms))){
    if(dt_uid(atom)==dt_uid(exatom)){
flag = FALSE;
break;
    }
}
dt_reset(atoms);
    } /* end exatom */
} /* end exatoms */
    } /* end expath */
} /* end expaths */
    } /* end expathset */
} /* end if exclusion */
    } /* end else */



/***clean up pre-screening objects    ***/

    dt_dealloc(pathset);
    dt_dealloc(expathset);

    return(flag);

} /* END screen */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "trans.h"
#include "mol.h"
#include "sep.h"
#include "rep.h"
#include "debug.h"
```

```
extern int syncount;

/**********************************************************************
 * separate
 *
 * breaks apart a molecule into fragments.
 **********************************************************************/
voidseparate(molequeue *Reactant, transform *Transform, molequeue **Lastptr)
{
   charparent[MAXSMILES];
   intcount,
length;
   molequeue*Product;


   if(screen(Reactant, Transform) == FALSE)
return;

/***malloc two new molecules (Product[0] and [1])***/

   if(NULL == (Product = (molequeue *)malloc(sizeof(molequeue)*2))){
fprintf(stderr, "Out of memory in  separate.\n");
fprintf(stderr, "Unable to open molecule %d.\n", syncount);
exit(1);
   }

/**********************************************************************
 * In the following section the product will be created.
 * 1  make 2 duplicates of the reactant.
 * 2  loop through the 2 molecules
 * a. find key0 in product0
 *b. loop through reactant set
 *if appropriate add new atom
 *loop through reactant bonds and add
 *c  loop from z atom to end of molecule deleteing all atoms/bonds
 *d. add rest of product atoms
 *e. add product bonds
 *f  delete reactant set
 *g. set hydrogens in new molecule
 *h. make unmodifiable
 * 3. fix both molequeues
 * 4  return both molequeues
 **********************************************************************/


/***loop through both product halves of reactant***/

   for(count=0;count < 2;count++){

/***copy reactant molecule to product molecules  ***/

Product[count].molecule = dt_copy(Reactant->molecule);

/***use subroutine to apply each part of the transform  ***/

if(NULL == subproduct(&(Product[count]), Transform, count)){
   fprintf(stderr, "Failed to make product %d from %s in %s \n",\
   count, Reactant->smiles, Transform->name),
   continue;
}

strncpy(Product[count].smiles, dt_cansmiles(&length, Product[count].molecule, TRUE), length);
Product[count].smiles[length] = '\0',
fprintf(stderr, "%d %.*s\n", syncount, strlen(Product[count].smiles), Product[count].smiles);
sprintf(parent, "%d", Reactant->synno);
strcpy(Product[count].parent, parent);
strcat(Product[count].parent, ":");
strcat(Product[count].parent, Reactant->smiles);
Product[count].identical = -1;
strcpy(Product[count].transform, Transform->name);
Product[count].Next = NULL,
Product[count].Previous = *Lastptr;
Product[count].depth = Reactant->depth + 1;
Product[count].synno = syncount,
syncount++,
Product[count].nochildren = 0;


Reactant->children[Reactant->nochildren] = Product[count].synno;
Reactant->nochildren++;

(*Lastptr)->Next = &(Product[count]);
*Lastptr = &(Product[count]),

   } /* end loop through product halves */

   return,
}

/**********************************************************************
 * subproduct
 *
 * takes 1/2 product and 1/2 transform and returns one of the products
 **********************************************************************/
molequeue *subproduct(molequeue *Product, transform *Transform, int subcount)
{
   intdelete,
copyct,
holdct,
count,
count1,
delcount,
reactatom,
reactbond,
atomcount,
bondcount,
max,
   dt_Handlepathset,
paths,
path,
atoms,
atom,
bonds,
bond,
newbond,
holdbond[5],
copybonds[5],
delatoms[MAXSMILES],
delbonds[MAXTEMPLATE],
newatoms[MAXTEMPLATE];

/***make product molecule modifiable   ***/

   if(FALSE == dt_mod_on(Product->molecule)){
fprintf(stderr, "Can't modify %s in sep.\n", Product->smiles);
dt_dealloc(Product->molecule);
free(Product);
return(NULL_OB);
   }

/***find reactant path in product molecule  ***/

   if(NULL_OB == (pathset = dt_umatch(Transform->screenin, Product->molecule, FALSE))){
fprintf(stderr, "Unable to find remove: %s of %s in %s\n", \
   Transform->inclusion, \
   Transform->name, \
   Product->smiles);
/***this should NEVER occur but is left in for historical reasons ***/
fprintf(stderr, "BIG problem, a reactant conatined screenin, but the product didn't.\n");
dt_dealloc(Product->molecule);
free(Product);
return(NULL_OB);
   }

/***get atoms in screenin   ***/

   paths = dt_stream(pathset, TYP_PATH);
   path = dt_next(paths);
   atoms = dt_stream(path, TYP_ATOM);

/***loop through reactant set  ***/

   reactatom = 0;
   delcount = 0;
   reactbond = 0;
   while(NULL_OB != (delatoms[reactatom] = dt_next(atoms))){
if(NULL_OB != (bonds = dt_stream(delatoms[reactatom], TYP_BOND))){
   while(NULL_OB != (delbonds[reactbond] = dt_next(bonds))){
reactbond++;
   }
}
reactatom++;
delcount++;
   }

   atomcount = 0;
   while(atomcount < reactatom){

/***if appropriate add product atom***/

if(Transform->type.sep.productatom[subcount][atomcount].atomicno > 0){
   newatoms[atomcount] = dt_addatom(Product->molecule,\
   Transform->type.sep.productatom[subcount][atomcount].atomicno, 0);
   dt_setaromatic(newatoms[atomcount],\
   Transform->type.sep.productatom[subcount][atomcount].aromatic);
   dt_setcharge(newatoms[atomcount],\
   Transform->type.sep.productatom[subcount][atomcount].formalcharge);

/***loop through reactant atom bonds and add to product ***/

if(NULL_OB != (bonds = dt_stream(delatoms[atomcount], TYP_BOND))){
   copyct = 0;
   while(NULL_OB != (copybonds[copyct] = dt_next(bonds))){
copyct++,
   }
   for(count=0;count<copyct;count++){
newbond = dt_addbond(newatoms[atomcount],\
dt_xatom(delatoms[atomcount], copybonds[count]), \
dt_bondtype(copybonds[count]));
   }
} /* end loop through bonds */

} /* end if atomicno >0 */

atomcount++;

   } /* end while reactant set atom */

/***loop from Z atom through product deleting unused half***/

   atomcount = 0;
   while(atomcount < reactatom){

if(Transform->type.sep.productatom[subcount][atomcount].atomicno == -2){
   if(NULL_OB != (bonds = dt_stream(delatoms[atomcount], TYP_BOND))){
holdct = 0;
while(NULL_OB != (holdbond[holdct] = dt_next(bonds))){
   holdct++;
} /* end if bond */
for(count = 0,count<holdct;count++){
   if(NULL_OB != (atom = dt_xatom(delatoms[atomcount], holdbond[count]))){
delete = TRUE;
for(count1=0;count1<delcount;count1++){
   if((atom == delatoms[count1])||(atom == newatoms[count1])){
delete = FALSE; /* don't delete key atoms */
break;
```

```
    }
} /*end loop through delatoms */
if(delete){
    if(FALSE == deletefragment(holdbond[count], &atom, delatoms, &delcount)){
fprintf(stderr, "\nError deleteing fragment.\n");
    }
} /* end delete */
    } /* end if atom */
} /* end for count */
    } /* end if bonds */
} /* end elseif */

atomcount++;

    } /* end while reactant set atom */

/***add rest of the product atoms***/

    for(atomcount = reactatom;atomcount < Transform->type.sep.atomnumber[subcount];atom-
count++){
newatoms[atomcount] = dt_addatom(Product->molecule,\
Transform->type.sep.productatom[subcount][atomcount].atomicno, 0);
dt_setaromatic(newatoms[atomcount],\
Transform->type.sep.productatom[subcount][atomcount].aromatic);
dt_setcharge(newatoms[atomcount],\
Transform->type.sep.productatom[subcount][atomcount].formalcharge);
    }

/***add product bonds    ***/

    for(bondcount = 0;bondcount < Transform->type.sep.bondnumber[subcount];bondcount++){
if(NULL_OB == (newbond = dt_bond(newatoms[Transform->type.sep.productbond[subcount][bond-
count].iatom1],\
 newatoms[Transform->type.sep.productbond[subcount][bondcount].iatom2]))){
    newbond = dt_addbond(newatoms[Transform->type.sep.productbond[subcount][bond-
count].iatom1],    \
 newatoms[Transform->type.sep.productbond[subcount][bondcount].iatom2],    \
  Transform->type.sep.productbond[subcount][bondcount].bondtype);
}
else{
    dt_setbondtype(newbond, Transform->type.sep.productbond[subcount][bondcount].bondtype);
}
    }

/***delete reactant atoms and bonds***/

    for(count = 0;count < delcount;count++){
dt_dealloc(delatoms[count]);
    }
    for(count = 0;count < reactbond;count++){
dt_dealloc(delbonds[count]);
    }

/***    set number of implicit hydrogens for each atom      ***/

    for(count=0;count < Transform->type.sep.atomnumber[subcount];count++){
dt_setimp_hcount(newatoms[count], hcount(&(newatoms[count])));
    }

/***initialize product mmolequeue and reorder queue***/

    if(FALSE == (dt_mod_off(Product->molecule))){
fprintf(stderr, "Unable to turn modify off in separate.\n");

/***clean up after reaction    ***/

dt_dealloc(pathset);
dt_dealloc(paths);
dt_dealloc(path);
dt_dealloc(atoms);
dt_dealloc(bonds);
for(count=0;count < Transform->type.sep.atomnumber[subcount];count++){
    dt_dealloc(newatoms[count]);
}
/*free(Product);*/
return(NULL_OB);
    } /* end if FALSE */

/***clean up after reaction    ***/

    dt_dealloc(pathset);
    dt_dealloc(paths);
    dt_dealloc(path);
    dt_dealloc(atoms);
    dt_dealloc(bonds);
    for(count=0;count < Transform->type.sep.atomnumber[subcount];count++){
dt_dealloc(newatoms[count]);
    }

/***return product***/

    return(Product);

} /* end subproduct */

/**********************************************************************
 * deletefragment
 *
 * recursively deletes any connected fragment starting with one atom.
 * be careful not to backtrack up the bond you start with
 **********************************************************************/
int deletefragment(dt_Handle backbond, dt_Handle *atom, dt_Handle *delete, int *incount)
{
    intcount,
count1,
repeat,
followct;
    dt_Handlexatom,
bonds,
bond,
followbond[5],

/***mark atom for deletion***/

    if(NULL_OB == (delete[*(incount)] = dt_xatom(dt_xatom(*atom, backbond), backbond))){
fprintf(stderr, "Whats up wit dat");
    }

/*    if(NULL_OB == (delete[*(incount)] = dt_copy(*atom))){
fprintf(stderr, "Error %d ", dt_uid(*atom));
dt_dealloc(bonds);
dt_dealloc(bond);
return(FALSE);
    }
*/
    ++*(incount);

/***loop through all connections to get more atoms to delete    ***/

    if(NULL_OB != (bonds = dt_stream(*atom, TYP_BOND))){
followct=0;
while(NULL_OB != (bond = dt_next(bonds))){
    if(bond != backbond){
followbond[followct]=bond;
followct++;
    } /* end if NOT backbond */
} /* end if bond */


for(count=0;count<followct;count++){
    xatom = dt_xatom(*atom, followbond[count]);

/***loop through delatoms making sure that you're not looping back on self***/

    repeat = FALSE;
    for(count1 = 0;count1 < *(incount);count1++){
if(xatom == delete[count1]){
    repeat = TRUE;
    break;
}
    }

/***if everythings ok, go to deeper level on recursion  ***/

    if(!repeat){
if(FALSE == deletefragment(followbond[count],   \
&xatom,  delete, incount)){
    fprintf(stderr, "%d, ", dt_uid(*atom));
    dt_dealloc(bonds);
    dt_dealloc(bond);
    return(FALSE);
}
    }

} /* end for followct */

    } /* end if bonds */

/***clean up after connections  ***/

    dt_dealloc(bonds);

    return(TRUE);

} /* end delete fragment */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "trans.h"
#include "mol.h"
#include "con.h"
#include "rep.h"
#include "debug.h"

extern int syncount;

/**********************************************************************
 * connect
 *
 * joins a molecule or fragment to a library of other fragments.
 **********************************************************************/
voidconnect(molequeue *Reactant, transform *Transform, molequeue **Lastptr)
{
    charparent[MAXSMILES];
    intsize,
newct,
copyct,
attach,
reactatom,
reactbond,
atomcount,
interbondct,
bondcount,
concount,
connumber,
xatomct,
count, count1,
frag,
max,
length,
realbond,
seqcount,
chival[MAXCONNECTIONS],
wildchival[MAXTEMPLATE];
    dt_Handleerror,
errors,
```

```
tempmol,
pathset,
paths,
path,
atoms, atoms1, atoms2, atom3, atom4,
atom, atom1, atom2,
bonds, bonds1, bonds2,
bond, bond1, bond2,
newbond,
copybonds[MAXCONNECTIONS],
copyatoms[MAXCONNECTIONS],
delatoms[MAXTEMPLATE],
delbonds[MAXTEMPLATE],
newatoms[MAXTEMPLATE],
newbonds[MAXTEMPLATE],
chiseq[MAXCONNECTIONS],
conatoms[10],
conbonds[10],
wildchiseq[MAXTEMPLATE];
   dt_Stringstr;
   molequeue*Product;

   if(screen(Reactant, Transform) == FALSE)
return;

/***malloc new molecule (Product)***/

   if(Transform->flag == 'I'){
size = 1,
   }
   else {
      size = Transform->type.con.Library->size;
   }
   if(size == 0){
      return;
   }

   if(NULL == (Product = (molequene *)malloc(sizeof(molequeue)*size))){
fprintf(stderr, "Out of memory in  con\n");
fprintf(stderr, "Unable to open molecules %d-%d\n", syncount, syncount+size);
fprintf(stderr, "Malloc call for size = %d\n",sizeof(molequeue)*size),
exit(1);
   }

/*****************************************************************************
 * In the following section the product will be created.
 * 1  duplicate reactant->temp
 * 2  find reactant set
 * 3. loop through reactant set
 *if appropriate add new atom
 *    loop through reactant bonds and add to product
 * 4. add rest of product atoms.
 * 5. add product bonds.
 * 6. delete reactant set.
 * 7. loop through library
 *duplicate temp->product
 *add fragment to product
 * 7. set hydrogens in new molecule
 * this should result in the product molecule
 *****************************************************************************/

/***copy reactant molecule to product molecule   ***/

   tempmol = dt_copy(Reactant->molecule);

/***make product molecule modifiable    ***/

   if(FALSE == dt_mod_on(tempmol)){
fprintf(stderr, "Can't modify %s in con\n", Reactant->smiles);
dt_dealloc(tempmol);
free(Product),
return,
   }

/***find reactant path in product molecule   ***/

   if(NULL_OB == (pathset = dt_umatch(Transform->screenin, tempmol, FALSE))){
fprintf(stderr, "Unable to find remove: %s of %s in %s\n", \
   Transform->inclusion, \
   Transform->name, \
   Reactant->smiles);
/***this should NEVER occur but is left in for historical reasons ***/
fprintf(stderr, "BIG problem, a reactant conatined screenin, but the product didn't.\n");
dt_dealloc(tempmol),
free(Product);
return;
   }

/***get atoms in screenin    ***/

   paths = dt_stream(pathset, TYP_PATH);
   path = dt_next(paths),
   atoms = dt_stream(path, TYP_ATOM);

/***loop through reactant set get atoms, bonds  ***/

   reactatom = 0;
   reactbond = 0;
   while(NULL_OB != (delatoms[reactatom] = dt_next(atoms))){
if(NULL_OB != (bonds = dt_stream(delatoms[reactatom], TYP_BOND))){
   while(NULL_OB != (delbonds[reactbond] = dt_next(bonds))){
reactbond++;
   }
}
dt_dealloc(bonds);
reactatom++;
   }
   dt_dealloc(atoms);

   dt_dealloc(pathset);
   dt_dealloc(paths);

   atomcount = 0;
   bondcount = 0;
   while(atomcount < reactatom){


/***if appropriate add product atom***/

if(Transform->type.con.productatom[atomcount].atomicno >= 0){

   if(Transform->type.con.productatom[atomcount].atomicno > 0){
newatoms[atomcount] = dt_addatom(tempmol,\
   Transform->type.con.productatom[atomcount].atomicno, 0);
dt_setaromatic(newatoms[atomcount],\
   Transform->type.con.productatom[atomcount].aromatic);
dt_setcharge(newatoms[atomcount],\
   Transform->type.con.productatom[atomcount].formalcharge);
   }

/***atomicno 0 is for wildcard, copy reactant atom***/

   wildchival[atomcount] = DX_CHI_NONE;
   if(Transform->type.con productatom[atomcount].atomicno == 0){
newatoms[atomcount] = dt_addatom(tempmol,\
   dt_number(delatoms[atomcount]), dt_imp_hcount(delatoms[atomcount]));
dt_setaromatic(newatoms[atomcount],\
   dt_aromatic(delatoms[atomcount]));
dt_setcharge(newatoms[atomcount],\
   dt_charge(delatoms[atomcount]));

/***copy wildcard atom chirality if it exists   ***/

wildchiseq[atomcount] = dt_alloc_seq();
if(TRUE == dt_imp_hcount(delatoms[atomcount])){
   atom1 = dt_isohydro();
   wildchiseq[atomcount] = dt_append(wildchiseq[atomcount], atom1);
}
if(NULL_OB != (bonds = dt_stream(delatoms[atomcount], TYP_BOND))){
   while(NULL_OB != (bond = dt_next(bonds))){
wildchiseq[atomcount] = dt_append(wildchiseq[atomcount], bond);
   } /* end while wildcard's bonds */
} /* end if bonds */
dt_dealloc(bonds);
dt_reset(wildchiseq[atomcount]);
wildchival[atomcount] = dt_chival(delatoms[atomcount], wildchiseq[atomcount]);

   } /* end if wildcard */

/***loop through reactant atom bonds and add to product ***/

   if(NULL_OB != (bonds = dt_stream(delatoms[atomcount], TYP_BOND))){
copyct = 0;
while(NULL_OB != (copybonds[copyct] = dt_next(bonds))){

/***only count bonds to unchanged atoms ***/

   realbond = TRUE;
   chival[copyct] = DX_CHI_NONE;
   copyatoms[copyct] = dt_xatom(delatoms[atomcount], copybonds[copyct]);
   for(count=0;count<reactatom;count++){
if(copyatoms[copyct] == delatoms[count]){
realbond = FALSE;
break;
}
   }

/***if bond is to non-delatom check for chirality in xatom    ***/

   if(realbond){
chiseq[copyct] = dt_alloc_seq();
if(TRUE == dt_imp_hcount(copyatoms[copyct])){
   atom1 = dt_isohydro(),
   chiseq[copyct] = dt_append(chiseq[copyct], atom1);
}
if(NULL_OB != (bonds1 = dt_stream(copyatoms[copyct], TYP_BOND))){
   while(NULL_OB != (bond1 = dt_next(bonds1))){
chiseq[copyct] = dt_append(chiseq[copyct], bond1);
   } /* end while non-delatom's bonds */
} /* end if bonds */
dt_dealloc(bonds1);
dt_reset(chiseq[copyct]);
chival[copyct] = dt_chival(copyatoms[copyct], chiseq[copyct]);
copyct++;
   } /* end if bond is real (ie-to a non-delatom) */

/***if atom on either end of copybond is chiral substitute newbond.    ***/

   else{
for(count=0;count<atomcount;count++){
   if(copyatoms[copyct] == delatoms[count]){
break;
   }
}
if((count<atomcount)&&((wildchival[atomcount] > DX_CHI_NONE)|| \
   (Transform->type con productatom[count].atomicno == 0))){
copyatoms[copyct] = newatoms[count],
    copyct++,
} /* end if wildchival */
   } /* end else */

} /* while there are copybonds */

/***make new bond, substituting chirality if necessary  ***/

for(count=0,count<copyct;count++){
   newbonds[bondcount] = dt_addbond(newatoms[atomcount],\
```

```
   copyatoms[count], dt_bondtype(copybonds[count]));

/***if atom across bond is chiral reset chirality***/

   if(chival[count] > DX_CHI_NONE){
dt_reset(chiseq[count]);
while(NULL_OB != (bond = dt_next(chiseq[count]))){
   if(bond == copybonds[count]){
if(FALSE == dt_delete(chiseq[count])){
   fprintf(stderr, "trouble repairing non-delatom chirality in %s.\n\n", Transform->name);
   }
dt_next(chiseq[count]);
chiseq[count] = dt_insert(chiseq[count], newbonds[bondcount]);
dt_reset(chiseq[count]);
dt_dealloc(copybonds[count]);
if(FALSE == dt_setchival(copyatoms[count], chiseq[count], chival[count])){
/*   fprintf(stderr, "trouble resetting non-delatom chirality in %s.\n\n", Transform->name);*/
}
break;
   }
}
   } /* end if xatom is chiral */

/***if atom across bond is wildcard reset chirality***/

   xatomct = -1;
   for(count1=0;count1<reactatom;count1++){
if(copyatoms[count] == newatoms[count1]){
   xatomct = count1;
   break;
}
   }
   if((xatomct >= 0)&&(wildchival[count1] > DX_CHI_NONE)){
dt_reset(wildchiseq[count1]);
while(NULL_OB != (bond = dt_next(wildchiseq[count1]))){
   if(bond == copybonds[count]){
if(FALSE == dt_delete(wildchiseq[count1])){
/*   fprintf(stderr, "trouble repairing non-delatom chirality in %s.\n\n", Transform->name);*/
   }
dt_next(wildchiseq[count1]);
wildchiseq[count1] = dt_insert(wildchiseq[count1], newbonds[bondcount]);
dt_reset(wildchiseq[count1]);
dt_dealloc(copybonds[count]);
if(FALSE == dt_setchival(copyatoms[count], wildchiseq[count1], wildchival[count1])){
/*   fprintf(stderr, "trouble resetting non-delatom chirality in %s.\n\n", Transform->name);*/
}
break;
   }
}
   } /* end if xatom is chiral */

/***if current atom is a chiral wildcard reset chirality***/

   if(wildchival[atomcount] > DX_CHI_NONE){
dt_reset(wildchiseq[atomcount]);
while(NULL_OB != (bond = dt_next(wildchiseq[atomcount]))){
   if(bond == copybonds[count]){
if(FALSE == dt_delete(wildchiseq[atomcount])){
/*   fprintf(stderr, "trouble repairing wildcard chirality in %s.\n\n", Transform->name);*/
   }
dt_next(wildchiseq[atomcount]);
wildchiseq[atomcount] = dt_insert(wildchiseq[atomcount], newbonds[bondcount]);
dt_reset(wildchiseq[atomcount]);
if(FALSE == dt_setchival(newatoms[atomcount], wildchiseq[atomcount], wildchival[atomcount])){
/*   fprintf(stderr, "Trouble resetting wildcard atom chirality in %s.\n\n", Transform->name); */
}
else{
}
break;
   }
}

   } /* end if wildcard is chiral */

/***if newbond was made, increment bondcount   ***/

   if(NULL_OB != newbonds[bondcount]){
bondcount++;
   }
} /* end for make newbonds */
for(count=0;count<copyct;count++){
   dt_dealloc(chiseq[count]);
}

   } /* end loop through reactant atom's bonds */
   dt_dealloc(bonds);

} /* end if atomicno >0 */

atomcount++;
   } /* end while reactant set atom */

/**************************************************************************
***************************************************************************
**************************do not edit above this line***********************
***************************************************************************
**************************************************************************/

/***add rest of the product atoms***/

   for(atomcount = atomcount;atomcount < Transform->type.con.atomnumber;atomcount++){
newatoms[atomcount] = dt_addatom(tempmol,\
Transform->type.con.productatom[atomcount].atomicno, 0);
dt_setaromatic(newatoms[atomcount],\
Transform->type.con.productatom[atomcount].aromatic);
dt_setcharge(newatoms[atomcount],\
Transform->type.con.productatom[atomcount].formalcharge);
   }
```

```
/***add rest of the product bonds  ***/

   concount = 0;
   interbondct = bondcount;
   for(bondcount = 0;bondcount < Transform->type.con.bondnumber;bondcount++){
if((Transform->type.con.productbond[bondcount].iatom1 < 0xffff)&&   \
  (Transform->type.con.productbond[bondcount].iatom2 < 0xffff)){
if(NULL_OB == (newbonds[bondcount+interbondct] = dt_bond(newatoms[Transform->type.con.pro-
ductbond[bondcount].iatom1],\
 newatoms[Transform->type.con.productbond[bondcount].iatom2]))){
newbonds[bondcount+interbondct] = dt_addbond(newatoms[Transform->type.con.productbond[bond-
count].iatom1],   \
   newatoms[Transform->type.con.productbond[bondcount].iatom2],   \
   Transform->type.con.productbond[bondcount].bondtype);
}
else{
   dt_setbondtype(newbonds[bondcount+interbondct], Transform->type.con.productbond[bond-
count].bondtype);
}
}
else{
   if(Transform->type.con.productbond[bondcount].iatom1 > 0xffff){
connumber = (Transform->type.con.productbond[bondcount].iatom1) >> 16;
atom2 = newatoms[Transform->type.con.productbond[bondcount].iatom2];
   }
   else{
connumber = (Transform->type.con.productbond[bondcount].iatom2) >> 16;
atom2 = newatoms[Transform->type.con.productbond[bondcount].iatom1];
   }
   if(NULL_OB == (conatoms[connumber] = dt_addatom(tempmol, 0, connumber))){
fprintf(stderr, "\nProblem adding conatom %d in transform %s.\n\n", connumber, Transform->name);
   }
   if(NULL_OB == (conbonds[connumber] = dt_addbond(conatoms[connumber], atom2,   \
Transform->type.con.productbond[bondcount].bondtype))){
fprintf(stderr, "\nProblem making conbond %d in transform %s.\n\n", connumber, Transform->name);
   }
   newbonds[bondcount+interbondct] = conbonds[connumber];
   concount++;
}
   } /* end adding product bonds */


/***delete reactant atoms and bonds***/

   for(count = 0;count < reactatom;count++){
dt_dealloc(delatoms[count]);
   }
   for(count = 0;count < reactbond;count++){
dt_dealloc(delbonds[count]);
   }

/***   set number of implicit hydrogens for each atom in base    ***/

for(count=0;count < Transform->type.con.atomnumber;count++){
   dt_setimp_hcount(newatoms[count], hcount(&(newatoms[count])));
}

/*******************************************************************
 **********************start from rep******************************
 ******************************************************************/
/***set chirality on chiral transform atoms   ***/

   for(count=0;count < Transform->type.con.atomnumber;count++){
if(Transform->type.con.productatom[count].chival != DX_CHI_NONE){

/***get chiral sequence of bonds***/

   chiseq[0] = dt_alloc_seq();
   for(seqcount=0;seqcount < MAXCHIRAL;seqcount++){
if(Transform->type.con.productatom[count].chiseq[seqcount] == DIV_IMP_HYDROGEN){
   atom = dt_isohydro();
   chiseq[0] = dt_append(chiseq[0], atom);
}
else if(Transform->type.con.productatom[count].chiseq[seqcount] >= 0){
   chiseq[0] = dt_append(chiseq[0],\
newbonds[(Transform->type.con.productatom[count].chiseq[seqcount])+interbondct]);
}
else{
   break;
}
   }

/***set chirality of atom***/

   if(FALSE == dt_setchival(newatoms[count], chiseq[0],\
   Transform->type.con.productatom[count].chival)){
/*fprintf(stderr, "Error setting chirality of atom in 1to1.\n");*/
fprintf(stderr, "Transform: %s, Productatom: %d\n", Transform->name, count);
fprintf(stderr, "chival = %d.\n", Transform->type.con.productatom[count].chival);
debug_atom(newatoms[count]);
debug_sequence(chiseq[0]);
   }
   dt_dealloc(chiseq[0]);
   dt_dealloc(atom);
}
   }

/***set chirality of bonds***/

   for(count=0;count<Transform->type.con.bondnumber;count++){
if((Transform->type.con.productbond[count].chival == DX_CHI_CIS)||  \
  (Transform->type.con.productbond[count].chival == DX_CHI_TRANS)){
  if(FALSE == dt_setdbo(newbonds[count+interbondct],   \
newbonds[(Transform->type.con.productbond[count].chiseq[0])+interbondct], \
newbonds[(Transform->type.con.productbond[count].chiseq[1])+interbondct], \
Transform->type.con.productbond[count].chival)){
fprintf(stderr, "Error setting double bond chirality in connect.\n");
```

```
fprintf(stderr, "Transform: %s, Productbond: %d\n", Transform->name, count);
fprintf(stderr, "chival = %d \n", Transform->type.con.productbond[count].chival);
errors = dt_errors(DX_ERR_NOTE);
while(NULL_OB != (error = dt_next(errors))){
   str = dt_stringvalue(&length, error);
   fprintf(stderr, "%s \n", str);
}
   }
  }
   }

/*****************************************************************
 *************************end from rep***************************
 ****************************************************************/
/***for library transforms can exit now***/

   if(Transform->flag == 'l'){
dt_dealloc(Reactant->molecule);
if(NULL_OB == (Reactant->molecule = dt_copy(tempmol))){
   fprintf(stderr, "test error\n");
}
dt_dealloc(tempmol);
dt_dealloc(pathset);
dt_dealloc(paths);
dt_dealloc(path);
dt_dealloc(atoms);
dt_dealloc(bonds);
free(Product);
for(count=0;count<reactatom;count++){
   if(Transform->type.con.productatom[count].atomicno == 0){
dt_dealloc(wildchiseq[count]);
   }
}

return;
   }

/***loop through library making many products   ***/

   for(frag=0;frag < Transform->type.con.Library->size;frag++){

/***copy fragment***/

Product[frag].molecule = dt_copy(tempmol);

/***make adjuncts***/

atoms1 = dt_stream(tempmol, TYP_ATOM);
atoms2 = dt_stream(Product[frag].molecule, TYP_ATOM);
while(NULL_OB != (atom1 = dt_next(atoms1))){
   atom2 = dt_next(atoms2);
   dt_setadjunct(atom1, atom2);
}
bonds1 = dt_stream(tempmol, TYP_BOND);
bonds2 = dt_stream(Product[frag].molecule, TYP_BOND);
while(NULL_OB != (bond1 = dt_next(bonds1))){
   bond2 = dt_next(bonds2);
   dt_setadjunct(bond1, bond2);
}
dt_dealloc(atoms1);
dt_dealloc(atoms2);
dt_dealloc(bonds1);
dt_dealloc(bonds2);

/***copy stream of atoms from fragment  ***/

newct = 0;
if(NULL_OB != (atoms = dt_stream(Transform->type.con.Library->Fragments[frag], TYP_ATOM))){
   while(NULL_OB != (atom = dt_next(atoms))){
newatoms[newct] = dt_addatom(Product[frag].molecule, dt_number(atom), dt_hcount(atom));
if(dt_aromatic(atom)==TRUE){
   if(FALSE ==(dt_setaromatic(newatoms[newct], TRUE))){
fprintf(stderr, "Error making %d aromatic with %s\n",\
   dt_uid(newatoms[newct]), Transform->name);
   }
}
dt_setadjunct(atom, newatoms[newct]);
newct++;
   }
}
dt_dealloc(atoms);

/***copy stream of bonds from fragment  ***/

if(NULL_OB != (bonds = dt_stream(Transform->type.con.Library->Fragments[frag], TYP_BOND))){
   while(NULL_OB != (bond = dt_next(bonds))){
if(NULL_OB != (atoms = dt_stream(bond, TYP_ATOM))){
   atom1 = dt_next(atoms);
   atom2 = dt_next(atoms);
   atom3 = dt_adjunct(atom1);
   atom4 = dt_adjunct(atom2);
   newbond = dt_addbond(atom3, atom4, dt_bondtype(bond));
}
dt_setadjunct(bond, newbond);
dt_dealloc(atoms);
   }
   dt_dealloc(bonds);
}

/***copy chirality of atoms from fragment***/

if(NULL_OB != (atoms = dt_stream(Transform->type.con.Library->Fragments[frag], TYP_ATOM))){
   while(NULL_OB != (atom = dt_next(atoms))){
chiseq[0] = dt_alloc_seq();
chiseq[1] = dt_alloc_seq();
if(TRUE == dt_imp_hcount(atom)){
   atom1 = dt_isohydro();
   chiseq[0] = dt_append(chiseq[0], atom1);
   chiseq[1] = dt_append(chiseq[1], atom1);
}
if(NULL_OB != (bonds = dt_stream(atom, TYP_BOND))){
   while(NULL_OB != (bond = dt_next(bonds))){
bond1 = dt_adjunct(bond);
chiseq[0] = dt_append(chiseq[0], bond);
chiseq[1] = dt_append(chiseq[1], bond1);
   }
}
dt_dealloc(bonds);
dt_reset(chiseq[0]);
dt_reset(chiseq[1]);
chival[0] = dt_chival(atom, chiseq[0]);
chival[1] = chival[0];
if(chival[0] > DX_CHI_NONE){
   atom = dt_adjunct(atom);
   if(FALSE == dt_setchival(atom, chiseq[1], chival[1])){
/*fprintf(stderr, "Trouble copying chirality in fragment %d of library %s \n\n", frag, Transform-
>type.con.Library->name);*/
   }
}
dt_dealloc(chiseq[0]);
dt_dealloc(chiseq[1]);
   }
}
dt_dealloc(atoms);

/***copy chirality of bonds from fragment***/

if(NULL_OB != (bonds = dt_stream(Transform->type.con.Library->Fragments[frag], TYP_BOND))){
   while(NULL_OB != (bond = dt_next(bonds))){
if(DX_BTY_DOUBLE == dt_bondtype(bond)){
   if(NULL_OB != (atoms = dt_stream(bond, TYP_ATOM))){
atom1 = dt_next(atoms);
if(0 < dt_imp_hcount(atom1)){
   bond1 = dt_isohydro();
}
else{
   bonds1 = dt_stream(atom1, TYP_BOND);
   while(NULL_OB != (bond1 = dt_next(bonds1))){
if(bond != bond1)
   break;
   }
}
atom2 = dt_next(atoms);
if(0 < dt_imp_hcount(atom2)){
   bond2 = dt_isohydro();
}
else{
   bonds2 = dt_stream(atom2, TYP_BOND);
   while(NULL_OB != (bond2 = dt_next(bonds2))){
if(bond != bond2)
   break;
   }
}
   }
   if(DX_CHI_NO_DBO < (chival[0] = dt_dbo(bond, bond1, bond2))){
bond = dt_adjunct(bond);
if(DX_BTY_UNKNOWN < dt_bondtype(bond1)){
   bond1 = dt_adjunct(bond1);
}
if(DX_BTY_UNKNOWN < dt_bondtype(bond2)){
   bond2 = dt_adjunct(bond2);
}
if(FALSE == dt_setdbo(bond, bond1, bond2, chival[0])){
   fprintf(stderr, "Trouble copying double bond chirality in fragment %d of library %s \n\n", frag,
Transform->type.con.Library->name);
}
   } /* end found chiral dbl */
} /* found dbl bond */
dt_dealloc(atoms);
dt_dealloc(bonds1);
dt_dealloc(bonds2);
   }
   dt_dealloc(bonds);
}

/***make the new bond which connects fragment to base ***/
/***need to change to to handle multiple connections   ***/

for(connumber=1;connumber<=concount;connumber++){
   atom = dt_xatom(conatoms[connumber], conbonds[connumber]);
   atom1 = dt_adjunct(atom);
   atom2 = dt_adjunct(Transform->type.con.Library->Attachatoms[frag][connumber]);

/***copy initial chirality   ***/

   chiseq[1] = dt_alloc_seq();
   if(TRUE == dt_imp_hcount(atom1)){
atom = dt_isohydro();
   }
   if(NULL_OB != (bonds = dt_stream(atom1, TYP_BOND))){
while(NULL_OB != (bond = dt_next(bonds))){
   chiseq[1] = dt_append(chiseq[1], bond);
}
   }
   dt_reset(chiseq[1]);
   chival[1] = dt_chival(atom1, chiseq[1]);
   dt_dealloc(bonds);

   chiseq[2] = dt_alloc_seq();
   if(TRUE == dt_imp_hcount(atom2)){
atom = dt_isohydro();
   }
   if(NULL_OB != (bonds = dt_stream(atom2, TYP_BOND))){
while(NULL_OB != (bond = dt_next(bonds))){
   chiseq[2] = dt_append(chiseq[2], bond);
}
```

```
   }
   dt_reset(chiseq[2]);
   chival[2] = dt_chival(atom2, chiseq[2]);
   dt_dealloc(bonds);

/***make new bond   ***/

   if(NULL_OB == (newbond = dt_addbond(atom1, atom2, dt_bondtype(conbonds[connumber])))){
debug_atom(atom1);
debug_atom(atom2);
fprintf(stderr, "\nTrouble making connection bond %d in %s.\n\n", connumber, Transform->name);
   }

/***if necessary reset chirality***/

   if(chival[1] > DX_CHI_NONE){
bond1 = dt_adjunct(conbonds[connumber]);
while(NULL_OB != (bond = dt_next(chiseq[1]))){
   if(bond == bond1){
dt_delete(chiseq[1]);
dt_next(chiseq[1]);
chiseq[1] = dt_insert(chiseq[1], newbond);
break;
   }
}
dt_dealloc(bond1);
dt_reset(chiseq[1]);
if(FALSE == dt_setchival(atom1, chiseq[1], chival[1])){
/*   fprintf(stderr, "\nTrouble resetting chirality of root connect atom %d in %s.\n\n", connumber,
Transform->name);*/
}
   }
   if(chival[2] > DX_CHI_NONE){
/* need to have bond1 = Transform->type.con.Library->Attachbonds[frag][connumber] eventually */
while(NULL_OB != (bond = dt_next(chiseq[2]))){
   if(bond == bond1){
dt_delete(chiseq[2]);
dt_next(chiseq[2]);
chiseq[2] = dt_insert(chiseq[2], newbond);
break;
   }
}
dt_dealloc(bond1);
dt_reset(chiseq[2]);
if(FALSE == dt_setchival(atom2, chiseq[2], chival[2])){
/*   fprintf(stderr, "\nTrouble resetting chirality of root connect atom %d in %s.\n\n", connumber,
Transform->name);*/
}
   }
   dt_dealloc(chiseq[1]);
   dt_dealloc(chiseq[2]);

/***clean up connection place-holders   ***/

   atom = dt_adjunct(conatoms[connumber]);
   dt_dealloc(atom);

} /* end making connections */

/***   set number of implicit hydrogens for attachment site atoms ***/

dt_setimp_hcount(atom2, hcount(&(atom2)));

/***set number of implicit hydrogens for fragment atoms ***/
/*
atoms = dt_stream(Transform->type.con.Library->Fragments[frag], TYP_ATOM);
while(NULL_OB != (atom = dt_next(atoms))){
   atom = dt_adjunct(atom);
   dt_setimp_hcount(atom, hcount(&(atom)));
}
dt_dealloc(atoms);
*/
/***initialize product mmolequeue and reorder queue***/

if(FALSE == (dt_mod_off(Product[frag].molecule))){
   fprintf(stderr, "Unable to turn modify off in connect.\n");
   fprintf(stderr, "Reactant: %s  Fragment: %s\n", Reactant->smiles, Transform->type.con.Library-
>Externalids[frag]);
debug_molecule(Product[frag].molecule);
debug_molecule(Transform->type.con.Library->Fragments[frag]);
debug_molecule(tempmol);

/***clean up after reaction   ***/

   dt_dealloc(pathset);
   dt_dealloc(paths);
   dt_dealloc(atoms);
   dt_dealloc(bonds);
   for(count=0;count < newct;count++){
   dt_dealloc(newatoms[count]);
   }
   for(count=0;count<reactatom;count++){
if(Transform->type.con.productatom[count].atomicno == 0){
   dt_dealloc(wildchiseq[count]);
}
   }
   return;
} /* end if FALSE */

strncpy(Product[frag].smiles, dt_cansmiles(&length, Product[frag].molecule, TRUE), length);
Product[frag].smiles[length] = '\0';
strcpy(Product[frag].externalid, Reactant->externalid);
if(strlen(Product[frag].externalid)+\
  strlen(Transform->type.con.Library->Externalids[frag]) < MAXSMILES){
   strcat(Product[frag].externalid, Transform->type.con.Library->Externalids[frag]);
}
/*fprintf(stderr, "%d %.*s\n", syncount, strlen(Product[frag].smiles), Product[frag].smiles);*/
sprintf(parent, "%d", Reactant->synno);
strcpy(Product[frag].parent, parent);
strcat(Product[frag].parent, ":");
strcat(Product[frag].parent, Reactant->smiles);
Product[frag].identical = -1;
strcpy(Product[frag].transform, Transform->name);
Product[frag].Next = NULL;
Product[frag].Previous = *Lastptr;
Product[frag].depth = Reactant->depth + 1;
Product[frag].synno = syncount;
syncount++;
Product[frag].nochildren = 0;


Reactant->children[Reactant->nochildren] = Product[frag].synno;
Reactant->nochildren++;

(*Lastptr)->Next = &(Product[frag]);
*Lastptr = &(Product[frag]);

   } /* end loop through fragments */

/***clean up after reaction   ***/

   for(count=0;count<reactatom;count++){
if(Transform->type.con.productatom[count].atomicno == 0){
   dt_dealloc(wildchiseq[count]);
}
   }
   dt_dealloc(tempmol);
   dt_dealloc(atoms);
   dt_dealloc(bonds);

   return;
}
#include <stdio.h>
#include <stddef.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include "trans.h"
#include "parsesmiles.h"

/*******************************************************************
 * parsesmiles
 *
 * ags 4/25/96
 *
 * this routine is a pseudo-smiles parser. it currently does not handle
 * nested branching ( ie - CCC(NCC(Br)C)CC ) correctly.  this can be handled by
 * disconnection and 'ring' closure ( ie - CCC1CC.N1CC(Br)C ).
 * ***can fix this by putting openbranchbond on a stack and pop/push with ) and (
 * this
 * routine handles the special characters X, Y, and Z which stand for
 * atom termination, connection point, and separation point respectively.
 *******************************************************************/
int parsesmiles(char *string, atoms *newatoms, int *atomnumber, bonds *fragbonds, int *bondnumber)
{
   int   chiralnumber;

/***get atoms   ***/

   if(0 > (*atomnumber = parseatoms(string, newatoms))){
fprintf(stderr, "Error parsing atoms in %s\n", string);
return(FALSE);
   }

/***get bonds   ***/

   if(0 > (*bondnumber = parsebonds(string, newatoms, *atomnumber, fragbonds))){
fprintf(stderr, "Error parsing bonds in %s\n", string);
return(FALSE);
   }

/***set chirality***/

   if(0 > (chiralnumber = setchiral(string, newatoms, *atomnumber, fragbonds, *bondnumber))){
fprintf(stderr, "Error parsing chiral centers in %s\n", string);
return(FALSE);
   }

/***done parsing return ***/

   return(TRUE);

} /* end parsesmiles */

/*******************************************************************
 * parseatoms
 *
 * 4/25/96 ags
 *
 * this routine takes a smiles-transform string and gets a list of
 * the atoms out.
 *******************************************************************/
int parseatoms(char *string, atoms *atoms)
{
   int   max = 0,
   count = 0,
   icount,
   chicount,
   atomcount = 0,
   inbracket = FALSE,
   bracketsize = 0,
   branchlevel = 0;
   char   aletter[53];

   strcpy(aletter, "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz*");
   max = strlen(string);
```

```
    while(count < max){

/***check if atom is inside a bracket   ***/

if(string[count] =='[' ){
    inbracket = TRUE;
    while(string[count+1+bracketsize] != ']'){
bracketsize++;
    }
}

/***check if now outside bracket***/

if(string[count] == ']'){
    inbracket = FALSE,
    bracketsize = 0,
}

/***increment branch level on (***/

if(string[count] == '('){
    branchlevel++;
}

/***decrement branch level on )***/

if(string[count] == ')'){
    branchlevel--;
}

/***check for and build atom symbols   ***/

/*
if(((string[count] >= 'A')&&(string[count] <= 'Z'))||\
  ((string[count] >= 'a')&&(string[count] <= 'z'))){
*/

if(NULL != strchr(aletter, string[count])){

/***note position of atom in string   ***/

    atoms[atomcount].position = count;
    atoms[atomcount].inbracket = inbracket;
    atoms[atomcount].bracketsize = bracketsize;
    atoms[atomcount].branchlevel = branchlevel;

/***parse atom symbol into atom structure***/

/**** is wildcard atom***/

    if(string[count] == '*'){
atoms[atomcount].symbol = '*';
atoms[atomcount].atomicno = DX_ATN_WILD;
atoms[atomcount].aromatic = 0;
    }
/***Cl and Br are special cases***/

    if((string[count+1] == 'r')||(string[count+1] == 'l')){
if(string[count] == 'B'){
    count++;
    atoms[atomcount].symbol = 'R';
    atoms[atomcount].atomicno = 35;
    atoms[atomcount].aromatic = 0;
}

    else if(string[count] == 'C'){
    count++,
    atoms[atomcount].symbol = 'L',
    atoms[atomcount].atomicno = 17;
    atoms[atomcount].aromatic = 0;
}
    }

/***all other atoms   ***/

    else {
atoms[atomcount].symbol = string[count],
atoms[atomcount].atomicno = atno(&string[count]);
if((string[count] >= 'a')&&(string[count] <= 'z')){
    atoms[atomcount].aromatic = TRUE;
}
else{
    atoms[atomcount].aromatic = FALSE,
}
    }

/***general atom initializatoin***/

    atoms[atomcount].formalcharge = 0;
    atoms[atomcount].chival = DX_CHI_NONE;

/***initialize ibond   ***/

    for(icount=0;icount<MAXIBOND;icount++){
atoms[atomcount].ibond[icount] = -1,
    }

/***check to see if extra information is coming***/

    if(inbracket){

count++;

/***set formal charge of each atom, either [C++] or [C+2] format***/

while(string[count] == '+'){
    if((string[count+1] >= '1')&&(string[count+1] <= '9')){
atoms[atomcount].formalcharge += (string[count+1] - 48);
count += 2;
    }
    else{
atoms[atomcount].formalcharge++;
count++;
    }
}
while(string[count] == '-'){
    if((string[count+1] >= '1')&&(string[count+1] <= '9')){
atoms[atomcount].formalcharge -= (string[count+1] - 48);
count += 2;
    }
    else{
atoms[atomcount].formalcharge--;
count++;
    }
}

/***get chirality***/
/***allow TH as default [C@] or [C@@]   ***/
/***allow AL, SP, TB, or OH but only as stated case (ie-NOT by degree defaults) ***/
/***allow [C@@XX] to by synonymous with [C@XX2]***/
/***allow [C@@@SP] to by synonymous with [C@SP3]***/
/***only AL1-2, SP1-3, TB1-2, and OH1-2 are allowed, TB1-20 and OH1-30 exist   ***/

chicount = 0;
while(string[count] == '@'){
    chicount++;
    count++;
}

if((string[count] == 'A')||(string[count] == 'S')||\
  (string[count] == 'T')||(string[count] == 'O')){
    switch(string[count]){

/***Alene chirality   ***/

case 'A':
    count += 2;
    if(chicount == 1){
if(string[count] == '1'){
    atoms[atomcount].chival = _CHI_VAL(DIV_CHI_AL, 1);
    count++;
}
else if(string[count] == '2'){
    atoms[atomcount].chival = _CHI_VAL(DIV_CHI_AL, 2);
    count++;
}
else{
    atoms[atomcount].chival = _CHI_VAL(DIV_CHI_AL, 1);
}
    }
    else{
atoms[atomcount].chival = _CHI_VAL(DIV_CHI_AL, 2);
    }
    break;

/***square-planar chirality   ***/

case 'S':
    count += 2;
    if(chicount == 1){
if(string[count] == '1'){
    atoms[atomcount].chival = _CHI_VAL(DX_CHI_SP, 1);
    count++;
}
else if(string[count] == '2'){
    atoms[atomcount].chival = _CHI_VAL(DX_CHI_SP, 2);
    count++;
}
else if(string[count] == '3'){
    atoms[atomcount].chival = _CHI_VAL(DX_CHI_SP, 3);
    count++;
}
else{
    atoms[atomcount].chival = _CHI_VAL(DX_CHI_SP, 1);
}
    }
    else if(chicount == 2){
atoms[atomcount].chival = _CHI_VAL(DX_CHI_SP, 2);
    }
    else{
atoms[atomcount].chival = _CHI_VAL(DX_CHI_SP, 3);
    }
    break;

/***Octachedral chirality   ***/

case 'O':
    count += 2;
    if(chicount == 1){
if(string[count] == '1'){
    atoms[atomcount].chival = _CHI_VAL(DX_CHI_OH, 1);
    count++;
}
else if(string[count] == '2'){
    atoms[atomcount].chival =  _CHI_VAL(DX_CHI_OH, 2);
    count++;
}
else{
    atoms[atomcount].chival =  _CHI_VAL(DX_CHI_OH, 1);
}
    }
    else{
atoms[atomcount].chival =  _CHI_VAL(DX_CHI_OH, 2);
    }
    break;
```

```
case 'T':
   count++;
   switch(string[count]){

/***Trigonal-bipyramidal chirality    ***/

case 'B'
   count++;
   if(chicount == 1){
if(string[count] == '1'){
   atoms[atomcount].chival =  _CHI_VAL(DX_CHI_TB, 1);
   count++,
}
else if(string[count] == '2'){
   atoms[atomcount].chival =  _CHI_VAL(DX_CHI_TB, 2);
   count++,
}
else{
   atoms[atomcount].chival =  _CHI_VAL(DX_CHI_TB, 1);
}
   }
   else{
atoms[atomcount].chival =  _CHI_VAL(DX_CHI_TB, 2);
   }
   break;

/***Tetrahedral chirality    ***/

case 'H'
   count++,
   if(chicount == 1){
if(string[count] == '1'){
   atoms[atomcount] chival = DX_CHI_THCCW;
   count++,
}
else if(string[count] == '2'){
   atoms[atomcount].chival = DX_CHI_THCW;
   count++,
}
else{
   atoms[atomcount].chival = DX_CHI_THCCW,
}
   }
   else{
atoms[atomcount] chival = DX_CHI_THCW;
   }
   break;
default
   break,
   } /* end switch BH */
default.
   break,
   } /* end switch  ASPT */

} /* end if ASTO */

/***tetrahedral without any specification is the default***/

else if(chicount '= 0){
   if(chicount == 1){
atoms[atomcount].chival = DX_CHI_THCCW,
   }
   if(chicount == 2){
atoms[atomcount].chival = DX_CHI_THCW;
   }
}
else{
   atoms[atomcount].chival = DX_CHI_NONE;
}

/***look for explicit hydrogens, but ignore them***/

if(string[count] == 'H'){
   count++,
}

count--, /* to catch ] */

   } /* end bracket */

/***finish atom***/

   atomcount++;

} /* end if alphanumeric*/

   count++,

   } /* end while count */

   return(atomcount), /* returns the number of independent atoms found */

} /* end parse atom */

/**********************************************************************
 * atno
 *
 * returns the atomic number for a character
 **********************************************************************/
int atno(char *id)
{
   typedef struct {
charsymbol;
intatomicnumber;
   }convert,
   convert array[] = {
'X', -1,   /* X for 1:1 atoms from reactant that are not in product */
'Y', -2,
'Z', -3,
'H', 1,
'B', 5,
'C', 6,
'c', 6,
'N', 7,
'n', 7,
'O', 8,
'o', 8,
'F', 9,
'P', 15,
'S', 16,
's', 16,
'L', 17,   /*Cl = L*/
'R', 35,   /*Br = R*/
'I', 53,
NULL, NULL
   };
   int count = 0;
   while((array[count] symbol != *id)&& \
 (array[count].symbol != NULL)){
      count++;
   }
   return(array[count].atomicnumber);
} /* end atno */

/**********************************************************************
 * parsebonds
 *
 * 4/29/96 ags
 *
 * this routine takes a smiles-transform string and gets a list of the
 * bonds out, along with standard bond properties; order, connections...
 **********************************************************************/
int parsebonds(char *string, atoms *atoms, int maxatoms, bonds *bonds)
{

   int   max = 0,
   bondcount = 0,
   count = 0,
   counter,
   separation,
   nonexplicit,
   bondclass;

   max = strlen(string);

/***get implicit bonds (atoms without explicit bonds or disconnects between them)***/

   for(count=0;count < maxatoms-1;count++){

/***find out-of-bracket separation between atoms***/

separation = atoms[count+1].position - atoms[count+1].inbracket   \
  - atoms[count].position - atoms[count].bracketsize;

/***count ring and external bond separation    ***/

counter = atoms[count] position + atoms[count].bracketsize + 1;
while(counter < atoms[count+1] position){
   if((isdigit(string[counter]))||(string[counter]=='&')){
counter++;
   }
   else if(isdigit(string[counter+1])){
counter++;
   }
   else{
break,
   }
}
nonexplicit = counter - atoms[count].position - atoms[count].bracketsize;

/***compare 2 separations to see if atoms have implicit bond    ***/

if(separation == nonexplicit){
   if((atoms[bonds[bondcount].iatom1].aromatic == TRUE)&&\
     (atoms[bonds[bondcount].iatom2].aromatic == TRUE)){
   bondcount += makebond(DIV_BCLS_IMPLICIT, bondcount, atoms, maxatoms, bonds,   \
   string, count, DX_BTY_AROMAT, DX_CHI_NONE);
   }
   else{
   bondcount += makebond(DIV_BCLS_IMPLICIT, bondcount, atoms, maxatoms, bonds,   \
   string, count, DX_BTY_SINGLE, DX_CHI_NONE);
   }
}
   }

/***get explicit single, double, triple, branch, and directional single bonds    ***/

   count = 0;

/*
   while(count < max){
if((string[count] == '=')||(string[count] == '\')||\
  (string[count] == '/')||(string[count] == '#')||\
  (string[count] == '(')||(string[count] == ')')){
*/

   while(max != (count += strcspn(string+count, "-=\\/#():"))){

/***exclude explicits inside brackets (which can be charges)  ***/

counter = count + strspn(string+count, "0123456789-:/\\=#H+");
if((counter < max)&&(string[counter] == ']')){
   count++;
   continue;
}
```

```
/***if not a ring or external bond, make different bond types   ***/

   if((string[count+1] < '0')||(string[count+1] > '9')){
switch(string[count]){

/***- is an explicit single bond***/

   case '-':
bondcount += makebond(DIV_BCLS_EXPLICIT, bondcount, atoms, maxatoms, bonds,\
string, count, DX_BTY_SINGLE, DX_CHI_NONE);
break;

/***= is a double bond***/

   case '=':
bondcount += makebond(DIV_BCLS_EXPLICIT, bondcount, atoms, maxatoms, bonds,\
string, count, DX_BTY_DOUBLE, DX_CHI_NONE);
break;

/***\ is a directional single bond   ***/

   case '\\':
bondcount += makebond(DIV_BCLS_EXPLICIT, bondcount, atoms, maxatoms, bonds,\
string, count, DX_BTY_SINGLE, DIV_CHI_BACKWARD);
break;

/***/ is the other directional single bond   ***/

   case '/':
bondcount += makebond(DIV_BCLS_EXPLICIT, bondcount, atoms, maxatoms, bonds,\
string, count, DX_BTY_SINGLE, DIV_CHI_FORWARD);
break;

/***# is a triple bond***/

   case '#':
bondcount += makebond(DIV_BCLS_EXPLICIT, bondcount, atoms, maxatoms, bonds,\
string, count, DX_BTY_TRIPLE, DX_CHI_NONE);
break;

/***( opens a branch chain   ***/

   case '(':
if(atoms[postatom(atoms, maxatoms, count)].position -   \
  atoms[postatom(atoms, maxatoms, count)].inbracket == count+1){
   bondcount += makebond(DIV_BCLS_EXPLICIT, bondcount, atoms, maxatoms, bonds,   \
   string, count, DX_BTY_SINGLE, DX_CHI_NONE);
}
break;

/***) closes a branch chain   ***/

   case ')':
if(atoms[postatom(atoms, maxatoms, count)].position -   \
  atoms[postatom(atoms, maxatoms, count)].inbracket == count+1){
      bondcount += makebond(DIV_BCLS_EXPLICIT, bondcount, atoms, maxatoms, bonds,   \
   string, count, DX_BTY_SINGLE, DX_CHI_NONE);
}
break;

/***: for explicit aromatic bond***/

   case ':':
bondcount += makebond(DIV_BCLS_EXPLICIT, bondcount, atoms, maxatoms, bonds,\
string, count, DX_BTY_AROMAT, DX_CHI_NONE);
break;

/***it should prevent defaults***/

   default:
fprintf(stderr, "Fatal Error in parsing explicit bonds.\n\n");
exit(1);
break;
} /* end switch */

   } /* end if not ringbond */

   count++;

/*} *//* end if */

   } /* end while */

/***parse ring-forming AND external bonds   ***/

   count = 0;
   while(max > (count += strcspn(string+count, "123456789"))){

/***exclude numbers inside brackets (which are stereo signals)  ***/

counter = count + strspn(string+count, "0123456789-/\=#H+");
if((counter < max)&&(string[counter] == ']')){
   count++;
   continue;
}

/***determine if numerical bond is ring or external***/

counter = count-1;
while(((string[counter] > '0')&&(string[counter] <= '9'))||\
      (string[counter] == ':')||(string[counter] == '-')||\
      (string[counter] == '/')||(string[counter] == '\\')||\
      (string[counter] == '=')||(string[counter] == '#')){
   counter--;
}

/***ring bonds  ***/

if(string[counter] != '&'){
   bondclass = DIV_BCLS_RING;
}

/***external bonds***/

else if(string[counter] == '&'){
   bondclass = DIV_BCLS_EXTERNAL;
}

/***number bonds should be either ring or external   ***/

else{
   fprintf(stderr, "Error: Number bond which is neither ring nor external.");
   fprintf(stderr, "Error: String:% s\n\n", string);
   exit(1);
}

/***make bond of appropriate class(ring or external) and type***/

   switch(string[count-1]){

case ':':
   bondcount += makebond(bondclass, bondcount, atoms, maxatoms, bonds,   \
   string, count, DX_BTY_AROMAT, DX_CHI_NONE);
   break;

case '-':
   bondcount += makebond(bondclass, bondcount, atoms, maxatoms, bonds,   \
   string, count, DX_BTY_SINGLE, DX_CHI_NONE);
   break;

case '/':
   bondcount += makebond(bondclass, bondcount, atoms, maxatoms, bonds,   \
   string, count, DX_BTY_SINGLE, DIV_CHI_FORWARD);
   break;

case '\\':
   bondcount += makebond(bondclass, bondcount, atoms, maxatoms, bonds,   \
   string, count, DX_BTY_SINGLE, DIV_CHI_BACKWARD);
   break;

case '=':
   bondcount += makebond(bondclass, bondcount, atoms, maxatoms, bonds,   \
   string, count, DX_BTY_DOUBLE, DX_CHI_NONE);
   break;

case '#':
   bondcount += makebond(bondclass, bondcount, atoms, maxatoms, bonds,   \
   string, count, DX_BTY_TRIPLE, DX_CHI_NONE);
   break;

default:
   bondcount += makebond(bondclass, bondcount, atoms, maxatoms, bonds,   \
   string, count, DX_BTY_SINGLE, DX_CHI_NONE);
   break;

   } /* end switch bond type */

count++;

   } /* end while numerical markers */

/***return bondcount, which is the total number of bonds***/

  return(bondcount);

} /* end parsebonds */

/**********************************************************************
 * makebond
 *
 * 4/26/96 ags
 *
 * routine string position and bondtype and chirality and adds a member
 * to the bondlist.
 **********************************************************************/
int makebond(int bondclass, int bondcount, atoms *atoms, int maxatoms, bonds *bonds, char* string,
int position, int bondtype, int chival)
{
   int   count,
   goodbond = TRUE,
   fullbond = TRUE;
   static inthalfflag[MAXTEMPLATE];

/***initialize flags once per molecule***/

   if(bondcount == 0){
for(count=0;count<MAXTEMPLATE;count++){
   halfflag[count] = FALSE;
}
   }

switch(bondclass){

/***make implicit bonds***/

   case DIV_BCLS_IMPLICIT:
bonds[bondcount].iatom1 = position;
bonds[bondcount].iatom2 = position+1;
bonds[bondcount].bondtype = bondtype;
bonds[bondcount].position = atoms[position+1].position - atoms[position+1].inbracket;
bonds[bondcount].branchlevel = atoms[position].branchlevel;
bonds[bondcount].chival = chival;
break;

/***make explicit bonds***/
```

```
   case DIV_BCLS_EXPLICIT:
  bonds[bondcount].iatom2 = postatom(atoms, maxatoms, position);

/***special handling for (X and combinations (=X, (:X, look for lower branchlevel***/

if((string[position] == '(')||(string[position-1] == '(')){
   bonds[bondcount].iatom1 = preatom(atoms, maxatoms, position, atoms[bonds[bond-
count].iatom2].branchlevel-1);
}
else{
   bonds[bondcount].iatom1 = preatom(atoms, maxatoms, position, atoms[bonds[bond-
count].iatom2].branchlevel),
}
bonds[bondcount].bondtype = bondtype;
bonds[bondcount].position = position;
bonds[bondcount].branchlevel = atoms[bonds[bondcount].iatom2].branchlevel;
bonds[bondcount].chival = chival;
break;
   case DIV_BCLS_RING:
fullbond = FALSE; /* only half of ringbond */
if(halfflag[string[position]-48] == FALSE){
   halfflag[string[position]-48] = bondcount;
   bonds[bondcount].iatom1 = preatom(atoms, maxatoms, position, 999);
   bonds[bondcount].bondtype = bondtype;
   bonds[bondcount].position = position.
   bonds[bondcount].branchlevel = atoms[bonds[bondcount].iatom1].branchlevel;
   bonds[bondcount].chival = chival;
}
else{
   goodbond = FALSE;
   bonds[halfflag[string[position]-48]].iatom2 = preatom(atoms, maxatoms, position, 999);
   if(bondtype != DX_BTY_SINGLE){
bonds[halfflag[string[position]-48]].bondtype = bondtype;
   }
   bonds[halfflag[string[position]-48]].position =
JOIN_POSITIONS(bonds[halfflag[string[position]-48]].position, position);
   if(chival != DX_CHI_NONE){
bonds[halfflag[string[position]-48]].chival = chival;
   }

/***register ringbonds in atoms here rather than below    ***/

   count = 0;
   while(atoms[bonds[halfflag[string[position]-48]].iatom1].ibond[count] >= 0){
count++,
   }
   atoms[bonds[halfflag[string[position]-48]].iatom1].ibond[count] = halfflag[string[position]-48];
   count = 0;
   while(atoms[bonds[halfflag[string[position]-48]].iatom2].ibond[count] >= 0){
count++,
   }
   atoms[bonds[halfflag[string[position]-48]].iatom2].ibond[count] = halfflag[string[position]-48];
}
break;
   case DIV_BCLS_EXTERNAL:
bonds[bondcount].iatom2 = (string[position]-48) << 16;
bonds[bondcount].iatom1 = preatom(atoms, maxatoms, position, 999);
bonds[bondcount].bondtype = bondtype,
bonds[bondcount].position = position;
bonds[bondcount].branchlevel = atoms[bonds[bondcount].iatom1].branchlevel;
bonds[bondcount].chival = chival;
break;
   default:
goodbond = FALSE,
break;
}

/***verify any aromatic bonds with exception for halfmade ring bonds***/

   if((bondtype == DX_BTY_AROMAT)&&(fullbond == TRUE)){
if(((atoms[bonds[bondcount].iatom1].aromatic != TRUE)||   \
 (atoms[bonds[bondcount].iatom2].aromatic != TRUE))&&    \
 (halfflag[string[count]] != bondcount)){
   fprintf(stderr, "Error attempting to make aromatic bond between non-aromatic atoms.");
   goodbond = FALSE,
}
   }

/***if it is really a bond, reference the atoms***/
/***ringbonds handled separately in switch above***/

   if(fullbond == TRUE){
count = 0,
while(atoms[bonds[bondcount].iatom1].ibond[count] >= 0){
   count++.
}
atoms[bonds[bondcount].iatom1].ibond[count] = bondcount,

/***if external only reference the first (known) atom   ***/

if(bondclass != DIV_BCLS_EXTERNAL){
   count = 0;
   while(atoms[bonds[bondcount].iatom2].ibond[count] >= 0){
count++.
   }
   atoms[bonds[bondcount].iatom2].ibond[count] = bondcount;
}
   }

/***returns number of bonds made (1 if no errors, 0 if errors)***/

   return(goodbond);

} /* end makebond */

/**********************************************************************
 * preatom
 *
 * 4/25/96 ags
 *
 * routine takes the string position and returns the atomnumber of
 * the previous atom in the bonding tree.
 **********************************************************************/
int preatom(atoms *atomlist, int maxatoms, int current, int branchlevel)
{
   int atomcount = 0;

/***look forward until find first atom after bond or run out***/

   while((atomcount < maxatoms)&&(atomlist[atomcount].position < current)){
atomcount++;
   }

/***go back one atom    ***/

   atomcount--;

/***look backward from there until find atom of same or lower level    ***/

   while((atomcount >= 0)&&(atomlist[atomcount].branchlevel > branchlevel)){
atomcount--;
   }

   if((atomcount == maxatoms)||(atomcount < 0)){
fprintf(stderr, "Error finding previous atom.\n");
exit(1);
   }

   return(atomcount);
}

/**********************************************************************
 * postatom
 *
 * 4/25/96 ags
 *
 * routine takes the string position and returns the atomnumber of
 * the next atom in the bonding tree.
 **********************************************************************/
int postatom(atoms *atomlist, int maxatoms, int current)
{
   int atomcount = 0;

   while((atomlist[atomcount].position < current)&&(atomcount < maxatoms)){
atomcount++;
   }

   if(atomcount == maxatoms){
fprintf(stderr, "Error finding next atom in postatom.\n");
fprintf(stderr, "Error: maxatoms:%d current:%d atomcount:%d\n\n", maxatoms, current, atomcount);
exit(1);
   }

   return(atomcount);
}

/**********************************************************************
 * setchiral
 *
 * 4/30/96 ags
 *
 * this routine takes a smiles-transform string and orders the list of
 * bonds in the chiral sequence so that the chiral class and order are
 * correct for the ordered sequence.
 **********************************************************************/
int setchiral(char *string, atoms *atoms, int maxatoms, bonds *bonds, int maxbonds)
{
   int   chicount = 0,
   xatomct,
   atomct,
   bondct,
   subbondct,
   ibondct1, ibondct2,
   tempatom,
   temppos,
   xatom1,
   atom1, atom2,
   bond1, bond2,
   chi1, chi2,
   count = 0,
   bondsexpected,
   seqct,
   class, set;
   int   position[MAXTEMPLATE];
   char   chiral;

/********************************************/
/***first handle chiral atoms***/
/********************************************/
/***look for chiral atoms***/

   for(atomct=0;atomct<maxatoms;atomct++){
if(atoms[atomct].chival != DX_CHI_NONE){

   chicount++;

   bondsexpected = chibondct(atoms[atomct].chival);
   if(bondsexpected > MAXCHIRAL){
fprintf(stderr, "Error: More bonds than chirality allows.  Adjust MAXCHIRAL.\n");
fprintf(stderr, "Error: String:  %s\nAtom: %d\n\n", string, atomct);
chicount = -999;
   }

/***initialize atom   ***/

   seqct = 0;
```

```
   for(count=0;count<MAXCHIRAL;count++){
atoms[atomct].chiseq[count] = -1;
   }

/***look through bonds of this atom***/

   bondct = 0;
   while((bondct < MAXIBOND)&&(atoms[atomct].ibond[bondct] >= 0)){

/***if allene get secondary bonds***/

if(_CHI_CLASS(atoms[atomct].chival) == DIV_CHI_AL){
   xatom1 = otheratom(atomct, atoms[atomct].ibond[bondct], atoms, bonds);
   subbondct = 0;
   while((subbondct < MAXIBOND)&&\
 (atoms[xatom1].ibond[subbondct] >= 0)){
if((bonds[atoms[xatom1].ibond[subbondct]].iatom1 != atomct)&&\
  (bonds[atoms[xatom1].ibond[subbondct]].iatom2 != atomct)){
atoms[atomct].chiseq[seqct] = atoms[xatom1].ibond[subbondct];
position[seqct] = bondposition(atoms[xatom1].ibond[subbondct], \
    xatom1, atoms, bonds);
seqct++;
} /* end if not same dbl bond */

subbondct++;

   } /* end while secondary bonds left */

} /* end if allene */

/***if NOT allene just use primary bond***/

else{
   atoms[atomct].chiseq[seqct] = atoms[atomct].ibond[bondct];
   position[seqct] = bondposition(atoms[atomct].ibond[bondct], atomct, atoms, bonds);
   seqct++;
}

bondct++;

   } /* end loop through atom's bonds */

/***if necessary add implicit hydrogen  ***/

   if(seqct != bondsexpected){
if(seqct+1 == bondsexpected){
   atoms[atomct].chiseq[seqct] = DIV_IMP_HYDROGEN;
   position[seqct] = atoms[atomct].position;
}
else{
   fprintf(stderr, "Error: Bonding/Chiral class incompatability\n");
   fprintf(stderr, "Error: String: %s\nAtom: %d\n\n", string, atomct);
   chicount = -999;
}
   }

/***sort chiseq by bond position***/

   for(seqct=1;seqct<bondsexpected;seqct++){
tempatom = atoms[atomct].chiseq[seqct];
temppos = position[seqct];
count = seqct-1;
while((count >=0)&&(position[count] > temppos)){
   atoms[atomct].chiseq[count+1] = atoms[atomct].chiseq[count];
   position[count+1] = position[count];
   count--;
}
atoms[atomct].chiseq[count+1] = tempatom;
position[count+1] = temppos;
   }

} /* end if chiral atom*/

/***must make div allene-type into day allene-type***/

if(_CHI_CLASS(atoms[atomct].chival) == DIV_CHI_AL){
   class = DIV2DX_AL(atoms[atomct].chival);
   set = _CHI_COSET(atoms[atomct].chival);
   atoms[atomct].chival = _CHI_VAL(class, set);
}

   } /* end for loop through all atoms */

/*************************************/
/***now handle chiral bonds   ***/
/*************************************/

/***look for double bonds  ***/

   for(bondct=0;bondct<maxbonds;bondct++){
if(bonds[bondct].bondtype == DX_BTY_DOUBLE){
   chiral = FALSE;

/***look for directed single bonds at either end***/

   atom1 = bonds[bondct].iatom1;
   ibondct1 = 0;
      while(atoms[atom1].ibond[ibondct1] >= 0){
bond1 = atoms[atom1].ibond[ibondct1];
if((bonds[bond1].chival == DIV_CHI_FORWARD)||   \
  (bonds[bond1].chival == DIV_CHI_BACKWARD)){
   chi1 = FORBACK_TO_UPDOWN(bonds[bond1].chival,   \
bonds[bond1].position, atoms[atom1].position);

/***now look for second directed single bond at other end***/

   atom2 = bonds[bondct].iatom2;
   ibondct2 = 0;
   while(atoms[atom2].ibond[ibondct2] >= 0){
bond2 = atoms[atom2].ibond[ibondct2];
if((bonds[bond2].chival == DIV_CHI_FORWARD)||   \
  (bonds[bond2].chival == DIV_CHI_BACKWARD)){
   chi2 = FORBACK_TO_UPDOWN(bonds[bond2].chival,   \
bonds[bond2].position, atoms[atom2].position);

/***have 2 directed single bonds around a double bond ..set chirality and chiseq***/

   bonds[bondct].chiseq[0] = bond1;
   bonds[bondct].chiseq[1] = bond2;
   if(chi1 == chi2){
bonds[bondct].chival = DX_CHI_CIS;
   }
   else{
bonds[bondct].chival = DX_CHI_TRANS;
   }
   chiral = TRUE;
   chicount++;

/***chirality set, just get out and look for other double bonds***/

} /* end if bond2 is chiral */

if(chiral)
   break;
ibondct2++;

   } /* end while ibond2 */

} /* end if bond1 is chiral */

if(chiral)
   break;
ibondct1++;

   } /* end while ibond1 */

} /* end if double bond */

   } /* end for loop through all bonds */

/***return number of chiral atoms and bonds or # < 0 if error   ***/

   return(chicount);

} /* end setchiral */

/*********************************************************************
 * chibondct
 *
 * 4/29/96 ags
 *
 * returns the default number of bonds for an atom of given chiral value.
 *********************************************************************/
int chibondct(int value)
{
   intbonds;

   switch(_CHI_CLASS(value)){
case DX_CHI_NONE:
   bonds = 0;
   break;

case DX_CHI_TH:
   bonds = 4;
   break;

case DIV_CHI_AL:
   bonds = 4;
   break;

case DX_CHI_SP:
   bonds = 4;
   break;

case DX_CHI_TB:
   bonds = 5;
   break;

case DX_CHI_OH:
   bonds = 6;
   break;

default:
   fprintf(stderr, "Default reached in chibondct.\n\n");
   bonds = 0;
   break;
   }

   return(bonds);
}
/*********************************************************************
 * otheratom
 *
 * 5/13/96 ags
 *
 * returns the atom number of the atom accross the ibond from iatom.
 *********************************************************************/
int otheratom(int iatom, int ibond, atoms *atoms, bonds *bonds)
{
   int xatom;

   return (iatom == bonds[ibond].iatom2 ?\
bonds[ibond].iatom1 : bonds[ibond].iatom2);
}
/*********************************************************************
```

```
 * bondposition
 *
 * 5/13/96 ags
 *
 * returns bond position of the ibond.  if it is a ringbond, it returns
 * the position associated with iatom.
 ********************************************************************/
int bondposition(int ibond, int iatom, atoms *atoms, bonds *bonds)
{
   int position,

/***check if ringbond   ***/

   if(bonds[ibond].position > 0xffff){

/***get position associated with iatom  ***/

if(bonds[ibond].iatom1 == iatom){
   position = FIRST_POSITION(bonds[ibond].position);
}
else{
   position = SECOND_POSITION(bonds[ibond].position);
}
   }

/*** if not ringbond, just get position    ***/

   else{
position = bonds[ibond].position;
   }

   return(position);

}
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "mol.h"
#include "trans.h"
#include "io.h"
#include "debug.h"
#include "con.h"
#include "parsesmiles.h"

intsyncount;

/*********************************************************
 * readlevels
 *
 * reads reactions for each level of depth
 *
 * ags 4/97
 *********************************************************/
int **readlevels(FILE *Rxnlevelsfile, transform *Transforms, int transcount, int
 *rxnlevels, int maxdepth)
{

   int    count,
   level,
   *Transet,
   **Leveltrans,
   goodread = TRUE,
   found;
   char    buf[MAXBUF];
   char    dummy[MAXBUF],
   char    transname[MAXBUF];

/***initialize arrays and variables***/

   *rxnlevels = -1;

   if(NULL == (Transet = (int *)malloc(maxdepth*sizeof(int)))){
fprintf(stderr, “Out of memory trying to malloc transet in readlevels.\n”);
fprintf(stderr, “size = %d\n”,maxdepth*sizeof(int));
exit(1);
   }
   if(NULL == (Leveltrans = (int **)malloc(maxdepth*sizeof(int*)))){
fprintf(stderr, “Out of memory trying to malloc Leveltrans in readlevels.\n”);
exit(1);
   }
   if(NULL == (Leveltrans[0] = (int *)malloc(maxdepth*transcount*sizeof(int)))){
fprintf(stderr, “Out of memory trying to malloc Leveltrans in readlevels.\n”);
fprintf(stderr, “%d %d %d\n”,maxdepth,transcount,sizeof(int));
fprintf(stderr, “size = %d\n”,maxdepth*transcount*sizeof(int));
exit(1),
   }
   memset(Leveltrans[0],0,maxdepth*transcount*sizeof(int)/sizeof(char));
   for(count=1;count<maxdepth,count++){
Leveltrans[count] = Leveltrans[count-1]+transcount;
   }

/***read input line***/

   while(NULL != fgets(buf, MAXBUF, Rxnlevelsfile)){

sscanf(buf, “%s %d %s[ !-z]”, &dummy, &level, &transname);

if(level <= maxdepth){
found = FALSE;
for(count=0;count<transcount;count++){
   if(strcmp(transname,Transforms[count].name)==0){
Leveltrans[level][count] = TRUE,
found = TRUE;
   }
}
if(!found){
   fprintf(stdout,"Transform %s not found among transforms.\n”),
   goodread = FALSE;
}
if(level > *rxnlevels){
   *rxnlevels = level;
}
} /* end if level < max */
   }

/***if no errors return array else die***/

   if(goodread){
(*rxnlevels)++; /* adjust for starting at 0 */
return(Leveltrans);
   }
   else{
exit(1);
   }

} /* end readlevels */
/*********************************************************
 * get_input
 *
 * reads input file.
 *
 * ags 10/96
 *********************************************************/
int get_input(FILE *Infile, filenames *Filenames, molparms *Molecule_parameters, transparms
*Transform_parameters)
{
  int    goodread = TRUE,
     count;
  int    parmcheck=0;
  char    buf[MAXBUF];
  char    keyword[MAXBUF];
  char    value[MAXBUF],

/***   set defaults if there are any  ***/

/***   read input file    ***/

  while(NULL != fgets(buf, MAXBUF, Infile)){
     sscanf(buf, “%s %s[ !-z]”, &keyword, &value);

/***   parse strings      ***/

     if(strcmp(keyword, “smiles_file”)==0){
        strncpy(Filenames->smiles, value, strlen(value));
        continue,
     }
     if(strcmp(keyword, “transform_file”)==0){
        strncpy(Filenames->transform, value, strlen(value));
        continue;
     }
     if(strcmp(keyword, “library_file”)==0){
        strncpy(Filenames->library, value, strlen(value));
        continue;
     }
     if(strcmp(keyword, “output_file”)==0){
        strncpy(Filenames->output, value, strlen(value));
        continue;
     }
     if(strcmp(keyword, “vbindings_file”)==0){
        strncpy(Filenames->vbind, value, strlen(value));
        continue;
     }
     if(strcmp(keyword, “rxn_levels_file”)==0){
        strncpy(Filenames->rxn_levels, value, strlen(value));
        continue;
     }

/***   parse numbers      ***/

     if(strcmp(keyword, “maximum_depth”)==0){
  sscanf(value, “%d”, &(Molecule_parameters->maxdepth));
  parmcheck |= 1<<0;
  continue;
     }
     if(strcmp(keyword, “maximum_mol_wt”)==0){
        sscanf(value, “%d”, &(Molecule_parameters->maxmolwt));
  parmcheck |= 1<<1;
        continue;
     }
     if(strcmp(keyword, “minimum_mol_wt”)==0){
        sscanf(value, “%d”, &(Molecule_parameters->minmolwt));
  parmcheck |= 1<<2;
        continue;
     }
     if(strcmp(keyword, “maximum_molecules”)==0){
        sscanf(value, “%d”, &(Molecule_parameters->maxmolecule));
  parmcheck |= 1<<3;
        continue;
     }
  }

/***way overcomplicated checking for all the parameters***/

  if(parmcheck < 15){
goodread = FALSE;
fprintf(stderr, “Missing essential parameter(s) from input file.\n”);
for(count=0;count<4;count++){
  if(!parmcheck&1<<count){
switch(count){
  case 0:
fprintf(stderr, “Need maximum_depth.\n”);
break;
  case 1:
fprintf(stderr, “Need maximum_mol_wt.\n”),
break;
  case 2:
```

```
fprintf(stderr, "Need minimum_mol_wt.\n");
break;
   case 3
fprintf(stderr, "Need maximum_molecules.\n");
break;
}
   }
}
   }

   fclose(Infile);
   return(goodread);

} /* end getin */

/************************************************************************
 * re_read molequeues
 *
 * an i/o routine to read in the next molequeue for transtromation
 * from the middle of the temp file  It will put this next molequeue
 * into the molequeue that is sent to it and if successful will return
 * a pointer to that same molequeue  if there is a problem it will
 * return null
 ************************************************************************/
molequeue *re_read(FILE *Outread, molequeue *Active)
{
   char  buf[MAXBUF];

/***read data for new active into old active place***/

   /* one line for everything in temp molecule */

   if(NULL == (fgets(buf, MAXBUF, Outread))){
 fprintf(stderr, "NULL in re_read\n");
 return(NULL);
   }
   buf[strlen(buf)-1]='\0';
   sscanf(buf, "%d %s %s %s %d %[^\n]",  \
   &(Active->synno),  \
   Active->smiles,  \
   Active->transform,  \
   Active->parent,  \
   &(Active->depth),  \
   Active->externalid);

   if(NULL == (Active->molecule = (dt_smilin(strlen(Active->smiles), Active->smiles)))){
fprintf(stderr, "Error reading in smiles from temp. %s\n", Active->smiles);
fprintf(stderr, "Parent  %s\n", Active->parent);
fprintf(stderr, "depth = %d, transform = %s\n\n", Active->depth, Active->transform);
   }

/***reset fields not read in   ***/


   Active->identical = -1;
   Active->tanimoto = 0.0;
   Active->nochildren = 0;
   Active->Next = NULL;
   Active->Previous = NULL;

   return(Active);
}
/************************************************************************
 * temp write
 *
 * an i/o routine to write a product and some molequeue information to
 * a temporary output file  a pointer to the next molequeue in the
 * queue is returned.
 ************************************************************************/
molequeue *tempwrite(FILE *Outfile, molequeue *Done)
{

/***write abbreviated form of molequeue to temporary outfile   ***/

   fprintf(Outfile, "%d %s %s %s %d %s\n",  \
Done->synno,  \
Done->smiles,  \
Done->transform,  \
Done->parent,  \
Done->depth,  \
Done->externalid);
   fflush(Outfile);

/***rejoin the queue***/

   if(Done->Next != NULL)
Done->Next->Previous = Done->Previous;
   if(Done->Previous != NULL)
Done->Previous->Next = Done->Next;

/***release the molecule's memory***/

   dt_dealloc(Done->molecule);

/***return next molequeue***/

   return(Done->Next);

}
/************************************************************************
 * readlibfile
 *
 * an i/o routine to read in the library format
 ************************************************************************/
library *readlibs(FILE *Libfile, library *Libraries, int *libtot, transform *Transforms, int transtot)
{
   charbuf[MAXBUF],
keyword[MAXBUF],
```

```
libinfo[MAXBUF],
transname[MAXNAME],
smiles[MAXBUF],
externalid[MAXBUF],
smarts[5];
   intcount, count1,
fragct[MAXLIBRARY],
libct = -1,
transct,
connumber,
delct;
   dt_Handlekey,
pathset,
paths,
path,
atoms,
atom,
bonds,
bond,
xatom,
delatoms[MAXATTACHATOMS];
   molequeue*Frag;

/***initialize fragment***/

   if(NULL == (Frag = (molequeue *)malloc(sizeof(molequeue)))){
fprintf(stderr, "Out of memory mallocing temporary fragment in readlibs.\n");
exit(1);
   }

/***initialize fragct***/

   for(count=0;count<MAXLIBRARY;count++){
fragct[count]=0;
   }

/***get size and number of libraries   ***/

   while(NULL != fgets(buf, MAXBUF, Libfile)){
      sscanf(buf, "%s", &keyword);
if(0 == strcmp(keyword, "Library:")){
   libct++;
}
else if(0 != strcmp(buf, "\n")){
   fragct[libct]++;
}
   }
   libct++;

/***set total number of libraries***/

   *libtot = libct;

/***malloc libraries, fragments, and attachment atoms   ***/

   if(NULL == (Libraries = (library *)malloc(libct*sizeof(library)))){
fprintf(stderr, "Out of memory trying to malloc library.\n");
return(NULL);
   }

   for(count=0;count<libct;count++){
if(NULL == (Libraries[count].Fragments =\
(dt_Handle *)malloc(fragct[count]*sizeof(dt_Handle)))){
   fprintf(stderr, "Out of memory trying to malloc fragment.\n");
   return(NULL);
}
if(NULL == (Libraries[count].Externalids =\
(char **)malloc(fragct[count]*sizeof(char *)))){
   fprintf(stderr, "Out of memory trying to malloc fragment id's.\n");
   return(NULL);
}
if(NULL == (Libraries[count].Externalids[0] =\
(char *)malloc(MAXNAME*fragct[count]*sizeof(char)))){
   fprintf(stderr, "Out of memory trying to malloc fragment id's.\n");
   return(NULL);
}
if(NULL == (Libraries[count].Attachatoms =\
(dt_Handle **)malloc(fragct[count]*sizeof(dt_Handle *)))){
   fprintf(stderr, "Out of memory trying to malloc attachment.\n");
   return(NULL);
}
if(NULL == (Libraries[count].Attachatoms[0] =\
(dt_Handle *)malloc(MAXATTACHATOMS*fragct[count]*sizeof(dt_Handle)))){
   fprintf(stderr, "Out of memory trying to malloc attachment atoms in readlib.\n");
   return(NULL);
}
for(count1=1;count1<fragct[count];count1++){
   Libraries[count].Externalids[count1] = Libraries[count].Externalids[count1 - 1] + MAXNAME;
   Libraries[count].Attachatoms[count1] = Libraries[count].Attachatoms[count1 - 1] + MAXAT-
TACHATOMS;
}
   }

/***reset before getting data   ***/

   rewind(Libfile);
   libct = -1;
   strcpy(smarts, "[#0]");
   if(NULL_OB == (key = dt_smartin(strlen(smarts), smarts))){
fprintf(stderr, "BIG ERROR: Unable to make %s into smarts in readlibs.\n", smarts);
exit(1);
   }

   while(NULL != fgets(buf, MAXBUF, Libfile)){

/***get header of library***/

      sscanf(buf, "%s %[ \!-z]", &keyword, &libinfo);
```

```
if(0 == strcmp(keyword, “Library:”)){
   if(libct >= 0){
Libraries[libct].size = count+1;
fprintf(stderr, “\nRead %d fragments for Library %s.\n\n”, Libraries[libct].size, Libraries[libct].name);
   }
   libct++;
   sscanf(libinfo, “%s %s”, &Libraries[libct].name, &transname);
fprintf(stderr, “Reading library %s and ”,Libraries[libct].name);
fprintf(stderr, “converting it using transform %s.\n”, transname);


/***find correct transform for library    ***/

   transct = -1;
   for(count=0;count<transtot;count++){
if(0 == strcmp(Transforms[count].name, transname)){
   transct = count;
   break;
}
   }
   if(transct < 0){
fprintf(stderr, “Unable to find transform %s to transform library %s.\n”, \
   transname, Libraries[libct].name);
exit(1);
   }
   count = -1;
}
else if(0 != strcmp(buf, “\n”)){

/***read new fragment smiles and make molequeue***/

   buf[strlen(buf)-1] = ‘\0’;
   memset(externalid, 0, strlen(buf));
   sscanf(buf, “%s %s”, &smiles, &externalid);

   if(NULL_OB == (Frag->molecule = dt_smilin(strlen(smiles), smiles))){
fprintf(stderr, “\nUnable to convert %s to library molecule in %s.\n”, \
   smiles, Libraries[libct].name);
continue;
   }
   else{
count++;
if(count%10 == 0)
 fprintf(stderr, “.”);
   }

   memset(Libraries[libct].Externalids[count], ‘\0’, MAXNAME);
   if(strlen(externalid) > MAXNAME){
fprintf(stderr, “Fragment name %s is too long.  Max. length = %d.\n”, externalid, MAXNAME);
   }
   else if(0 == strcmp(externalid, “”)){
sprintf(Libraries[libct].Externalids[count], “,%d %d”, libct, count);
   }
   else{
sprintf(Libraries[libct].Externalids[count], “,%s”, externalid);
   }
   strcpy(Frag->smiles, smiles);

/***transform fragment molequele***/

   connect(Frag, Transforms+transct, NULL);

   Libraries[libct].Fragments[count] = dt_copy(Frag->molecule);
   dt_dealloc(Frag->molecule);

/***find attachment atoms in fragment and dealloc dummy pointer atoms   ***/

   if(FALSE == dt_mod_on( Libraries[libct].Fragments[count])){
fprintf(stderr, “\nUnable to turn modify on for %d in library %s.\n”, count, Libraries[libct].name);
count--;
continue;
   }

   delct = 0;
   if(NULL_OB != (pathset = dt_vmatch(key, Libraries[libct].Fragments[count], FALSE))){
if(NULL_OB != (paths = dt_stream(pathset, TYP_PATH))){
   while(NULL_OB != (path = dt_next(paths))){
if(NULL_OB != (atoms = dt_stream(path, TYP_ATOM))){
   while(NULL_OB != (atom = dt_next(atoms))){
connumber = dt_hcount(atom);
bonds = dt_stream(atom, TYP_BOND);
bond = dt_next(bonds);
xatom = dt_xatom(atom, bond);
delatoms[delct] = atom;
delct++;
Libraries[libct].Attachatoms[count][connumber] = xatom;
dt_dealloc(bonds);
   }
}
dt_dealloc(atoms);
   }
   dt_dealloc(path);
}
dt_dealloc(paths);
for(count1=0;count1<delct;count1++){
   dt_dealloc(delatoms[count1]);
}
dt_dealloc(pathset);
   }
   else{
fprintf(stderr, “\nUnable to find key in molecule: %s for library: %s.\n\n”, \
   buf, Libraries[libct].name);
count--;
continue;
   }

   if(FALSE == dt_mod_off( Libraries[libct].Fragments[count])){
fprintf(stderr, “\nUnable to turn modify off for %d in library %s.\n”, count, Libraries[libct].name);
count--;
continue;
   }

} /* end if molecule */

   } /* end if nextline in library file */

/***need to finish counting libraries and fragments    ***/

   Libraries[libct].size = count+1;
fprintf(stderr, “\nRead %d fragments for Library %s.\n”, Libraries[libct].size, Libraries[libct].name);
   libct++;

/***avoid read errors   ***/

   if(libct > *libtot){
fprintf(stderr, “Read more libraries than malloced for.  libct = %d\n.”, libct);
exit(1);
   }


/***die if all libraries not read***/

   if(libct != *libtot){
fprintf(stderr, “Stopping becuase of library error.\n”);
exit(1);
   }
   return(Libraries);

 }
/*****************************************************************************
 * readinmols
 *
 * an i/o routine to read in all the molecules from a SMILES input
 * file, create a linked list from them, and return a pointer to the
 * first molequeue.
 ****************************************************************************/
molequeue *readinmols(FILE *Smilesfile, molequeue **Lastptr)
{
   char    *buf,
   smiles[MAXBUF],
   tag[10],
   *Startptr,
   *Endptr;
   molequeue*Latest,
*First = NULL;
   intcount,
length,
goodmol = FALSE;

char junk[5000];

   if(NULL == (buf = (char *)malloc(MAXBUF*sizeof(char)))){
fprintf(stderr, “Out of memory way too early in readinmols\n”);
exit(1);
   }

   while(NULL != fgets(buf, MAXBUF, Smilesfile)){

strcpy(junk, buf);

/***use SMILES string to allocate molecular object***/

buf[strlen(buf)-1] = ‘\0’;

memset(tag, 0, 10);
sscanf(buf, “%4s”, &tag);
if(buf[0] == ‘|’){
   continue;
}
if(NULL == (Startptr = strchr(buf, ‘<’))){
   fprintf(stderr, “Couldn’t find a ‘<’ in a line of molfile.\n”);
   fprintf(stderr, “Buffer:  %s\n”, buf);
   continue;
}
strcpy(Startptr, Startptr+1);
if(NULL == (Endptr = strchr(buf, ‘>’))){
   fprintf(stderr, “Couldn’t find a ‘>’ in a line of molfile.\n”);
   fprintf(stderr, “Buffer:  %s\n”, buf);
   continue;
}
*Endptr = ‘\0’;
if(0 == strcmp(tag, “$SMI”)){

/***allocate next molecule in linked list***/

   if(NULL == (Latest = (molequeue *)malloc(sizeof(molequeue)))){
fprintf(stderr, “Out of memory in: readinmols\n”);
fprintf(stderr, “Molecule Number: %d\n”, syncount+1);
return(First);
   }

   if(NULL == (Latest->molecule = (dt_smilin(strlen(Startptr), Startptr)))){
fprintf(stderr, “Error reading in smiles: %s\n”, Startptr);
goodmol = FALSE;
free(Latest);
continue;
   }
   goodmol = TRUE;

/***initialize molequeue***/

   strncpy(Latest->smiles, dt_cansmiles(&length, Latest->molecule, TRUE), length);
   Latest->smiles[length] = ‘\0’;
   sprintf(Latest->externalid, “DIV%08d”, syncount);
   strcpy(Latest->parent, “-1:None\0”);
   Latest->identical = -1;
```

```
   strcpy(Latest->transform, "None\0");
   for(count=0;count<MAXCHILDREN;count++){
Latest->children[count] = 0;
   }
   Latest->Next = NULL;
   Latest->Previous = NULL;
   Latest->depth = 0;
   Latest->synno = syncount;
   syncount++;
   Latest->nochildren = 0;

/***note the first moleqeue as the First***/

   if(First == NULL){
First = Latest;
*Lastptr = Latest;
   }
   else{
(*Lastptr)->Next = Latest;
Latest->Previous = *Lastptr;
*Lastptr = Latest;
   }

/***parse non-smiles lines from the tdt***/

}
else if((0 == strcmp(tag, "$TAG")&&(goodmol))){
   memset(Latest->externalid, 0, MAXSMILES);
   strcpy(Latest->externalid, Startptr);
   fprintf(stderr, "%s\n", Latest->externalid);
   continue;
}
else{
   continue;
}

   memset(buf, '\0', MAXBUF),
   }/* end while */

/***after all the molecules, return a pointer to the first one.***/

/*  free(buf);  */
   return(First);
}

/*******************************************************************
 * retire
 *
 * an i/o routine to remove a molecule from the list and write it to
 * an output file.  a pointer to the next molequeue in line is returned
 *******************************************************************/
molequeue *retire(FILE *Outfile, molequeue *Done)


{
   intcount,
   molequeue*Next,

/***print out the vitals for the retireing molecule    ***/

   fprintf(Outfile, "%d: %s <---%s--- %s\n", Done->synno, Done->smiles,   \
   Done->transform, Done->parent),
   fprintf(Outfile, "DIV%08d depth = %d\n", Done->synno, Done->depth);
   fprintf(Outfile, "$SMI<%s>$TAG<%s>\n", Done->smiles, Done->externalid);
   fprintf(Outfile, "Tanimoto = %.2f\n", Done->tanimoto);
   if(Done->identical != -1){
fprintf(Outfile, "Identical>> %d", Done->identical);
   }
   else{
fprintf(Outfile, "Children>> %d:  ", Done->nochildren);
for(count=0;count < Done->nochildren,count++){
   fprintf(Outfile, "%d ", Done->children[count]);
}
   }
   fprintf(Outfile, "\n\n");
   fflush(Outfile);


/***close the list***/

   Next = Done->Next;
   if(Done->Next != NULL)
Done->Next->Previous = Done->Previous;
   if(Done->Previous != NULL)
Done->Previous->Next = Done->Next;

/***deallocate memory   ***/

   dt_dealloc(Done->molecule);
/*    free(Done);commented out b/c have trouble with multiple molequeue's
 *malloced by the same call...
 */

/***return pointer to the next molequeue***/

   return(Next);
}

/*******************************************************************
 * readtrans
 *
 * an i/o routine to read in reactions from a reaction file.  The
 * reactions are placed in an array of rxn structs, and the total
 * number of reactions is returned.
 *******************************************************************/
int readtrans(FILE *Transformfile, transform *transforms)
{
   charbuf[MAXBUF],
keyword[MAXKEY];
   inti,
transcount = 0,
smartlen;

/***read transforms***/

   while((transcount < MAXTRANS)&&   \
  (NULL != (fgets(buf, MAXBUF, Transformfile)))){

/***blank lines separate transforms***/
/**********************************************
NEED TO INITIALIZE REACTION STRUCTURES
**********************************************/

while(0 != strcmp(buf, "\n")){

/***read keyword and find text***/

      sscanf(buf, "%s %[ !-z]", &keyword, &buf);

/***find datatype and enter data***/

   if(0 == strcmp(keyword, "name:")){
strncpy(transforms[transcount].name, buf, strlen(buf));
   }
   else if(0 == strcmp(keyword, "type:")){
      transforms[transcount].flag = buf[0];
   }
   else if(0 == strcmp(keyword, "inclusion:")){
      strncpy(transforms[transcount].inclusion, buf, strlen(buf));
   }
   else if(0 == strcmp(keyword, "exclusion:")){
      strncpy(transforms[transcount].exclusion, buf, strlen(buf));
   }
   else if(0 == strcmp(keyword, "mark:")){
      strncpy(transforms[transcount].marked, buf, strlen(buf));
   }
   else if(0 == strcmp(keyword, "library:")){
strncpy(transforms[transcount].type.con.libraryname, buf, strlen(buf));
   }
   else if(0 == strcmp(keyword, "productset:")){
switch(transforms[transcount].flag){
   case 'r':
strncpy(transforms[transcount].type.rep.productset,\
buf, strlen(buf));
break;
   case 'l': /* same as type con */
   case 'c':
strncpy(transforms[transcount].type.con.productset,\
buf, strlen(buf)),
break;

}
   }
   else if(0 == strcmp(keyword, "productset1:")){
      strncpy(transforms[transcount].type.sep.productset[0],\
buf, strlen(buf));
   }
   else if(0 == strcmp(keyword, "productset2:")){
      strncpy(transforms[transcount].type.sep.productset[1],\
buf, strlen(buf));
   }

/***break out if reach end of file (or read error)***/

   if(NULL == fgets(buf, MAXBUF, Transformfile)){
break;
   }

} /* end while not blank line. */

/***debugging feedback***/

fprintf(stderr, "==================\n%d\n", transcount);
fprintf(stderr, "%s\n", transforms[transcount].name);
fprintf(stderr, "%s\n", transforms[transcount].inclusion);
fprintf(stderr, "%s\n", transforms[transcount].exclusion);

/***routine which converts external-->internal data-types ***/

switch(transforms[transcount].flag){
   case 'r':
if(NULL != (parserep(&(transforms[transcount])))){
      transcount++;

/***debugging   ***/
fprintf(stderr, "%s\n", transforms[transcount-1].type.rep.productset);
fprintf(stderr, "%d\n", transforms[transcount-1].type.rep.atomnumber);
/*
for(i=0;i<transforms[transcount-1].type.rep.atomnumber;i++){
   fprintf(stderr, "%c %d\n", transforms[transcount-1].type.rep.productatom[i].symbol, trans-
forms[transcount-1].type.rep.productatom[i].atomicno);
}*/
}
break;
   case 's':
if(NULL != (parsesep(&(transforms[transcount])))){
   transcount++;

/***debugging   ***/
fprintf(stderr, "%s\n", transforms[transcount-1].type.sep.productset[0]);
fprintf(stderr, "%d\n", transforms[transcount-1].type.sep.atomnumber[0]);
/*
for(i=0;i<transforms[transcount-1].type.sep.atomnumber[0];i++){
   fprintf(stderr, "%c %d\n", transforms[transcount-1].type.sep.productatom[0][i].symbol, trans-
forms[transcount-1].type.sep.productatom[0][i].atomicno);
}*/
```

```
fprintf(stderr, "%s\n", transforms[transcount-1].type.sep.productset[1]);
fprintf(stderr, "%d\n", transforms[transcount-1].type.sep.atomnumber[1]);
/*
for(i=0;i<transforms[transcount-1].type.sep.atomnumber[1];i++){
   fprintf(stderr, "%c %d\n", transforms[transcount-1].type.sep.productatom[1][i].symbol, trans-
forms[transcount-1].type.sep.productatom[1][i].atomicno);
}*/
}
break;
   case 'I': /* same as type con */
   case 'c':
if(NULL != (parsecon(&(transforms[transcount])))){
   transcount++;

/***debugging  ***/
fprintf(stderr, "%s\n", transforms[transcount-1].type.con.productset);
fprintf(stderr, "%d\n", transforms[transcount-1].type.con.atomnumber);
/*
for(i=0;i<transforms[transcount-1].type.con.atomnumber;i++){
   fprintf(stderr, "%c %d\n", transforms[transcount-1].type.con.productatom[i].symbol, trans-
forms[transcount-1].type.con.productatom[i].atomicno);
}*/
}
break;
   default:
fprintf(stderr, "Can't find transform type for %s.\n",   \
transforms[transcount].name);
fprintf(stderr, "It will not be used.\n\n");
}

/***more debugging***/
/*
*/
   }/* end while count<MAX && not EOF */

   return(transcount);
}
/********************************************************************
 * parse replacement
 *
 * an i/o routine which converts the smiles-like external transform
 * data-types to internal pattern, atom, and bond data for replacement
 * transform types.
 ********************************************************************/
transform *parserep(transform *Transform)
{
   int   maxatoms,
   maxbonds,
   count,
   subcount,
   char   string[MAXTEMPLATE],
   atoms  newatoms[MAXTEMPLATE],
   bonds  fragbonds[MAXTEMPLATE],

/***initialize variables***/

   for(count=0;count<MAXTEMPLATE;count++){
string[count]='\0';
fragbonds[count].iatom1 = -1;
fragbonds[count].iatom2 = -1;
   }

/***convert inclusion and exclusion SMARTS to pattern objects ***/

   if(FALSE == getsmart(Transform)){
return(NULL);
   }

/***parse transform product SMILES(limited set) into atoms and bonds   ***/

   strncpy(string, Transform->type.rep.productset, strlen(Transform->type.rep.productset));

   if(FALSE == parsesmiles(string, newatoms, &maxatoms, fragbonds, &maxbonds)){
return(NULL);
   }

/**********************************************************/
/***done parsing, now add internal atoms/bonds***/
/**********************************************************/

/***count product atoms***/

   Transform->type.rep.atomnumber = maxatoms;

/***add atoms  ***/

   for(count = 0;count < maxatoms;count++){
Transform->type.rep.productatom[count].symbol = newatoms[count].symbol;
Transform->type.rep.productatom[count].atomicno = newatoms[count].atomicno;
Transform->type.rep.productatom[count].aromatic = newatoms[count].aromatic;
Transform->type.rep.productatom[count].formalcharge = newatoms[count].formalcharge;
Transform->type.rep.productatom[count].chival = newatoms[count].chival;
for(subcount=0;subcount<MAXCHIRAL;subcount++){
   Transform->type.rep.productatom[count].chiseq[subcount] = newatoms[count].chiseq[subcount];
}
   }

/***count product bonds***/

   Transform->type.rep.bondnumber = maxbonds;

/***add fragment bonds  ***/

   for(count = 0;count<maxbonds;count++){
Transform->type.rep.productbond[count].iatom1 = fragbonds[count].iatom1;
 Transform->type.rep.productbond[count].iatom2 = fragbonds[count].iatom2;
 Transform->type.rep.productbond[count].bondtype = fragbonds[count].bondtype;
Transform->type.rep.productbond[count].chival = fragbonds[count].chival;
Transform->type.rep.productbond[count].chiseq[0] = fragbonds[count].chiseq[0];
Transform->type.rep.productbond[count].chiseq[1] = fragbonds[count].chiseq[1];

   }

/***return pointer to Transform if parsing is sucessful***/
/*
   fprintf(stderr, "%d\n", Transform->type.rep.atomnumber);
   for(count=0;count<Transform->type.rep.atomnumber;count++){
fprintf(stderr, "%c %d\n", Transform->type.rep.productatom[count].symbol, Transform->type.rep.pro-
ductatom[count].atomicno);
   }
*/
   return(Transform);

}
/********************************************************************
 * parse separation
 *
 * an i/o routine which converts the smiles-like external transform
 * data-types to internal pattern, atom, and bond data for separation
 * transform types
 ********************************************************************/
transform *parsesep(transform *Transform)
{
   int   half,
   count = 0,
   subcount,
   maxatoms,
   maxbonds,
   char   string[MAXTEMPLATE];
   atoms  newatoms[MAXTEMPLATE];
   bonds  fragbonds[MAXTEMPLATE];

/***convert inclusion and exclusion SMARTS to pattern objects ***/

   if(FALSE == getsmart(Transform)){
return(NULL);
   }

/***loop through both half's of the transform  ***/

   for(half=0;half < 2;half++){

/***initialize variables***/

   for(count=0;count<MAXTEMPLATE;count++){
string[count]='\0';
fragbonds[count].iatom1 = -1;
fragbonds[count].iatom2 = -1;
   }

/***parse transform product pseudo-SMILES into atoms and bonds   ***/

strncpy(string, Transform->type.sep.productset[half], strlen(Transform->type.sep.productset[half]));

if(FALSE == parsesmiles(string, newatoms, &maxatoms, fragbonds, &maxbonds)){
   return(NULL);
}

/**********************************************************/
/***done parsing, now add internal atoms/bonds***/
/**********************************************************/

/***count product atoms***/

Transform->type.sep.atomnumber[half] = maxatoms;

/***add atoms  ***/

for(count = 0;count<maxatoms;count++){
   Transform->type.sep.productatom[half][count].symbol = newatoms[count].symbol;
   Transform->type.sep.productatom[half][count].atomicno = newatoms[count].atomicno;
   Transform->type.sep.productatom[half][count].aromatic = newatoms[count].aromatic;
   Transform->type.sep.productatom[half][count].formalcharge = newatoms[count].formalcharge;
   Transform->type.sep.productatom[half][count].chival = newatoms[count].chival;
   for(subcount=0;subcount<MAXCHIRAL;subcount++){
Transform->type.sep.productatom[half][count].chiseq[subcount] = newatoms[count].chiseq[subcount];
   }
}

/***count product bonds***/

Transform->type.sep.bondnumber[half] = maxbonds;

/***add fragment bonds  ***/

for(count = 0;count<maxbonds;count++){
   Transform->type.sep.productbond[half][count].iatom1 = fragbonds[count].iatom1;
    Transform->type.sep.productbond[half][count].iatom2 = fragbonds[count].iatom2;
    Transform->type.sep.productbond[half][count].bondtype = fragbonds[count].bondtype;
   Transform->type.sep.productbond[half][count].chival = fragbonds[count].chival;
   Transform->type.sep.productbond[half][count].chiseq[0] = fragbonds[count].chiseq[0];
   Transform->type.sep.productbond[half][count].chiseq[1] = fragbonds[count].chiseq[1];
}

   } /* end loop through transform halves */

   return(Transform);

}
/********************************************************************
 * parse connection
 *
 * an i/o routine which converts the smiles-like external transform
 * data-types to internal pattern, atom, and bond data for connection
 * transform types.
```

```
 *********************************************************************/
transform *parsecon(transform *Transform)
{
   int    count = 0,
   maxatoms,
   maxbonds,
   subcount,
   char    string[MAXTEMPLATE],
   atoms   newatoms[MAXTEMPLATE];
   bonds   fragbonds[MAXTEMPLATE*3],

/***initialize variables***/

   for(count=0;count<MAXTEMPLATE;count++){
string[count]='\0';
fragbonds[count].iatom1 = -1,
fragbonds[count].iatom2 = -1;
   }

/***convert inclusion and exclusion SMARTS to pattern objects ***/

   if(FALSE == getsmart(Transform)){
return(NULL),
   }

/***parse transform product pseudo-SMILES into atoms and bonds    ***/

   strncpy(string, Transform->type.con.productset, strlen(Transform->type.con.productset));

   if(FALSE == parsesmiles(string, newatoms, &maxatoms, fragbonds, &maxbonds)){
return(NULL),
   }

/**********************************************************/
/***done parsing, now add internal atoms/bonds***/
/**********************************************************/

/***count product atoms***/

   Transform->type.con.atomnumber = maxatoms;

/***add atoms   ***/

   for(count = 0;count<maxatoms;count++){
Transform->type.con.productatom[count].symbol = newatoms[count].symbol,
Transform->type.con.productatom[count].atomicno = newatoms[count].atomicno;
Transform->type.con.productatom[count].aromatic = newatoms[count].aromatic;
Transform->type.con.productatom[count].formalcharge = newatoms[count].formalcharge;
Transform->type.con.productatom[count].chival = newatoms[count].chival;
for(subcount=0;subcount<MAXCHIRAL;subcount++){
   Transform->type.con.productatom[count].chiseq[subcount] = newatoms[count].chiseq[subcount];
}
   }

/***count product bonds***/

   Transform->type.con.bondnumber = maxbonds,

/***add fragment bonds  ***/

   for(count = 0;count<maxbonds;count++){
Transform->type.con.productbond[count].iatom1 = fragbonds[count].iatom1;
 Transform->type.con.productbond[count].iatom2 = fragbonds[count].iatom2,
 Transform->type.con.productbond[count].bondtype = fragbonds[count].bondtype;
Transform->type.con.productbond[count].chival = fragbonds[count].chival;
Transform->type.con.productbond[count].chiseq[0] = fragbonds[count].chiseq[0];
Transform->type.con.productbond[count].chiseq[1] = fragbonds[count].chiseq[1],
   }

/***return pointer to Transform if parsing is sucessful***/

   return(Transform);

}

/**********************************************************************
 * getsmart
 *
 * fills in smarts strings for transforms, returning boolean for success.
 * note, this returns TRUE even if the exclusion fails as long as the
 * inclusion succeeds
 *********************************************************************/
int getsmart(transform *Transform)
{
/***convert inclusion to daylight smarts***/

   if(NULL_OB == (Transform->screenin = dt_smartin(strlen(Transform->inclusion),\
   Transform->inclusion))){
fprintf(stderr, "Unable to convert %s to SMARTS.\n",    \
   Transform->inclusion);
fprintf(stderr, "Reaction %s will not work.\n", Transform->name),
memset(Transform->inclusion, 0, strlen(Transform->inclusion)),
return(FALSE);
   }

/***if exclusion exists convert to daylight smarts pattern***/

   if(strlen(Transform->exclusion)>0){
if(NULL_OB == (Transform->screenout = dt_smartin(    \
   strlen(Transform->exclusion),    \
   Transform->exclusion))){
   fprintf(stderr, "Unable to convert %s to SMARTS.\n",    \
Transform->exclusion),
   fprintf(stderr, "Reaction %s will be applied generally.\n", Transform->name);
}
   }

/***if marked exists convert to daylight smarts pattern***/

   if(strlen(Transform->marked)>0){
if(NULL_OB == (Transform->markscreen = dt_smartin(    \
   strlen(Transform->marked),    \
   Transform->marked))){
   fprintf(stderr, "Unable to convert %s to SMARTS.\n",    \
Transform->marked);
   fprintf(stderr, "Reaction %s will will not mark atoms.\n", Transform->name);
}
   }

/***if gets to this point, transform passes    ***/

   return(TRUE);

} /* end getsmart */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "mol.h"
#include "trans.h"
#include "debug.h"
#include "io.h"
#include "screen.h"

externintsyncount;

/**********************************************************************
 * prune
 *
 * subroutine takes a molequeue as an argument and returns whether or
 * not that molequele should be transformed. current screens include
 * total molecules, depth of search, molecular weight, and tanimoto
 * similarity.
 *********************************************************************/
int prune(molequeue *Testmol, molparms *Parameters)
{

   int    mw,
   goodmol = TRUE,

/***screen for total number of molequeue's***/

if(syncount >= MAXMOLEQUEUE){
   goodmol = FALSE;
}

/***screen for depth of search  ***/

if(Testmol->depth >= Parameters->maxdepth){
   goodmol = FALSE;
}

/***screen molecular weight before reacting    ***/

/*
mw = get_molwt(Testmol->molecule),
if((mw >= MAXMOLWT)||(mw < MINMOLWT)){
   goodmol = FALSE;
}*/ /* end if not NULL */


/***screen for 2d similarity to starting mol    ***/

if(FALSE){
   if(FALSE == tanimoto(Testmol, 0.0, 1.0)){
fprintf(stderr, "FALSE=%d\n", Testmol->tanimoto);
goodmol = FALSE;
   }
}

/***return TRUE for pass return FALSE for pruned***/

return(goodmol);

}
/**********************************************************************
 * tanimoto
 *
 * subroutine takes a molequeue and (min, max) as arguments. the first
 * time it is called, it will remember the fp of the molequeue and
 * compare all future fp's (Daylight 2D) against it. The tanimoto
 * between the two will be stored in the molequeue and if it is within
 * the limits set by (min, max) will return TRUE otherwise it returns
 * FALSE.
 *********************************************************************/
int tanimoto(molequeue *Testmol, float min, float max)
{

   static intflag = TRUE;
   static charheadname[MAXSMILES];
   static dt_Handlefp;
   static dt_Handlefp1;

/***get fp of first molequeue   ***/

   if(flag){
strcpy(headname, Testmol->smiles);
if(NULL_OB == (fp = dt_fp_fingerprint(Testmol->molecule, FALSE))){
   fprintf(stderr, "Error getting fp of first molecule, %s\n\n",   \
Testmol->smiles);
}
flag = FALSE;
   }

/***compare the fp of each molecule with that of the first molecule***/
```

```
   fp1 = dt_fp_fingerprint(Testmol->molecule, FALSE);

   Testmol->tanimoto = dt_fp_tanimoto(fp,fp1);

   dt_dealloc(fp1);

/***return boolean for tanimoto within limits   ***/

   if((Testmol->tanimoto <= max) && (Testmol->tanimoto >= min)){
return(TRUE);
   }
   else{
return(FALSE);
   }

}

/****************************************************************
* get molecular weight of molecule
****************************************************************/
int get_molwt(dt_Handle mol)
{
   intmolwt = 0;
   dt_Handleatoms,
atom;
   const static floatweight[54]={
0 , /* X0*/
1.008, /* H1*/
4.003, /* H2 */
6.94, /* H3 */
9.01, /* H4 */
10.81, /* H5 */
12.011, /* H5 */
14.01, /* H5 */
16.00, /* H5 */
19.00, /* H5 */
20.18, /* H10 */
22.99, /* H10 */
24.31, /* H10 */
26.98, /* H10 */
28.09, /* H10 */
30.97, /* H10 */
32.06, /* H10 */
35.45, /* H10 */
39.95, /* H10 */
39.10, /* H10 */
40.08, /* H20 */
44.96, /* H21 */
47.90, /* H22 */
50.94, /* H23 */
52.00, /* H24 */
54.95, /* H25 */
55.85, /* H26 */
58.93, /* H27 */
58.71, /* H28 probably wrong wt */
63.55, /* H20 */
65.37, /* H30 */
69.72, /* H30 */
72.59, /* H30 */
74.92, /* H30 */
78.96, /* H30 */
79.90, /* H30 */
83.80, /* H30 */
85.47, /* H30 */
87.62, /* H30 */
88.91, /* H30 */
91.22, /* H40 */
92.91, /* H40 */
95.94, /* H40 */
98.91, /* H40 */
101.07, /* H40 */
102.91, /* H40 */
106.4, /* H40 */
107.87, /* H40 */
112.40, /* H40 */
114.82, /* H40 */
118.69, /* H50 */
121.75, /* H50 */
127.60, /* H50 */
126.90 /* H53 */
};

   if(mol == NULL_OB)
return(0);
   atoms=dt_stream(mol, TYP_ATOM);
   while (NULL_OB != (atom = dt_next(atoms))){
      molwt += weight[dt_number(atom)]; /* atomic number * 2 */
      molwt += dt_hcount(atom)*1.008;   /* number of hydrogens */
   }

   dt_dealloc(atoms);

   return(molwt);
}

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "trans.h"
#include "mol.h"
#include "debug.h"
#include "select.h"

/*******************************************************************
 * screen
 *
 * takes a reactant molecule and a transform and returns the a set of
 * paths where the transform is supposed to take place
 *
 * 6/96 ags.
 *******************************************************************/
dt_Handle *screen(molequeue *Reactant, transform *Transform)
{

   dt_Handlepathset,
paths,
path,
atoms,
atom,
expathset,
expaths,
expath,
exatoms,
exatom;
   intpathcount = 0,
flag;

/***check to see if screen is in molecule, if not return 0 ***/
/**think lots about using dt_match rather than dt_umatch**/
   if(NULL_OB == (pathset = dt_umatch(Transform->screenin, Reactant->molecule, FALSE))){
return(NULL);
   }

   else{
flag = TRUE;

/***count number of occurences in molecule, if > 1    ***/
/***    find least/most sterically hindered one.    ***/

pathset = steric(pathset, Reactant->molecule, 3, DIV_RANK_THRESHOLD);

/***check for exclusion structures which intersect inclusion path.***/

atoms = dt_stream(path, TYP_ATOM);
*********really need to loop over pathset and paths*******
      if(0 != strcmp(Transform->exclusion, "\0")){
   if(NULL_OB != (expathset = dt_match(Transform->screenout, Reactant->molecule, FALSE))){
if(NULL_OB != (expaths = dt_stream(expathset, TYP_PATH))){
   while(NULL_OB != (expath = dt_next(expaths))){
if(NULL_OB != (exatoms = dt_stream(expath, TYP_ATOM))){
   while(NULL_OB != (exatom = dt_next(exatoms))){
while(NULL_OB != (atom = dt_next(atoms))){
   if(dt_uid(atom)==dt_uid(exatom)){
flag = FALSE;
break;
   }
}
dt_reset(atoms);
   } /* end exatom */
} /* end exatoms */
   } /* end expath */
} /* end expaths */
   } /* end expathset */
} /* end if exclusion */
   } /* end else */


/***clean up pre-screening objects    ***/

   dt_dealloc(expathset);

   return(pathset);

} /* END screen */

/*******************************************************************
 * steric
 *
 * takes a pathset and a molecule and ranks the steric hinderance of
 * each of the paths in the pathset.  depending on the flag "rank",
 * steric will remove cpds above a threshold, rank all the cpds, rank
 * only the cpds below a threshold, or return the least hindered path.
 *
 * 6/96 ags.
 *******************************************************************/
dt_Handle *steric(dt_Handle *pathset, dt_Handle *mol, int threshold, int rank)
{
   dt_Handlepaths,
path,
atoms,
atom,
xatom,
xatombonds,
xatombond,
xxatom,
pathatoms,
pathatom,
bonds,
bond;
   int*score,
pathct=0,
count;

/***count the number of paths   ***/

   if(NULL_OB = (paths = dt_stream(pathset, TYP_PATH))){
fprintf(stderr, "No paths in steric.\n");
return(NULL);
   }
   while(NULL_OB != (path = dt_next(paths))){
pathct++;
   }

/***malloc and initialize score array  ***/

   if(NULL == (score = (int *)malloc(pathct*sizeof(int)))){
```

```
fprintf(stderr, "Unable to malloc %d scores in steric\n", pathct);
   }
   for(count=0;count<pathct;count++){
score[count] = -1;
   }

/***loop through paths and atoms***/

   pathct = 0;
   dt_reset(paths);
   while(NULL_OB != (path = dt_next(paths))){
    if(NULL_OB != (atoms = dt_stream(path, TYP_ATOM))){
pathatoms = dt_stream(path, TYP_ATOM);
while(NULL_OB != (atom = dt_next(atoms))){

/***look across bonds for atoms outside of current path***/

  if(NULL_OB != (bonds = dt_stream(atom, TYP_BOND))){
   while(NULL_OB != (bond = dt_next(bonds))){
    xatom = dt_xatom(atom, bond);
    dt_reset(pathatoms);
    nonpath = TRUE;
    while(NULL_OB != (pathatom = dt_next(pathatoms))){
if(pathatom = xatom){
  nonpath = FALSE;
  break;
}
     }

/***tally score and search for secondary non-path atoms***/

    if(nonpath){
     score[pathct] += (dt_aromatic(xatom) == TRUE ? DIV_STERIC_1a : DIV_STERIC_1A);
if(NULL_OB != (xbonds = dt_stream(xatom, TYP_BOND))){
 while(NULL_OB != (xbond = dt_next(xbonds))){
   xxatom = dt_xatom(xatom, xbond);
   dt_reset(pathatoms);
   nonpath = TRUE;
   while(NULL_OB != (pathatom = dt_next(pathatoms))){
    if(pathatom = xxatom){
nonpath = FALSE;
break;
    }
   }

/***if have secondary atoms tally score***/

   if(nonpath){
    score[pathct] += (dt_aromatic(xxatom) == TRUE ? DIV_STERIC_2a : DIV_STERIC_2A);
   }

/***loop through all 1ary and 2ndary atoms of every path collecting scores***/

  } /* end while xbond */
} /* end if xbonds */
    } /* end if nonpath */
   } /* end while bond */
  } /* end if bonds */
} /* end while next atom */
    } /* end if atoms */
    pathct++;
   }/* end while next path */

/***for debugging print each path with score    ***/


debug_molecule(mol);
dt_reset(paths);
pathct = 0;
while(NULL_OB != (path = dt_next(paths))){
   debug_path(path);
   fprintf(stderr, "\nPath score = %d \n\n", score[pathct]);
   pathct++;
}
} /* end steric */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#include "vbind.h"

dt_Handle   vbinding[1000];

int vbindinit(FILE *Bindingfile)
{
   int    vbindct,
   maxvbind,
   goodread;
   char    buf[MAXBUF],
   name[MAXBUF],
   smarts[MAXBUF];
   dt_Handle pattern;


/***read through bindings and get number***/

   vbindct = 0;
   while(NULL != fgets(buf, MAXBUF, Bindingfile)){
sscanf(buf, "%s %s[ !-~]", name, smarts);
if(strcmp(name,"") && strcmp(smarts,"")){
   vbindct++;
   memset(name, 0, MAXBUF);
   memset(smarts, 0, MAXBUF);
}
   }
   rewind(Bindingfile);
   maxvbind = vbindct;
```

```
/***malloc array of bindings    ***/
/*
   if(FALSE == (vbinding = (dt_Handle*)malloc(maxvbind*sizeof(dt_Handle)))){
fprintf(stderr, "\nOut of memory mallocing for vbindings, count = %d.\n\n", maxvbind);
exit(1);
   }
*/
/***read in binding    ***/

   vbindct = 0;
   while(NULL != fgets(buf, MAXBUF, Bindingfile)){
goodread = TRUE;
sscanf(buf, "%s %s[ !-~]", name, smarts);
if(strcmp(name,"") && strcmp(smarts,"")){

/***initialize bindings***/

   if(NULL_OB == (pattern = dt_smartin(strlen(smarts), smarts))){
fprintf(stderr, "Unable to convert %s to smarts.\n", smarts);
goodread = FALSE;
   }
   else if(NULL_OB == (vbinding[vbindct] = dt_alloc_vbind(strlen(name), name))){
fprintf(stderr, "Unable to alloc vbinding %s\n", name);
goodread = FALSE;
   }
   else if(FALSE == dt_setval(vbinding[vbindct], pattern)){
fprintf(stderr, "Unable to set %s to %s\n", name, smarts);
goodread = FALSE;
   }

/***increment counter if things are ok***/

   if(goodread){
vbindct++;
   }
   memset(buf, 0, MAXBUF);
   memset(name, 0, MAXBUF);
   memset(smarts, 0, MAXBUF);

}

   } /* end while fgets */

   return(vbindct);

}
#include <stdio.h>
#include <string.h>
#include "dt_smiles.h"
#include "debug.h"

/*********************************************************
* debug sequence
*********************************************************/
void debug_sequence(dt_Handle seq)
{
   dt_Handlemember;

   dt_reset(seq);
   while(NULL_OB != (member = dt_next(seq))){
debug_bond(member);
   }
   dt_reset(seq);
}
/*********************************************************
* debug atoms
*********************************************************/
void debug_atoms(dt_Handle atoms)
{
   dt_Handleatom;
   while(NULL_OB != (atom = dt_next(atoms))){
debug_atom(atom);
   }
}
/*********************************************************
* debug bonds
*********************************************************/
void debug_bonds(dt_Handle bonds)
{
   dt_Handlebond;
   while(NULL_OB != (bond = dt_next(bonds))){
debug_bond(bond);
   }
}
/*********************************************************
* debug pattern
*********************************************************/
void debug_pattern(dt_Handle pattern)
{
   dt_Handleatoms,
atom;

   atoms = dt_stream(pattern, TYP_ATOM);
   while(NULL_OB != (atom = dt_next(atoms))){
debug_atom(atom);
   }
   return;
}
/*********************************************************
* debug path
*********************************************************/
void debug_path(dt_Handle path)
{
   dt_Handleatoms,
atom;

   if(NULL_OB == (atoms = dt_stream(path, TYP_ATOM))){
```

```
fprintf(stderr,"no atoms in path\n");
return;
   }
   while(NULL_OB != (atom = dt_next(atoms))){
      debug_atom(atom);
   }
   fprintf(stderr, "\n");
}

/*********************************************************
* debug pathset
*********************************************************/
void debug_pathset(dt_Handle pathset)
{
   dt_Handlepaths,
path,
atoms,
atom;
   intlen;
   char*string,
symbol[10];

   paths = dt_stream(pathset, TYP_PATH);
   while(NULL_OB != (path = dt_next(paths))){
atoms = dt_stream(path, TYP_ATOM);
while(NULL_OB != (atom = dt_next(atoms))){
   debug_atom(atom);
}
fprintf(stderr, "\t");
   }
   fprintf(stderr, "\n");
}

/*********************************************************
* debug bond
*********************************************************/
void debug_bond(dt_Handle bond)
{
   dt_Handleatoms,
atom;
   if(NULL_OB != (atoms = dt_stream(bond, TYP_ATOM))){
while(NULL_OB != (atom = dt_next(atoms))){
   debug_atom(atom);
}
   }

   fprintf(stderr, "t=%d", dt_bondtype(bond));
}

/*********************************************************
* debug molecule
*********************************************************/
void debug_molecule(dt_Handle mol)
{
   dt_Handleatoms,
atom;
   intlen,
   char*string,
symbol[10];
   string=dt_cansmiles(&len,mol,TRUE);
   printf("%.*s\n",len,string);
   atoms=dt_stream(mol, TYP_ATOM);
   while(NULL_OB != (atom = dt_next(atoms))){
debug_atom(atom);
   }
   fprintf(stderr,"\n");
   dt_dealloc(atoms);
}

/**********************************************************
* debug atom
**********************************************************/
void debug_atom(dt_Handle atom)
{
   int    len,
   char    *string,
   symbol[10];

   if(NULL_OB == atom){
      fprintf(stderr, "Null atom.");
      return;
   }
   string = dt_symbol(&len,atom);
   strncpy(symbol,string,len);
   if(dt_aromatic(atom)) *symbol = *symbol + 'a' - 'A';
   fprintf(stderr,"%.*s(%d)", len, symbol, dt_uid(atom));
}
```

Appendix 4: CombiDOCK Datastructures and Code

Abstract

The following are selections of the datastructures and code from CombiDock version 2.0. These selection form the core of CombiDock and highlight the differences between CombiDock and the underlieing Dock algorithm. The CombiDock code grew out of version alpha 45 of the Dock4.0 code which was written entirely by Todd J.A. Ewing. The modifications and additions necessary for CombiDock version 1.0 were completed by Yax Sun. Optimization of version 1.0 and the modifications and additions which comprise CombiDock version 2.0 were completed by Allan Geoffrey Skillman, Jr.

Included Files

Datastructure Files:

dock.h

fragment.h

molecule.h

Code Files:

dock.c

fragment.c

rank.c

score.c

Datastructures:

dock.h

```
/*
Written by Todd Ewing
3/96
*/
/*
Modified by Geoff Skillman
*/
/*
* General Parameters
*/
#define ANCHOR_NEW_MAX 10 /* ***ALSO defined in definitions.h*** */

typedef struct combi_struct
{
  int flag;/* Flag for combinatorial fragment */
  int linker_lib_flag;
  int probe_flag;/* Flag for using probe fragments with the scaffold */
  double total_compounds;      /* Counter for total compounds docked */
  double total_conformations;  /* Counter for total conformations docked */

  FILE_NAME *file_name;/* List of files containing fragments */
  FILE **file;/* List of file pointers of fragments */

  FILE_NAME *probe_file_name;/* 2/99 Name of file containing probe fragments */
  FILE **probe_file;/* file containing probe fragments */

  FILE_NAME scaffold_file_name;
  FILE *scaffold_file;

  int max_sites;/* Maximum number of sites */
  int max_fragments[ANCHOR_NEW_MAX];/* Maximum number of fragments */
  int max_anchor_torsions;

  int max_probes[ANCHOR_NEW_MAX];/* Maximum number of probe fragments */

  MOLECULE_COMPOSE *molecule_compose;

} COMBI;

typedef struct client_struct
{
  int flag;/* Flag for client/server option */
  int total;/* Number of clients to whom jobs given */
  STRING20 *name;/* Names of the clients involved */

} CLIENT;

typedef struct dock_struct
{
  int restart;/* Flag to restart a job */
  int orient_ligand;/* Flag for whether ligand to be oriented */
  int multiple_ligands;/* Flag to process more than one ligand */
  int rank_ligands;/* Flag to rank ligands */
  int multiple_orients;/* Flag to write multiple ligand orientations */
  int rank_orients;/* Flag to rank orientations */
  int molecules_max;/* Maximum number of molecules to process */
  int initial_skip;/* Number of molecules to skip initially */
  int interval_skip;/* Number of molecules to skip for each read */
  int restart_interval;/* Frequency of saving restart info */

  int max_atoms;/* Maximum number of atoms */
  int max_heavies;/* Maximum number of heavy atoms*/
  int max_flexes;/* Maximum number of flexible bonds */

  FILE_NAME ligand_file_name;/* File containing ligand(s) */
  FILE_NAME receptor_file_name;/* File containing receptor */
  FILE_NAME dump_file_name;/* File by user to cause info dump */
  FILE_NAME quit_file_name;/* File by user to cause program quit */
  FILE_NAME info_file_name;/* File containing current run info */
  FILE_NAME restart_file_name;/* File where restart info written */

  FILE *ligand_file;/* File pointer for ligand(s) */

  COMBI combi;/* Combinatorial docking information */
  CLIENT client;/* Client/server information */

  int clash_cals;

} DOCK;
```

fragment.h

```
/*
Written by Todd Ewing
10/95
*/
/*
Modified by Geoff Skillman
*/
int check_clash
(
  LABEL_VDW *,
  MOLECULE *imolecule,
  MOLECULE *jmolecule
);

int write_fragments
(
  DOCK *dock,
  SCORE *score,
  LABEL *label,
  MOLECULE *molecule,
  int counter
);

void allocate_molecule_compose
(
  MOLECULE_COMPOSE *molecule_compose,
  int *fragment_total
);

void reallocate_molecule_compose
(
  MOLECULE_COMPOSE *
);

void free_molecule_compose
(
  MOLECULE_COMPOSE *
);

void merge_fragments
(
  MOLECULE *molecule
);

void add_bond (MOLECULE *molecule, int id1, int id2);
int rotate_fragment(XYZ rotation[3],XYZ origin,int current_atom,int
    previous_atom, MOLECULE *molecule);
void calc_properties(MOLECULE *mol);
void prescore_fragments(MOLECULE_COMPOSE *molecule_compose);
void merge_fragments(MOLECULE *molecule);
void add_fragment(MOLECULE *mol, MOLECULE *fragment, int site, float angle);
```

molecule.h

```
/*
Written by Todd Ewing
10/95
*/

/*
Structures to hold molecule data
10/95 te
*/

typedef struct info_struct
{
  char *name, *comment, *molecule_type, *charge_type, *status_bits;
  char *file_name;
  long file_position;
  int allocated;
  int fragment_single_allocated;
  int commentkey;  /* integer key based on comment */
  int conformation_flag;
} INFO;

typedef struct move_struct
{
  int initialized, anchored, oriented, moved, reflected;
  float rmsd;
  float initial[6], anchor[6], orient[6], final[6];
  XYZ com, anchor_com;/* Center of mass */
} MOVE;

typedef struct mol_score_struct
{
  int id, type;
  int bumpcount, iterations;
  float total, sub_total;
/***ags 3/98 - add scoring of chris lipinski "rule of 5" ***/
  float clogp;
  short amw;
  unsigned char hbd, hba;
} MOL_SCORE;

typedef struct arraysize_struct
{
  int score_parts, atoms, bonds, substs, features, sets, anchors;
  int flexes, initial_flexes;
  int sites, fragments;
} ARRAYSIZE;

typedef struct atom_struct
{
  int number, subst_id, label_id, definition_id, vdw_id, ring_id,
     flag, anchor_id;
  int *neighbor, neighbor_total;
  float charge;
  char *name, *type;
} ATOM;

typedef struct bond_struct
{
  int origin, target, flex_id;
  char *type;
} BOND;

typedef struct subst_struct
{
  int number, root_atom, dict_type, inter_bonds;
  char *name, *type, *chain, *sub_type, *status;
} SUBST;

typedef struct flex_struct
```

```
{
 int bond_id, origin, target, origin_wt, target_wt;
 float angle;
} FLEX;

typedef struct set_struct
{
 int type;
} SET;

typedef struct anchor_struct
{
 int   atom_total;
 int   atom[3];
 XYZ   coord[3];
 int   scaffold_linked;
 int   fragment_linked;
} ANCHOR_NEW;

typedef struct molecule_struct
{
 INFO info;
 MOVE move;
 MOL_SCORE score;

 ARRAYSIZE total, max;
 float *score_part;
 ATOM *atom;
 XYZ *coord, *initial_coord;
 BOND *bond;
 SUBST *subst;
 FLEX *flex, *initial_flex;
 ANCHOR_NEW *anchor;
 int *site;
 struct molecule_struct **fragment;
 struct molecule_struct *fragment_single;

} MOLECULE;

typedef struct molecule_fragment_struct
{
 int site_total;
 int *fragment_total;
 int *unique_fragment_total;
 MOLECULE scaffold;
 MOLECULE **fragment;
} MOLECULE_COMPOSE;

typedef struct linked_molecule
{
 struct linked_molecule *next_head, *next_member;
 MOLECULE *molecule;
} LINKED_MOLECULE;


/*
Routines used to manipulate molecule data structures
10/95 te
*/

void allocate_molecule   (MOLECULE *);
void allocate_info       (MOLECULE *);
void allocate_move       (MOLECULE *);
void allocate_score      (MOLECULE *);
void allocate_score_parts (MOLECULE *);
void allocate_atoms      (MOLECULE *);
void allocate_bonds      (MOLECULE *);
void allocate_substs     (MOLECULE *);
void allocate_sites      (MOLECULE *);
void allocate_fragments  (MOLECULE *);
void allocate_flexes     (MOLECULE *);
void allocate_fragment_singles (MOLECULE *);

void reset_molecule   (MOLECULE *);
void reset_info       (MOLECULE *);
void reset_move       (MOLECULE *);
void reset_score      (MOLECULE *);
void reset_score_parts (MOLECULE *);
void reset_atoms      (MOLECULE *);
void reset_atom       (ATOM *)   ;
void reset_bonds      (MOLECULE *);
void reset_bond       (BOND *)   ;
void reset_substs     (MOLECULE *);
void reset_subst      (SUBST *)  ;
void reset_sites      (MOLECULE *);
void reset_fragments  (MOLECULE *);
void reset_flexes     (MOLECULE *);

void free_molecule   (MOLECULE *);
void free_info       (MOLECULE *);
void free_move       (MOLECULE *);
void free_score      (MOLECULE *);
void free_score_parts (MOLECULE *);
void free_atoms      (MOLECULE *);
void free_atoms_only (MOLECULE *);
void free_atom_coordinates (MOLECULE *);
void free_atom       (ATOM *)   ;
void free_bonds      (MOLECULE *);
void free_bond       (BOND *)   ;
void free_substs     (MOLECULE *);
void free_subst      (SUBST *)  ;
void free_sites      (MOLECULE *);
void free_fragments  (MOLECULE *);
void free_fragment_singles  (MOLECULE *);
void free_flexes     (MOLECULE *);
void free_anchors    (MOLECULE *);

void reallocate_molecule   (MOLECULE *);
void reallocate_atoms      (MOLECULE *);
void reallocate_bonds      (MOLECULE *);
void reallocate_substs     (MOLECULE *);
void reallocate_sites      (MOLECULE *);
void reallocate_fragments  (MOLECULE *);
void reallocate_flexes     (MOLECULE *);
void reallocate_anchors    (MOLECULE *);

void copy_molecule (MOLECULE *, MOLECULE *);
void copy_info     (MOLECULE *, MOLECULE *);
void copy_move     (MOLECULE *, MOLECULE *);
void copy_score    (MOLECULE *, MOLECULE *);
void copy_atoms    (MOLECULE *, MOLECULE *);
void copy_atom     (ATOM *,    ATOM *)   ;
void copy_coord    (XYZ,       XYZ)      ;
void copy_bonds    (MOLECULE *, MOLECULE *);
void copy_bond     (BOND *,    BOND *)   ;
void copy_substs   (MOLECULE *, MOLECULE *);
void copy_subst    (SUBST *,   SUBST *)  ;
void copy_sites    (MOLECULE *, MOLECULE *);
void copy_fragments (MOLECULE *, MOLECULE *);
void copy_flexes   (MOLECULE *, MOLECULE *);
void copy_flex     (FLEX *,    FLEX *)   ;
void copy_anchor   (MOLECULE *, MOLECULE *);

void save_molecule   (MOLECULE *, FILE *);
void save_info       (MOLECULE *, FILE *);
void save_move       (MOLECULE *, FILE *);
void save_score      (MOLECULE *, FILE *);
void save_score_parts (MOLECULE *, FILE *);
void save_atoms      (MOLECULE *, FILE *);
void save_atom       (ATOM *,    FILE *);
void save_bonds      (MOLECULE *, FILE *);
void save_bond       (BOND *,    FILE *);
void save_substs     (MOLECULE *, FILE *);
void save_subst      (SUBST *,   FILE *);
void save_sites      (MOLECULE *, FILE *);
void save_fragments  (MOLECULE *, FILE *);
void save_flexes     (MOLECULE *, FILE *);

void load_molecule   (MOLECULE *, FILE *);
void load_info       (MOLECULE *, FILE *);
void load_move       (MOLECULE *, FILE *);
void load_score      (MOLECULE *, FILE *);
void load_score_parts (MOLECULE *, FILE *);
void load_atoms      (MOLECULE *, FILE *);
void load_atom       (ATOM *,    FILE *);
void load_bonds      (MOLECULE *, FILE *);
void load_bond       (BOND *,    FILE *);
void load_substs     (MOLECULE *, FILE *);
void load_subst      (SUBST *,   FILE *);
void load_sites      (MOLECULE *, FILE *);
void load_fragments  (MOLECULE *, FILE *);
void load_flexes     (MOLECULE *, FILE *);

void save_string (char **, FILE *);
void load_string (char **, FILE *);
```

Code:

dock.c

```
/**********************************************************************\
*                                                  *
*                                                  *
*        UUUUUUUUU   CCCCCCC   SSSSSSS  FF/  FFF/        *
*        UU/  UU/ CC/  CC/ SS/  SS/ FF/ FFF/         *
*       UU/  UU/ CC/  CC/ SS/     FFFFF/           *
*      UU/  UU/ CC/  CC/ SS/     FF/ FF\           *
*     UU/  UU/ CC/  CC/ SS/  SS/ FF/  FF\          *
*    UUUUUUUUU/  CCCCCCC/  SSSSSSS/  FF/   FF\         *
*                                                  *
*                                                  *
*   Copyright (C) 1991 Regents of the University of California     *
*              All Rights Reserved.              *
*                                                  *
*  This program implements the docking algorithm of I.D. Kuntz,    *
*  J Mol Biol 161, 269-288, 1982.                *
*                                                  *
*  Versions 1.0 and 1.1 of the dock program were based extensively on  *
*  the work of Robert Sheridan, Renee DesJarlais, and Tack Kuntz.   *
*                                                  *
*  Version 2.0, macrodock, is based on the work of Brian Shoichet    *
*  and Tack Kuntz, with help from Dale Bodian.           *
*                                                  *
*  Version 3.0, chemdock, is based on the work of Elaine Meng, Brian   *
*  Shoichet, and Tack Kuntz.                  *
*                                                  *
*  Version 3.5 was produced from version 3.0 by Mike Connolly and    *
*  Dan Gschwend.                          *
*                                                  *
*  Version 4.0 is based on the work of Todd Ewing and Tack Kuntz.   *
*                                                  *
*  CombiDock is based on the work of Geoff Skillman, Yax Sun, and Tack Kuntz *
*                                                  *
*                                                  *
\**********************************************************************/

#include <time.h>
#include "definitions.h"
#include "global.h"
#include "utility.h"
#include "molecule.h"
#include "grid_struc.h"
#include "list.h"
#include "dock.h"
#include "label.h"
```

```
#include "grid_score.h"
#include "score.h"
#include "match.h"
#include "io.h"
#include "io_grid.h"
#include "io_ligand.h"
#include "io_prepare.h"
#include "parm_dock.h"
#include "rank.h"
#include "rotrans.h"
#include "fragment.h"

GLOBAL global = {0};

main (int argc, char *argv[])
{
/*
* General variables (see header files for globally defined variables
* 6/95 te
*/
  int i, j, k, l;/* Counter variables */
  float angle;/* Angle for probe attachment */
  STRING80 line;            /* String used to compose output */
  float time = 0.0;/* Total amount of elapsed cpu time */
  int index[ANCHOR_NEW_MAX];/* counter from probes */
  int index_max;/* max for counter */
  int index_base[ANCHOR_NEW_MAX]; /* mod base for each index */
  int index_dims[ANCHOR_NEW_MAX]; /* size in each dimension for index */
  int rot[ANCHOR_NEW_MAX];/* counter from attachment torsions */
  int rot_max;/* max for rotation counter */
  int rot_base[ANCHOR_NEW_MAX]; /* mod base for each rotation */
  int rot_dims[ANCHOR_NEW_MAX]; /* size in each dimension for rotation */
  XYZ bond_vector, rotation[3];/* attachment bond vector & rotation */
  int minimize_tmp;/* space holder for minimize variable */
  int multiple_tmp;/* space holder for multiple variable */
  int interpolate_tmp;/* spacd holder for interpolation flag */

/*
* Variables for multiple ligands mode
* 6/95 te
*/
  int ligand_read;/* Flag for whether a ligand was read */
  int ligand_read_num = 0;/* Number of ligands read */
  int ligand_dock_num = 0;/* Number of ligands docked */
  int ligand_skip_num = 0;/* Number of ligands skipped */
  int iteration = 0;/* Counts number of processing cycles */
/* ags 11/97 counter of fragments for combi dock */
  int unique_frag_num = 0;
/*
* File pointers used when checking for the presence of control files
* 6/95 te
*/
  FILE *dump, *quit,
      *scaffold_duplicate;/* File to read scaffolds for probes */

/*
* Data structures containing docking parameters
* 2/96 te
*/
  DOCK dock = {0};/* Docking data structure */
  MATCH match = {0};/* Matching data structure */
  SCORE score = {0};/* Scoring data structure */
  LABEL label = {0};/* Labeling data structure */

  LIST ligand_orients = {0};/* List of best orientations */
  LIST best_ligands = {0};/* List of best molecules */
  LIST fragment_orients = {0};  /* List of orientations of a fragment */
  LIST best_scaffolds = {0};/* 2/99 ags List of the best scaffolds (& orientations */
  LIST scaffold_orients = {0};  /* 2/99 ags List of orientations of scaffold */

  MOLECULE ligand = {0};/* Ligand data structure */
  MOLECULE probe_fragment = {0}; /* Temporary probe fragment holder */

  MOLECULE_COMPOSE ligand_compose = {0};
             /* Ligand fragments and scaffold structure */
  MOLECULE_COMPOSE probe_compose = {0};
/* Probe fragments and scaffold structure */
  ATOM *head_frag_atom;        /* 3/98 ags ptr to atoms of initial fragment */
  BOND *head_frag_bond;/* 3/98 ags ptr to bonds of initial fragment */
/*
* Functions used in the main routine that also reside in this file.
* Other functions are declared in the header files.
* 6/95 te
*/
  void write_program_header (void);
  void set_memory_limit (void);
  float elapsed_time (float *);
  extern void get_index(int, int, int *, int *);  /* from rotrans.c */

/*
* Process command line arguments
* 6/95 te
*/
  process_commands (&dock, argc, argv);

/*
* Output program header and user information
* 6/95 te
*/
  write_program_header ();

/*
* Set ceiling on how memory can be dynamically allocated for this run
* 11/95 te
*/
  set_memory_limit ();

/*
* Read in dock parameters
* 6/95 te
*/
  if (!get_parameters
   (&dock, &match, &score, &label, &ligand_orients, &best_ligands))
  {
   fprintf (stdout, "Error reading control parameters, check output file.\n");
   exit (EXIT_FAILURE);
  }

/*
* Read in chemical label definitions
* 6/95 te
*/
  if ((label.chemical.flag || score.lipinski) &&
   !get_chemical_labels (&label.chemical))
  {
   fprintf (global.outfile, "Error reading in chemical definition file %s.\n",
    label.chemical.file_name);
   exit (EXIT_FAILURE);
  }

  if (match.chemical &&
   !generate_matchtable (&label.chemical))
  {
   fprintf (global.outfile, "Error generating chemical match table.\n");
   exit (EXIT_FAILURE);
  }

/*
* Read in chemgrid info: bump, vdw and electrostatic pot'l grids
* 6/95 te
*/
  if (score.flag.any)
  {
   fprintf (global.outfile, "Reading in chemgrid data grids.\n");

   if (!read_grids (&score.grid, &score.flag, &label.chemical))
   {
    fprintf (global.outfile, "Error reading chemgrid data grids.\n");
    exit (EXIT_FAILURE);
   }
  }

/*
* Read in vdw parameters
* 6/95 te
*/
  if (label.vdw.flag &&
   !get_vdw_labels (&label.vdw, &score.grid))
  {
   fprintf (global.outfile, "Error reading vdw parameter file %s.\n",
    label.vdw.file_name);
   exit (EXIT_FAILURE);
  }

/*
* Read in flexible bond parameters
* 10/95 te
*/
  if (label.flex.flag &&
   !get_flex_labels (&label.flex))
  {
   fprintf (global.outfile, "Error reading flexible bond file %s.\n",
    label.flex.file_name);
   exit (EXIT_FAILURE);
  }

/*
* Read in dummy atom label definitions
* 6/95 te
*/
  if (label.dummy.flag &&
   !get_dummy_labels (&label.dummy))
  {
   fprintf (global.outfile, "Error reading in dummy atom definition file %s.\n",
    label.dummy.file_name);
   exit (EXIT_FAILURE);
  }

/*
* Read in site points for matching
* 3/96 te
*/
  if (dock.orient_ligand)
   read_site_points (&dock, &match, &score, &label);

/*
* Initialize best molecule lists
* 2/96 te
*/
  allocate_list (&best_ligands);
  allocate_list (&ligand_orients);

/*
* Open ligand and fragment input files
* 6/95 te
* Open scaffold file
* 6/95 ys
*/
  if (!dock.combi.flag)
  {
   if (dock.client.flag != 'c')
    dock.ligand_file = efopen (dock.ligand_file_name, "r", global.outfile);
  }
  else
  {
   dock.combi.scaffold_file =
```

```
      efopen (dock.combi.scaffold_file_name, "r", global.outfile);



    for (i = 0; i < dock.combi.max_sites; i++)
      dock.combi.file[i] =
    efopen (dock.combi.file_name[i], "r", global.outfile);

/* 2/99 ags */
    if (dock.combi.probe_flag)
    {
      scaffold_duplicate =
        efopen (dock.combi.scaffold_file_name, "r", global.outfile);

      for(i = 0; i < dock.combi.max_sites; i++)
        dock.combi.probe_file[i] =
          efopen (dock.combi.probe_file_name[i], "r", global.outfile);
    }
  }
/*
* Open ligand and fragment output files
*/
/*  if (!dock.rank_ligands && (dock.client.flag != 's'))*/
  if (TRUE)
    for (i = 0; i < SCORE_TOTAL; i++)
      if (score.type[i].flag)
      {
        score.type[i].file =
          efopen (score.type[i].file_name, "w", global.outfile);

        if (dock.combi.flag && !score.combi.merge)
          for (j = 0; j < dock.combi.max_sites; j++)
            score.combi.type[i].file[j] =
              efopen (score.combi.type[i].file_name[j], "w", global.outfile);
      }


/*  1/98 ags this is the start of the most basic docking loop */
  while
  (
    !iteration ||
     (dock.multiple_ligands && (ligand_read_num < dock.molecules_max))
  )
  {
/****************************************************************
 *  2/99 ags this is a new experimental section for pre-docking a
 * scaffold with a set of probe fragments
 ****************************************************************/
    if (dock.combi.flag)
    {
    if(dock.combi.probe_flag && dock.orient_ligand)
    {

/*
 *read in probes 4/99 ags
 */

      probe_compose.site_total = dock.combi.max_sites;

      if(!iteration)
      {
allocate_molecule_compose (&probe_compose, dock.combi.max_probes);
      }
      score.pre_bump = 0;
      score.post_bump = 0;

      for (i = 0; i < probe_compose.site_total; i ++)
      {
rewind(dock.combi.probe_file[i]);
      }

      for (i = 0; i < dock.combi.max_sites; i ++)
      {
        for (j = 0; j < dock.combi.max_probes[i]; j++)
        {
          if ((ligand_read = read_ligand
          (
            &dock,
            &score,
            &label,
            &probe_compose.fragment[i][j],
            dock.combi.probe_file_name[i],
            dock.combi.probe_file[i],
            score.flag.any || label.flex.flag,
            (label.chemical.flag || score.lipinski),
            label.dummy.flag,
            label.vdw.flag
          )) != TRUE)
            break;
/*
 *        check if anchor atom information has been read in
 */
          if (probe_compose.fragment[i][j].anchor == NULL)
          {
            fprintf (global.outfile, "anchor atom missing. "
              "please check probe mol2 file:\n site: %d probe: %d\n",
              i, j);
            exit (EXIT_FAILURE);
          }
        } /* close j */

        if (j < probe_compose.fragment_total[i])
          probe_compose.fragment_total[i] = j;
      } /* close i */

/* set docking variable for probe docking */
      dock.combi.flag = FALSE; /* probe docking is NON-combi docking */
      multiple_tmp = dock.multiple_ligands; /* hold boolean */
```

```
      dock.multiple_ligands = TRUE; /* dock as multiple */
      minimize_tmp = score.type[SCORE_TMP].minimize; /* hold boolean */
      score.type[SCORE_TMP].minimize = score.probe_minimize; /* set min for probes */
      interpolate_tmp = score.option.interpolate;
      score.option.interpolate = 0;

/* begin potential loop around scaffolds (if no more scaffolds, END) */
      if ((read_ligand
        (
          &dock,
          &score,
          &label,
          &probe_compose.scaffold,
          dock.combi.scaffold_file_name,
          scaffold_duplicate,
          score.flag.any || label.flex.flag,
          (label.chemical.flag || score.lipinski),
          label.dummy.flag,
          label.vdw.flag
        )) != TRUE)
  {
      ligand_read_num = dock.molecules_max; /* no more molecules */

/* reset docking variables */

      dock.combi.flag = TRUE;
      dock.multiple_ligands = multiple_tmp;
      score.type[SCORE_TMP].minimize = minimize_tmp;
      score.option.interpolate = interpolate_tmp;

      break;
  }

      {
       if (probe_compose.scaffold.anchor == NULL)
       {
        fprintf (global.outfile, "anchor atom information missing. "
              "please check scaffold mol2 file\n");
        exit (EXIT_FAILURE);
       }

       ligand_read_num++;
       probe_compose.scaffold.total.sites =
         probe_compose.scaffold.max.sites = probe_compose.site_total;
      }

/*
 *    For probe dock, connect probes to the scaffold
 */
      orient_fragments (&probe_compose);
/*
 *  prepare for scoring
 */
      score.pre_bump = score.post_bump = 0;
      scaffold_orients.max[0] = score.combi.scaffold_orient_max;
      best_scaffolds.max[0] = score.combi.scaffold_greedy;

      for (i = 0; i < SCORE_TOTAL; i++)
      {
        score.type[i].number_written = 0;
scaffold_orients.max[i] = scaffold_orients.max[0];
best_scaffolds.max[i] = best_scaffolds.max[0];
      }
      ligand.max.flexes = 1;

      if(!iteration){
        allocate_molecule(&ligand);
        allocate_list(&scaffold_orients);
        allocate_list(&best_scaffolds);

/*  set up match for probe docking */
        match_setup (&dock, &match, &score, &label);
      }
      else{
reset_list(&best_scaffolds);
      }
/*
 *  2/99 ags loop through all combinations of probes and dock each one
 */
/*  malloc fragment_single for ligand to store probe fragments */


/*  set up index_max, dimension, and a mod base for index */
      index_max = 1;
      rot_max = 1;
      for (i=0;i<probe_compose.scaffold.total.sites;i++)
      {
        index_max *= probe_compose.fragment_total[i];
        rot_max *= dock.combi.max_anchor_torsions;
        index_dmns[i] = probe_compose.fragment_total[i];
        rot_dmns[i] = dock.combi.max_anchor_torsions;
      }
      for (i=0;i<probe_compose.scaffold.total.sites;i++)
      {
        index_base[i] = 1;
        rot_base[i] = 1;
        for(j=i;j<probe_compose.scaffold.total.sites;j++)
        {
  index_base[i] *= index_dmns[j];
  rot_base[i] *= rot_dmns[j];
        }
      }
/*
 *  2/99 ags create a whole molecule from probe_compose and put in ligand
 */

/* work through all combinations of probe fragments */
      for ( i = 0; i < index_max; i++)
```

```
      {
        get_index (i, probe_compose.scaffold.total_sites,
    index_dims, index); /* convert i to appropriate index [1,2,..] */

/* work through all combinations of attachment torsions */
        for(j = 0;j < rot_max; j++)
        {
  get_index (j, probe_compose.scaffold.total_sites,
    rot_dims, rot); /* convert j into rotation for each attachment */

/* copy the scaffold */
  reset_molecule(&ligand);
  copy_molecule(&ligand, &probe_compose.scaffold);

  for(k = 0;k < probe_compose.scaffold.total_sites;k++)
  {

/* add the appropriate fragment pieces */
          copy_molecule(&ligand.fragment_single[k], &probe_compose.fragment[k][index[k]]);

/* prepare for each fragment for rotation */
    ligand.flex->target = ligand.fragment_single[k].anchor[0].atom[1];
    ligand.flex->origin = ligand.fragment_single[k].anchor[0].atom[0];

    for (l=0;l<3;l++)
    {
     bond_vector[l] =
ligand.fragment_single[k].coord[ligand.flex->target][l] -
ligand.fragment_single[k].coord[ligand.flex->origin][l];
    }
    ligand.flex->angle = rot[k] * 2 * PI / dock.combi.max_anchor_torsions;

    rotateaxis
        (
          ligand.flex->angle,
          bond_vector,
          rotation
        );

/* rotate each fragment */
        rotate_fragment
        (
          rotation,
          ligand.fragment_single[k].coord[ligand.flex->target],
          ligand.flex->target,
          ligand.flex->origin,
          &ligand.fragment_single[k]
        );

  } /* end k loop through rotating each fragment */

/* make all of the fragments + scaffold into one */
  merge_fragments(&ligand);

/* prepare ligand for matching */
  get_centers (&dock, &match, &label, &ligand);

/*
*   2/99 ags standard call to match_driver (like multiple cpd docking )
*/
        match_driver
    (&dock, &match, &score, &label, &ligand, &scaffold_orients);

/*
*   2/99 ags merge the scaffold orients from this probe with the best
*/
  merge_list (&best_scaffolds, &scaffold_orients, 1);
  reset_list (&scaffold_orients);

       } /* close j--rotations loop & one in which final docking occurs */

/***************************************/
/* debug *** this is to write scaffolds */
/*
for(k=0;k<best_scaffolds.total[SCORE_TMP];k++){
fprintf (global.outfile,"writing scaffold %d \n",k);

copy_move ( &ligand, best_scaffolds.member[SCORE_TMP][k]);
copy_score ( &ligand, best_scaffolds.member[SCORE_TMP][k]);
transform_molecule(&dock,&ligand, NULL, NULL);

       write_ligand
       (
         &dock,
         &score,
         &label,
  best_scaffolds.member[SCORE_TMP][k],
  score.type[3].file_name,
  score.type[3].file,
  ++score.type[3].number_written
       );
}
*/
/***************************************/
/***************************************/
      } /* close i */

/* write probe docking information to scaffold specific info file */

      write_probe_info
      (
        &dock,
&score,
&best_scaffolds,
probe_compose.scaffold.info.name,
elapsed_time (NULL)
      );
```

```
/* set docking variables back for combi */
     dock.combi.flag = TRUE; /* reset combi flag now that were done with probe */
     dock.multiple_ligands = multiple_tmp; /* reset for combi */
     score.type[SCORE_TMP].minimize = minimize_tmp; /* reassign minimize for fragments */

    } /* end if probe && orient */

/**********************************************************************
 * 2/99 ags this is the end of the probe section & back
 * the entire combi library.
 **********************************************************************/
/*
* ys   start of combi read-in section
*/

     if(iteration)
     {
       free_molecule_compose (&ligand_compose);
unique_frag_num=0;
     }

     ligand_compose.site_total = dock.combi.max_sites;
     allocate_molecule_compose (&ligand_compose, dock.combi.max_fragments);

     for (i = 0; i < ligand_compose.site_total; i ++)
     {
rewind(dock.combi.file[i]);
     }

/*
 *read in scaffold for orientation
 */

     if (read_ligand
        (
         &dock,
         &score,
         &label,
         &ligand_compose.scaffold,
         dock.combi.scaffold_file_name,
         dock.combi.scaffold_file,
         score.flag.any || label.flex.flag,
         (label.chemical.flag || score.lipinski),
         label.dummy.flag,
         label.vdw.flag
        ) != TRUE)
     {
ligand_read_num = dock.molecules_max; /* no more molecules */
break;
     }
     else
     {
      if (ligand_compose.scaffold.anchor == NULL)
      {
       fprintf (global.outfile, "anchor atom information missing. "
              "please check scaffold mol2 file\n");
       exit (EXIT_FAILURE);
      }

      ligand_dock_num++;
      ligand_compose.scaffold.total.sites =
       ligand_compose.scaffold.max.sites = ligand_compose.site_total;
     }

/*
 *done with scaffold...read in side-chains
 */

     for (i = 0; i < dock.combi.max_sites; i ++)
     {
      unique_frag_num++;
      for (j = 0; j < dock.combi.max_fragments[i]; j++)
      {
       if ((ligand_read = read_ligand
       (
        &dock,
        &score,
        &label,
        &ligand_compose.fragment[i][j],
        dock.combi.file_name[i],
        dock.combi.file[i],
        score.flag.any || label.flex.flag,
        (label.chemical.flag || score.lipinski),
        label.dummy.flag,
        label.vdw.flag
       )) != TRUE)
        break;
/* ags 11/97 append this here rather than in score.c.  it is nice to have
*   so that you can append the info without getting a run-on when you
*   compose whole molecules   will become obsolete with multiple confor-
*   mations.
*/
  vstrcat (&ligand_compose.fragment[i][j].info.name, " ");
/* ags 11/97 set the comment key to identify unique fragments */
/* ags 3/98 set ptr to atoms of head fragment */
  ligand_compose.fragment[i][j].info.conformation_flag = FALSE;
  if((j>0)&&(0 != strcmp(ligand_compose.fragment[i][j].info.name,
 ligand_compose.fragment[i][j-1].info.name)))
   {
    unique_frag_num++;

    head_frag_atom = ligand_compose.fragment[i][j].atom;
    head_frag_bond = ligand_compose.fragment[i][j].bond;
   }

  else if(j==0)
   {
    head_frag_atom = ligand_compose.fragment[i][j].atom;
```

```
    head_frag_bond = ligand_compose.fragment[i][j].bond;
   }

/* ags 3/98 if not head, free atoms and point to head fragment */
  else{
    free_atoms_only(&(ligand_compose.fragment[i][j]));
    free_bonds(&(ligand_compose.fragment[i][j]));
    ligand_compose.fragment[i][j].atom = head_frag_atom;
    ligand_compose.fragment[i][j].bond = head_frag_bond;
/* 4/99 set flag that this is a conformation rather than real molecule */
    ligand_compose.fragment[i][j].info.conformation_flag = TRUE;
   }
  ligand_compose.fragment[i][j].info.commentkey = unique_frag_num;
/*
*        check if anchor atom information has been read in
*/
        if (ligand_compose.fragment[i][j].anchor == NULL)
        {
         fprintf (global.outfile, "anchor atom missing, "
           "please check fragment mol2 file.\n site: %d fragment. %d\n",
           i, j);
         exit (EXIT_FAILURE);
        }
       } /* close j (fragments)*/

       if (j < ligand_compose.fragment_total[i])
        ligand_compose.fragment_total[i] = j;

if (i == 0){
  ligand_compose.unique_fragment_total[i] = unique_frag_num;
}
else{
  ligand_compose.unique_fragment_total[i] =
    unique_frag_num - l;
}
l = unique_frag_num; /* l is # to calc difference in unique_frag */
     } /* close i (attachment sites)*/

/*
*     For scaffold-based combi. dock, connect fragments to the scaffold
*/
     orient_fragments (&ligand_compose);
/*
*3/98 ags - prepare library for "rule of 5" scoring
*/
     prescore_fragments (&ligand_compose);
    } /* end of combi */
/*
*   Reset the score records
*   8/95 te
*/
    score.pre_bump = score.post_bump = 0;

    for (i = 0; i < SCORE_TOTAL, i++)
     score.type[i].number_written = 0,


/*
*   ys 2/97
*
*   Update the clock
*/
    elapsed_time (NULL);

/*
*   Perform docking calculation
*   10/95 te
*/
/* 12/1 ags this is still inside the "Read in ligand" loop */
    if (dock.orient_ligand)
    {

/*
*       Initialize variables used for matching
*       10/95 te
*/
       if ((!iteration)&&(!dock.combi.probe_flag))
         match_setup (&dock, &match, &score, &label);

/*
*     do the combinatorial dock here
*/
      if (dock.combi.flag)
      {
       dock.combi.molecule_compose = &ligand_compose;

       if (dock.orient_ligand &&
!match.centers && !dock.combi.probe_flag &&
         !get_centers (&dock, &match, &label, &ligand_compose.scaffold));

if (dock.combi.probe_flag) /* 2/99 ags combi dock with probed scaffold orients */
{
   /* loop through scaffold orients */
   for (i = 0 ; i < best_scaffolds.total[SCORE_TMP]; i++)
   {
/* copy next scaffold's orient to ligand_compose */
/*copy_member ( &ligand_compose.scaffold,
best_scaffolds.member[SCORE_TMP][i]).
*/
copy_move ( &ligand_compose.scaffold,
best_scaffolds.member[SCORE_TMP][i]).

/* transform ligand_compose to scaffold's */
transform_molecule(&dock,&ligand_compose.scaffold,
NULL, NULL);

/* do single point combi for each */
dock.combi.molecule_compose = &ligand_compose;
evaluate_score (&dock, &score, &label,
&ligand_compose.scaffold, &ligand_orients, 0);

    }
}
else    /* just do regular no-probe combidock */
{
        match_driver
          (&dock, &match, &score, &label,
          &ligand_compose.scaffold, &ligand_orients);

}

      }

    } /* end "if dock.orient_ligand" */

/*
*  If the ligand was not to be oriented, then do a single point score
*  calculation
*  6/95 te
*/
   else if (dock.combi.flag)
   {
    dock.combi.molecule_compose = &ligand_compose;
    evaluate_score (&dock, &score, &label,
      &ligand_compose.scaffold, &ligand_orients, 0);
   }

   else
    evaluate_score (&dock, &score, &label, &ligand, &ligand_orients, 0);

/*
*  Write/store the best orientation(s) of this molecule
*  if it wasn't written out in the scoring function.
*  3/96 te
*/
   if (dock.rank_ligands || !dock.multiple_orients || dock.rank_orients)
   {
/*
*    Either store the best orientation(s)
*    3/96 te
*/
/* update best orients, but set minimum score to limit length of
*  ligand_orients in score.c ags 4/99 */

    if (dock.rank_ligands){

/* output info for each libraries performance */
      write_scaffold_info
      (
       &dock,
       &score,
       &ligand_orients,
 &ligand_compose,
       elapsed_time (NULL)
      );

/* update the overall list & set up score limit for next library */
merge_list (&best_ligands, &ligand_orients, 0);
reset_list (&ligand_orients);
if (best_ligands.total[SCORE_TMP] == best_ligands.max[SCORE_TMP]){
 score.type[SCORE_TMP].maximum =
 best_ligands.member[SCORE_TMP][best_ligands.total[SCORE_TMP]-1]->score.total;
}
    }


/*
*    Or write the best orientation(s) out to a file
*    3/96 te
*/
    else
     write_topscorers
     (
      &dock,
      &match,
      &score,
      &label,
      &ligand_orients,
      &ligand
     );
   } /* end short if rank & !multiple */

/*
*  Print out the final results
*  10/95 te
*/
/* 1/98 ags this gives garbage for combi--it writes to global.out */
   output_score_info
    (&dock, &score, &ligand, &ligand_orients, ligand_dock_num,
    elapsed_time (NULL));


   if (dock.orient_ligand && dock.multiple_ligands)
   {
/*
*    Output list of current top scorers
*    6/95 te
*/
    write_info
    (
     &dock,
     &score,
     &best_ligands,
     ligand_read_num,
     ligand_dock_num,
     ligand_skip_num,
     elapsed_time (NULL)
```

```
    ),

/*
*    Check to see if this job has been requested to shut down
*    6/95 te
*/
    if (quit = fopen (dock.quit_file_name, "r"))
    {
     fclose (quit),
     if (remove (dock.quit_file_name))
       fprintf
        (global.outfile, "* * * Unable to delete %s.\n", dock.quit_file_name),
    }

    if (dock.rank_ligands)
    {
/*
*     Check to see if this job has been requested to write its results so far
*     6/95 te
*/
      if (dump = fopen (dock.dump_file_name, "r"))
      {
       fclose (dump),
       if (remove (dock.dump_file_name))
        fprintf
         (global.outfile, "* * * Unable to delete %s.\n", dock.dump_file_name);
      }

      if (dump || quit)
      {
       write_topscorers
       (
        &dock,
        &match,
        &score,
        &label,
        &best_ligands,
        &ligand
       );

       fprintf (global.outfile,
        "* * * Current top scorers were written as requested.\n");
      }

      dump = NULL;
     } /* end if dock.rank_ligands */

     if (quit)
     {
      fprintf (global.outfile, "* * * Program has terminated as requested.\n");
      fclose (dock.ligand_file),

      fclose (global.outfile),
      exit (EXIT_SUCCESS);
     } /* end if quit */
    }/* end if (orient && multiple) */

   reset_list(&ligand.orients);

   iteration++,
  } /* end big loop-- while 1st time or multiple & < max */

/*
* Finished reading ligands in from disk, make sure at least one was read
* 6/95 te
*/
  if (ligand.read_num == 0)
  {
   fprintf (global.outfile, "Unable to read from ligand coordinate file.\n");
   exit (EXIT_FAILURE);
  }

/*
* Write out final results (in dock.rank_ligands mode)
* 6/95 te
*/
  if (dock.rank_ligands)
  {

   fprintf (global.outfile, "Writing top scoring molecules to disk.\n"),

   write_topscorers
   (
    &dock,
    &match,
    &score,
    &label,
    &best_ligands,
    &ligand
   ),

  } /* end if rank */

/*
* Close ligand output files
* 6/95 te
*/
  else
  {
   for (i = 0, i < SCORE_TOTAL; i++)
    if (score.type[i].flag)
    {
     fclose (score.type[i].file),

     if (dock.combi.flag && !score.combi.merge)
      for (j = 0, j < dock.combi.max_sites; j++)
       fclose (score.combi.type[i].file[j]),
    }
 }

 fprintf (global.outfile,
  "Finished processing molecule%s.\n",
  ligand_dock_num > 1 ? "s" : "");

 fclose (dock.ligand_file);
 fclose (global.outfile);
}


/* ////////////////////////////////////////////////////////////

Evaluate how much time has elapsed (with checking for wrap-around)
6/95 te

//////////////////////////////////////////////////////////// */

float elapsed_time (float *reset_value)
{
 static long clock_previous = 0;
 static long clock_current = 0;
 static float time;

 if (reset_value)
  time = *reset_value;

 clock_previous = clock_current;
 clock_current = clock ();

 if ((clock_previous > 0) && (clock_current < 0))
  time +=
   ((float) ((LONG_MAX - clock_previous) + (clock_current - LONG_MIN)))
   / ((float) CLOCKS_PER_SEC);

 else
  time +=
   ((float) (clock_current - clock_previous))
   / ((float) CLOCKS_PER_SEC);

 return time;
}


/* //////////////////////////////////////////////////////////// */

void write_program_header (void)
{
 if (global.outfile != stdout)
  fprintf (global.outfile, "\n"
   "    UUUUUUUUU    CCCCCCC    SSSSSSS   FF/   FFF/ \n"
   "    UU/   UU/ CC/   CC/ SS/   SS/ FF/ FFF/   \n"
   "   UU/   UU/ CC/   CC/ SS/       FFFFF/     \n"
   "  UU/   UU/ CC/   CC/ SS/       FF/ FF\\     \n"
   " UU/   UU/ CC/   CC/ SS/   SS/ FF/  FF\\    \n"
   " UUUUUUUUU/   CCCCCCC/   SSSSSSS/  FF/    FF\\   \n\n\n");

 else
  fprintf (global.outfile, "\n\n"
   "    UUU[4mUUUU[0mUU    C[4mCCCC[0mCC    S[4mSSSS[0mSS    FF/   FFF/ \n"
   "    UU/   UU/ CC/   CC/ SS/   SS/ FF/ FFF/   \n"
   "   UU/   UU/ CC/   CC/ SS/       FFFFF/     \n"
   "  UU/   UU/ CC/   CC/ SS/       FF/ FF\\     \n"
   " UU/   UU/ CC/   CC/ SS/   SS/ FF/  FF\\    \n"
  " [4mUUUUUUUUU[0m/  [4mCCCCCCC[0m/  [4mSSSSSSS[0m/  [4mFF[0m/   [4mFF\\[0m
\n\n\n"),

 fprintf (global.outfile,
  "University of California at San Francisco, DOCK %s\n", DOCK_VERSION);

 fflush (global.outfile);
}
```

fragment.c

```
/*
Written by Todd Ewing
10/95
*/
/*
Modified by Yax Sun and Geoff Skillman
*/
#include "definitions.h"
#include "utility.h"
#include "molecule.h"
#include "global.h"
#include "grid_struc.h"
#include "list.h"
#include "dock.h"
#include "label.h"
#include "grid_score.h"
#include "score.h"
#include "fragment.h"
#include "io_ligand.h"
#include "io_prepare.h"
#include "rotrans.h"
#include "vector.h"
#include "orient.h"

#define FRAGMENT_MAX_ATOMS    100;

void free_conformation
(
 MOLECULE *molecule
);

/* ////////////////////////////////////////////////////////////
```

```
Routine to check if two molecules clash.
8/95 te

//////////////////////////////////////////////////////////////// */
int check_clash
(
  LABEL_VDW *label_vdw,
  MOLECULE *imolecule,
  MOLECULE *jmolecule
)
{
  int i, j, k;
  float distance, reference;

  for (i = 0; i < imolecule->total.atoms; i++)
    if (label_vdw->member[imolecule->atom[i].vdw_id].heavy_id)
      for (j = 0; j < jmolecule->total.atoms; j++)
        if (label_vdw->member[jmolecule->atom[j].vdw_id].heavy_id)
        {
          for (k = 0, distance = 0.0; k < 3; k++)
            distance += SQR
              (imolecule->coord[i][k] - jmolecule->coord[j][k]);

          reference = 0.6 *
            (label_vdw->member[imolecule->atom[i].vdw_id].radius +
            label_vdw->member[jmolecule->atom[j].vdw_id].radius);

          reference = SQR (reference);

          if (distance < reference)
            return TRUE;
        }

  return FALSE;
}


/* //////////////////////////////////////////////////////////////// */

void allocate_molecule_compose
  (MOLECULE_COMPOSE *molecule_compose, int *fragment_total)
{
  int i;

/* allocate scaffold */

  ecalloc
  (
    (void **) &molecule_compose->scaffold,
    1,
    sizeof (MOLECULE),
    "molecule compose",
    global.outfile
  );

  molecule_compose->scaffold.max.sites = molecule_compose->site_total;
  allocate_molecule(&molecule_compose->scaffold);

/* allocate fragments */

  ecalloc
  (
    (void **) &molecule_compose->fragment_total,
    molecule_compose->site_total,
    sizeof (int),
    "fragment_total array",
    global.outfile
  );

  ecalloc
  (
    (void **) &molecule_compose->unique_fragment_total,
    molecule_compose->site_total,
    sizeof (int),
    "unique_fragment_total array",
    global.outfile
  );

  if (molecule_compose->site_total > 0)
  {

    ecalloc
    (
      (void **) &molecule_compose->fragment,
      molecule_compose->site_total,
      sizeof (MOLECULE *),
      "molecule compose",
      global.outfile
    );

    for (i = 0; i < molecule_compose->site_total; i++)
    {

      molecule_compose->fragment_total[i] = fragment_total[i];

      if (molecule_compose->fragment_total[i] > 0)
      ecalloc
      (
          (void **) &molecule_compose->fragment[i],
          molecule_compose->fragment_total[i],
          sizeof (MOLECULE),
          "molecule compose",
          global.outfile
      );

      memset(molecule_compose->fragment[i], 0,
sizeof(MOLECULE)*molecule_compose->fragment_total[i]/sizeof(char));
    }
  }
}

/* //////////////////////////////////////////////////////////////// */

void reallocate_molecule_compose
  (MOLECULE_COMPOSE *molecule_compose)
{
  int i;


  molecule_compose->scaffold.max.sites = molecule_compose->site_total;
  reallocate_molecule (&molecule_compose->scaffold);

  if (molecule_compose->site_total > 0)
  {
    ecalloc
    (
      (void **) &molecule_compose->fragment,
      molecule_compose->site_total,
      sizeof (MOLECULE *),
      "molecule compose",
      global.outfile
    );

    for (i = 0; i < molecule_compose->site_total; i++)
    {
      if (molecule_compose->fragment_total[i] > 0)
      ecalloc
      (
          (void **) &molecule_compose->fragment[i],
          molecule_compose->fragment_total[i],
          sizeof (MOLECULE),
          "molecule compose",
          global.outfile
      );

      memset(molecule_compose->fragment[i], 0,
sizeof(MOLECULE)*molecule_compose->fragment_total[i]/sizeof(char));
    }
  }
}

/* //////////////////////////////////////////////////////////// */

void free_molecule_compose
  (MOLECULE_COMPOSE *molecule_compose)
{
  int i, j;

  free_molecule (&molecule_compose->scaffold);

  for (i = 0; i < molecule_compose->site_total; i ++)
  {
    for (j = 0; j < molecule_compose->fragment_total[i]; j ++)
    {
      if(molecule_compose->fragment[i][j].info.conformation_flag)
      {
free_conformation (&molecule_compose->fragment[i][j]);
      }
      else
      {
free_molecule (&molecule_compose->fragment[i][j]);
      }
    }

    efree ((void **) &molecule_compose->fragment[i]);
  }

  efree ((void **) &molecule_compose->fragment_total);
  efree ((void **) &molecule_compose->unique_fragment_total);

  efree ((void **) &molecule_compose->fragment);
  molecule_compose->fragment = NULL;
}

/* //////////////////////////////////////////////////////////// */

void free_conformation (MOLECULE *molecule)
{
  free_info (molecule);
  free_move (molecule);
  free_score (molecule);
  free_atom_coordinates (molecule);
  molecule->atom = NULL;
  molecule->bond = NULL;
  free_substs (molecule);
  free_sites (molecule);
  free_fragment_singles (molecule);
  free_flexes (molecule);
  free_anchors (molecule);
}

/* //////////////////////////////////////////////////////////// */

void copy_molecule_compose
(
  MOLECULE_COMPOSE *copy_molecule_compose,
  MOLECULE_COMPOSE *molecule_compose,
  int             max_fragments
)
{
  int   i, j;

  copy_molecule_compose->site_total = molecule_compose->site_total;

  copy_molecule (&copy_molecule_compose->scaffold, &molecule_compose->scaffold);
```

```
 for (i = 0; i < molecule_compose->site_total; i ++)
 {
  copy_molecule_compose->fragment_total[i]
     = molecule_compose->fragment_total[i];
  copy_molecule_compose->unique_fragment_total[i]
     = molecule_compose->unique_fragment_total[i];
  for (j = 0; (j < max_fragments) &&
  (j < molecule_compose->fragment_total[i]); j ++)
   copy_molecule (&copy_molecule_compose->fragment[i][j],
    &molecule_compose->fragment[i][j]);
 }
}

/* ////////////////////////////////////////////////////// */
/* ags */
void merge_fragments
(
 MOLECULE *molecule
)
{
 int i, j;
 int fragment_id;
 int bonded_atom_total = 0;
 int bonded_atom[2][10] = {0};
 int tmp_id, tmp2_id;

 STRING20 score_string;

 MOLECULE molecule_tmp={0};

 molecule_tmp.max_atoms = molecule->total.atoms;
 molecule_tmp.max_bonds = molecule->total.bonds;

 for (i = 0; i < molecule->total.sites; i ++)
 {
  molecule_tmp.max_atoms +=
   molecule->fragment_single[i].total.atoms;

  molecule_tmp.max_bonds +=
   molecule->fragment_single[i].total.bonds + 2;
/*
*  +2 for adding bonds later connecting fragments and scaffolds
*/
 }

 allocate_molecule (&molecule_tmp);

 copy_info (&molecule_tmp, molecule);
 copy_move (&molecule_tmp, molecule);
 copy_score (&molecule_tmp, molecule);
 copy_substs (&molecule_tmp, molecule);
 copy_sites (&molecule_tmp, molecule);
 copy_anchor (&molecule_tmp, molecule);

/*
* use "flag" in atom structure to temporaryly hold the new atom number
*/

/*
* copy atoms and coords
*/
 for (i = 0; i < molecule->total.atoms; i ++)
 {
  if (!molecule->atom[i].anchor_id)
  {
   molecule->atom[i].flag = molecule_tmp.total.atoms;

   copy_atom (&molecule_tmp.atom[molecule_tmp.total.atoms],
    &molecule->atom[i]);

   copy_coord (molecule_tmp.coord[molecule_tmp.total.atoms],
    molecule->coord[i]);

   copy_coord (molecule_tmp.initial_coord[molecule_tmp.total.atoms ++],
    molecule->initial_coord[i]);
  }
 }

 for (i = 0; i < molecule->total.sites; i ++)
 {
  for (j = 0; j < molecule->fragment_single[i].total.atoms; j ++)
  {
   if (!molecule->fragment_single[i].atom[j].anchor_id)
   {
    molecule->fragment_single[i].atom[j].flag =
     molecule_tmp.total.atoms;

    copy_atom (&molecule_tmp.atom[molecule_tmp.total.atoms],
     &molecule->fragment_single[i].atom[j]);

    copy_coord (molecule_tmp.coord[molecule_tmp.total.atoms],
     molecule->fragment_single[i].coord[j]);

    copy_coord (molecule_tmp.initial_coord[molecule_tmp.total.atoms ++],
     molecule->fragment_single[i].initial_coord[j]);
   }
  }
 }


/*
* copy bonds
*/
 for (i = 0; i < molecule->total.bonds; i ++)
 {
  tmp_id = molecule->bond[i].target;
  tmp2_id = molecule->bond[i].origin;

  if ((molecule->atom[tmp_id].anchor_id == 0)
   && (molecule->atom[tmp2_id].anchor_id == 0))
  {
   copy_bond (&molecule_tmp.bond[molecule_tmp.total.bonds],
    &molecule->bond[i]);

   molecule_tmp.bond[molecule_tmp.total.bonds].origin =
    molecule->atom
    [molecule_tmp.bond[molecule_tmp.total.bonds].origin].flag;

   molecule_tmp.bond[molecule_tmp.total.bonds].target =
    molecule->atom
    [molecule_tmp.bond[molecule_tmp.total.bonds].target].flag;

   molecule_tmp.total.bonds ++;
  }
 }

 for (i = 0; i < molecule->total.sites; i ++)
 {
  for (j = 0; j < molecule->fragment_single[i].total.bonds; j ++)
  {
   tmp_id = molecule->fragment_single[i].bond[j].target;
   tmp2_id = molecule->fragment_single[i].bond[j].origin;

   if ((molecule->fragment_single[i].atom[tmp_id].anchor_id == 0)
    && (molecule->fragment_single[i].atom[tmp2_id].anchor_id == 0))
   {
    copy_bond (&molecule_tmp.bond[molecule_tmp.total.bonds],
     &molecule->fragment_single[i].bond[j]);

    molecule_tmp.bond[molecule_tmp.total.bonds].origin =
     molecule->fragment_single[i].atom
     [molecule_tmp.bond[molecule_tmp.total.bonds].origin].flag;

    molecule_tmp.bond[molecule_tmp.total.bonds].target =
     molecule->fragment_single[i].atom
     [molecule_tmp.bond[molecule_tmp.total.bonds].target].flag;

    molecule_tmp.total.bonds ++;
   }
  }
 }

 for (i = 0; i < molecule->total.sites; i++)
 {
  add_bond (&molecule_tmp,
   molecule->atom[molecule->anchor[i].atom[1]].flag,
   molecule->fragment_single[i].atom
    [molecule->fragment_single[i].anchor[0].atom[1]].flag);
/*
* used "0" above, DEBUG
*/
 }

 vstrcpy (&molecule_tmp.info.charge_type, "USER_CHARGE");
 vstrcpy (&molecule_tmp.info.status_bits, "****");

 vstrcat (&molecule_tmp.info.name, " ");
 for (i = 0; i < molecule->total.sites; i ++)
 {
/*
  vstrcat (&molecule_tmp.info.comment, " ");
  vstrcat (&molecule_tmp.info.comment,
   molecule->fragment_single[i].info.comment);
*/
  vstrcat (&molecule_tmp.info.name,
   molecule->fragment_single[i].info.name);
/*
  sprintf (score_string, " %.2f ",
   molecule->fragment_single[i].score.total);

  vstrcat (&molecule_tmp.info.comment, score_string);
*/
 }

/*
* update atom neighbor information
*/
 atom_neighbors (&molecule_tmp);

 copy_molecule (molecule, &molecule_tmp);
 free_molecule (&molecule_tmp);
}

/* ////////////////////////////////////////////////////// */

void add_bond (MOLECULE *molecule, int id1, int id2)
{
 molecule->bond[molecule->total.bonds].origin = id1;
 molecule->bond[molecule->total.bonds].target = id2;

 vstrcpy (&molecule->bond[molecule->total.bonds].type, "1");

 molecule->total.bonds ++;
}


/* ////////////////////////////////////////////////////////

initial orientation of fragments onto scaffold.

1/97
ysun

///////////////////////////////////////////////////////// */
```

```
void orient_fragments (MOLECULE_COMPOSE *molecule_compose)
{
  int  i, j, k;
  int  node_count;
  int  check_chirality=0, mirror_ligand, match_chirality;

  XYZ  rotation_matrix[3];
  XYZ  connect_atom[4];
  float  bond, angle, torsion;

  MOLECULE  scaffold_clique = {0};
  MOLECULE  fragment_clique = {0};

  scaffold_clique.max_atoms = molecule_compose->scaffold.max_atoms;
  fragment_clique.max_atoms = FRAGMENT_MAX_ATOMS;

  allocate_molecule (&scaffold_clique);
  allocate_molecule (&fragment_clique);

  for (i = 0; i < molecule_compose->site_total; i ++)
  {
/*
*   now generating new position
*/
    copy_coord (connect_atom[0], molecule_compose->scaffold.
      anchor[i].coord[2]);
    copy_coord (connect_atom[1], molecule_compose->scaffold.
      anchor[i].coord[1]);
    copy_coord (connect_atom[2], molecule_compose->scaffold.
      anchor[i].coord[0]);

/*
    gen_4th_atom (connect_atom, bond, angle, torsion);
*/

    for (j = 0; j < molecule_compose->fragment_total[i]; j ++)
    {
      for (node_count= 0; node_count < molecule_compose->
        fragment[i][j].anchor[0].atom_total; node_count ++)
        copy_coord (fragment_clique.coord[node_count],
          molecule_compose->fragment[i][j].anchor[0].coord[node_count]);

      bond = dist3 (fragment_clique.coord[1], fragment_clique.coord[2]);

      angle = angle_from_crd
      (
        fragment_clique.coord[0],
        fragment_clique.coord[1],
        fragment_clique.coord[2]
      );

      torsion = PI;

      gen_4th_atom (connect_atom, bond, angle, torsion);

      center_of_mass
      (
        fragment_clique.coord,
        node_count,
        molecule_compose->fragment[i][j].move.anchor_com
      );

      if (!orient_gk_
      (
        &node_count,
        connect_atom[1],
        fragment_clique.coord,
        molecule_compose->fragment[i][j].move.anchor_com,
        rotation_matrix,
        molecule_compose->fragment[i][j].move.anchor,
        &check_chirality,
        &mirror_ligand,
        &molecule_compose->fragment[i][j].move.reflected
      ))
        return ;

/*
*    Transform the fragment coordinates
*/
      transform
      (
        &molecule_compose->fragment[i][j].total_atoms,
        molecule_compose->fragment[i][j].initial_coord,
        molecule_compose->fragment[i][j].move.anchor_com,
        rotation_matrix,
        molecule_compose->fragment[i][j].move.anchor,
        molecule_compose->fragment[i][j].coord
      );

/*
*    clean up
*/
      molecule_compose->fragment[i][j].move.anchored = TRUE;

      get_angles_from_matrix
        (&molecule_compose->fragment[i][j].move.anchor[3], rotation_matrix);

      for (k = 0; k < molecule_compose->fragment[i][j].total_atoms; k ++)
        copy_coord (molecule_compose->fragment[i][j].initial_coord[k],
          molecule_compose->fragment[i][j].coord[k]);

      center_of_mass
      (
        molecule_compose->fragment[i][j].initial_coord,
        molecule_compose->fragment[i][j].total_atoms,
        molecule_compose->fragment[i][j].move.com
      );
    }
  }

  free_molecule (&scaffold_clique);
  free_molecule (&fragment_clique);
}

/* ////////////////////////////////////////////////////////////
initial calculation of amw, hbd, sum of O's and N's.
data taken from the program div ags 96
3/98 ags
//////////////////////////////////////////////////////////// */
void calc_properties (MOLECULE *mol)
{
  int i;

  float amw=0.0;
  float clogp=0.0;
  int hbd=0;
  int hba=0;
  int atomic_no;
  const static float  weight[54]={
      0.,                /* X    0 */
      1.008,              /* H    1 */
      4.003,              /* H    2 */
      6.94,              /* H    3 */
      9.01,              /* H    4 */
      10.81,              /* H    5 */
      12.011,              /* H    6 */
      14.01,              /* H    7 */
      16.00,              /* H    8 */
      19.00,              /* H    9 */
      20.18,              /* H   10 */
      22.99,              /* H   11 */
      24.31,              /* H   12 */
      26.98,              /* H   13 */
      28.09,              /* H   14 */
      30.97,              /* H   15 */
      32.06,              /* H   16 */
      35.45,              /* H   17 */
      39.95,              /* H   18 */
      39.10,              /* H   19 */
      40.08,              /* H   20 */
      44.96,              /* H   21 */
      47.90,              /* H   22 */
      50.94,              /* H   23 */
      52.00,              /* H   24 */
      54.95,              /* H   25 */
      55.85,              /* H   26 */
      58.93,              /* H   27 */
      58.71,              /* H   28 */
      63.55,              /* H   20 */
      65.37,              /* H   30 */
      69.72,              /* H   30 */
      72.59,              /* H   30 */
      74.92,              /* H   30 */
      78.96,              /* H   30 */
      79.90,              /* H   30 */
      83.80,              /* H   30 */
      85.47,              /* H   30 */
      87.62,              /* H   30 */
      88.91,              /* H   30 */
      91.22,              /* H   40 */
      92.91,              /* H   40 */
      95.94,              /* H   40 */
      98.91,              /* H   40 */
      101.07,                  /* H   40 */
      102.91,                  /* H   40 */
      106.4,              /* H   40 */
      107.87,                  /* H   40 */
      112.40,                  /* H   40 */
      114.82,                  /* H   40 */
      118.69,                  /* H   50 */
      121.75,                  /* H   50 */
      127.60,                  /* H   50 */
      126.90              /* H   53 */
      };

    static int type[150];

    type[68] = 0;  /* Dummy */
    type[72] = 1;  /* H */
    type[67] = 6;  /* C */
    type[78] = 7;  /* N */
    type[79] = 8;  /* O */
    type[80] = 15;  /* P */
    type[83] = 16;  /* S */
    type[70] = 19;  /* F */
    type[0] = 17;  /* Cl  done as special case later*/
    type[66] = 35;  /* Br */
    type[73] = 53;  /* I */


  for (i = 0; i < mol->total.atoms; i ++){

/***determine atom type***/

    atomic_no = type[(int)mol->atom[i].type[0]]; /* get atomic number for atom */
    if((atomic_no == 6)&&((int)mol->atom[i].type[1] == 108)){
atomic_no = 17; /* differentiate Carbon and Chlorine */
    }

/***get number of hydrogen bond acceptors    ***/

    if(mol->atom[i].label_id == 3){
        hba++;
    }

/***get molecular weight***/
```

```
    amw += weight[atomic_no]; /* sum the mw */

/***get hydrogen-bond donors    ***/

    if((mol->atom[i].label_id == 2)|| /* chem.defn of donor */
      ((mol->atom[i].label_id == 4)&&(mol->atom[i].definition_id == 0))){
  hbd++;  /* chem defn of polar && 0 3 */
    }

/*** NOTE: no clogp calculation for now***/

  }

/*printf ("hba = %d amw = %d hbd = %d\n", hba, (int)amw, hbd); */
  mol->score.hba = hba;
  mol->score.amw = (int) amw;
  mol->score.hbd = hbd;

}

/* ////////////////////////////////////////////////////////////
initial calculation of amw, hbd, sum of O's and N's.

3/98 ags
//////////////////////////////////////////////////////////// */
void prescore_fragments (MOLECULE_COMPOSE *molecule_compose)
{

  int  i, j, k;

  calc_properties(&molecule_compose->scaffold);

  for (i = 0; i < molecule_compose->site_total; i ++)
  {
   for (j = 0; j < molecule_compose->fragment_total[i]; j ++)
   {
     calc_properties(&molecule_compose->fragment[i][j]);


   }
  }

}

/* ////////////////////////////////////////////////////////////

2/97
ysun

//////////////////////////////////////////////////////////// */

int combi_clash
(
  DOCK *dock,
  SCORE *score,
  LABEL *label,
  MOLECULE *molecule_original,
  LIST *fragments,
  int *index
)
{
  int i, j, k;

  dock->clash_cals++;
/*
* ags
*/
  if ((dock->clash_cals < 10000)&&(dock->clash_cals % 250 == 1))
  {
   fprintf (global.outfile,
     "total clash checks done = %d  \n", dock->clash_cals);
   fflush (global.outfile);
  }
  if (dock->clash_cals % 10000 == 0)
  {
   fprintf (global.outfile,
     "total clash checks done = %d   \n", dock->clash_cals);
   fflush (global.outfile);
  }

  for(i = 0; i < molecule_original->total.sites; i++)
  {
/*
* 10/97 ags  add fragment in/dependence
*/

    for (j = i+1; j < molecule_original->total.sites; j++)
     if(score->combi.dependence[i+1][j+1])
     {
/*       if(molecule_clash(label, &molecule_original->fragment_single[i], 0,
         &molecule_original->fragment_single[j], 0)) */
       if(fragment_clash(label, fragments[i].member[SCORE_TMP][index[i]],
 fragments[j].member[SCORE_TMP][index[j]]))
       {
        return TRUE;
}
     }
  }
  return 0;
}

/* clash check for while ignoring anchor points - ags */
int molecule_clash(LABEL *label, MOLECULE *molecule1, int anchor1,
  MOLECULE *molecule2, int anchor2)
{

  int i, j, k;
```

```
  float dist2;

  for (i=0; i < molecule1->total.atoms; i++)
  {
/* ags 11/97 simplify heavy atom check for speed */
/*    if(label->vdw.member[molecule1->atom[i].vdw_id].heavy_id &&  */
/*    "H" is 72, "D" as in "Du" is 68 */
   if(( (int)molecule1->atom[i].type[0] != 72)&&
     ( (int)molecule1->atom[i].type[0] != 68)&&
     (i != molecule1->anchor[anchor1].atom[0])&&
     (i != molecule1->anchor[anchor1].atom[1]))
   {
    for(j = 0; j < molecule2->total.atoms; j ++)
    {
     if(( (int)molecule2->atom[j].type[0] != 72)&&
       ( (int)molecule2->atom[j].type[0] != 68)&&
       (j != molecule2->anchor[anchor2].atom[0])&&
  (j != molecule2->anchor[anchor2].atom[1]))
      {
       dist2 = square_distance (molecule1->coord[i],
        molecule2->coord[j]);

       if(dist2 < 2.00*2.00)
  {
         return 1;
  }
      }
     }
   }
  }

  return 0;
}

/* clash checking side-chain vs side-chain */
int fragment_clash(LABEL *label, MOLECULE *molecule1, MOLECULE *molecule2)
{

  int i, j, k;
  float dist2;

  for (i=0; i < molecule1->total.atoms; i++)
  {
/* ags 11/97 simplify heavy atom check for speed */
/*    if(label->vdw.member[molecule1->atom[i].vdw_id].heavy_id &&  */
   if(( (int)molecule1->atom[i].type[0] != 72)&&
     ( (int)molecule1->atom[i].type[0] != 68))
   {
    for(j = 0; j < molecule2->total.atoms; j ++)
    {
     if(( (int)molecule2->atom[j].type[0] != 72)&&
       ( (int)molecule2->atom[j].type[0] != 68))
      {
       dist2 = square_distance (molecule1->coord[i],
        molecule2->coord[j]);

       if((dist2 < 2.00*2.00)&&
    ((i != molecule1->anchor[0].atom[1])||
     (j != molecule2->anchor[0].atom[1])))
  {
         return 1;
  }
      }
     }
   }
  }

  return 0;
}


/* ////////////////////////////////////////////////////////////

non-recursive routine to rotate all atoms not part of the bond vector
in a fragment ***replaces todd's rotate_bonded_atoms*** for non-flexible
fragments
11/97 ags

//////////////////////////////////////////////////////////// */

int rotate_fragment
(
  XYZ rotation[3],
  XYZ origin,
  int current_atom,
  int previous_atom,
  MOLECULE *molecule
)
{
  int i;
  void transform_atom_ (XYZ, XYZ *, XYZ);

  for (i = 0; i < molecule->total.atoms; i++)
  {
   if((i != current_atom)&&(i != previous_atom))
   {
    transform_atom_
    (
     molecule->coord[i],
rotation,
origin
     );
   }
  }
  return TRUE;
}
```

rank.c

```
/*
Written by Todd Ewing
10/95
*/
/*
Modified by Geoff Skillman
*/
#include "definitions.h"
#include "utility.h"
#include "molecule.h"
#include "global.h"
#include "grid_struc.h"
#include "list.h"
#include "dock.h"
#include "label.h"
#include "grid_score.h"
#include "score.h"
#include "match.h"
#include "io.h"
#include "io_ligand.h"
#include "io_prepare.h"
#include "orient.h"
#include "rank.h"


/* ////////////////////////////////////////////////////////// */

int write_info
(
  DOCK *dock,
  SCORE *score,
  LIST *list,
  int ligand_read_num,
  int ligand_dock_num,
  int ligand_skip_num,
  float time
)
{
  int i, j, k;
  FILE *infofile = NULL;

  infofile = rfopen (dock->info_file_name, "w", global.outfile);

  fprintf (infofile, "%-35s : %10d\n", "Libraries read", ligand_read_num);
  fprintf (infofile, "%-35s : %10d\n", "Libraries docked", ligand_dock_num);
  fprintf (infofile, "%-35s : %10.2e\n", "Total Compounds", dock->combi.total_compounds);
  fprintf (infofile, "%-35s : %10.2e\n", "Total Conformations", dock->combi.total_conformations);
  fprintf (infofile, "%-35s : %10.2f\n", "Elapsed CPU time (sec)", time);
  fprintf (infofile, "%-35s : %10.2f\n", "Time per library (sec)",
   time / (float) ligand_dock_num);
  fprintf (infofile, "%-35s : %10.2f\n", "Time per compound (msec)",
   1000*time / dock->combi.total_compounds);
  fprintf (infofile, "%-35s : %10.2f\n", "Time per conformation (microsec)",
   1000*1000*time / dock->combi.total_conformations);

  if ((dock->rank_ligands)||(dock->combi.probe_flag))
   for (i = 0; i < SCORE_TOTAL; i++)
   {
    if (score->type[i].flag)
    {
     fprintf (infofile,
       "\nCurrent best %s scorers: \n", score->type[i].name);

     for (j = 0; j < list->total[i]; j++)
{
       fprintf (infofile, "%2d: %7.2f %1d %s ",
        j + 1,
        list->member[i][j]->score.total,
  list->member[i][j]->score.bumpcount,
        list->member[i][j]->info.name);
  for(k = 0; k < list->member[i][j]->total_sites; k++)
  {
   fprintf (infofile, "%s", list->member[i][j]->fragment_single[k].info.name);
  }
 fprintf (infofile, "\n");
}
    }
   }

 fclose (infofile);
}

/* ////////////////////////////////////////////////////////// */

int write_probe_info
(
  DOCK *dock,
  SCORE *score,
  LIST *list,
  char *scaffold_name,
  float time
)
{
  static int count=0;
  STRING80 line;
  int i, j;
  FILE *infofile = NULL;

/*  count++;
  sprintf (line, "%s.%d", dock->info_file_name,count);

  infofile = rfopen (line, "w", global.outfile);
*/
  fprintf (global.outfile,
     "\n**************************Library**************************\n");
  fprintf (global.outfile, "\n%-11s : %s\n", "Name",  scaffold_name);
  fprintf (global.outfile,
     "\n____________________Probe_Docking_Results____________________\n");

  if (dock->orient_ligand)
  {
    fprintf (global.outfile, "%-50s : %10d\n", "Orientations tried",
     score->pre_bump);

    if (score->flag.bump)
     fprintf (global.outfile, "%-50s : %10d\n", "Orientations scored",
      score->post_bump);
  }

  fprintf (global.outfile, "%-50s : %10.2f\n", "Elapsed cpu time (sec)", time);
  fprintf (global.outfile, "\n");

  if ((dock->rank_ligands)||(dock->combi.probe_flag))
   for (i = 0; i < SCORE_TOTAL; i++)
   {
    if (score->type[i].flag)
    {

     sprintf (line, "Best %s score", score->type[i].name);
     fprintf (global.outfile, "%-50s : %10.2f\n", line,
      list->member[i][0]->score.total);

     if (dock->orient_ligand || score->type[i].minimize)
     {
      sprintf (line, "RMSD of best %s scorer (A)",
       score->type[i].name);
      fprintf (global.outfile, "%-50s : %10.2f\n", line,
       list->member[i][0]->move.rmsd);
     }

     fprintf (global.outfile,
      "\nCurrent best %s scorers: \n", score->type[i].name);

     for (j = 0; j < list->total[i]; j++)
      fprintf (global.outfile, "%2d: %7.2f %1d %s %s\n",
       j + 1,
       list->member[i][j]->score.total,
  list->member[i][j]->score.bumpcount,
       list->member[i][j]->info.name,
       (list->member[i][j]->move.reflected ? "(REFLECTED)" : ""));
    }
   }
  fprintf (global.outfile, "\n");

/*  fclose (infofile); */
 }

/* ////////////////////////////////////////////////////////// */

int write_scaffold_info
(
  DOCK *dock,
  SCORE *score,
  LIST *list,
  MOLECULE_COMPOSE *library,
  float time
)
{
  static int count=0;
  STRING80 line;
  int i, j;
  float size=1, csize=1;
  FILE *infofile = NULL;

/*  count++;
  sprintf (line, "%s.%d", dock->info_file_name,count);

  infofile = rfopen (line, "a", global.outfile);
*/
  fprintf (global.outfile,
     "\n____________________Combi_Docking_Results____________________\n");
  fprintf (global.outfile, "%-35s : %10d\n", "Attachment sites", library->site_total);
  for(i = 0; i < library->site_total; i++)
  {
   sprintf (line, "Fragments at site %d", i);
   fprintf (global.outfile, "%-35s : %10d\n", line, library->unique_fragment_total[i]);
   sprintf (line, "Conformations at site %d", i);
   fprintf (global.outfile, "%-35s : %10d\n", line, library->fragment_total[i]);
   size *= library->unique_fragment_total[i];
   csize *= library->fragment_total[i];
  }
 dock->combi.total_compounds += size;
 dock->combi.total_conformations += csize;
 fprintf (global.outfile, "\n%-35s : %10.2e\n", "Virtual Size", size);
 fprintf (global.outfile, "%-35s : %10.2e\n", "Virtual Conformations", csize);
 for (i = 0; i < SCORE_TOTAL; i++)
 {
  if (score->type[i].flag)
  {
   fprintf (global.outfile, "\nCompounds on %6s list           : %10d\n",
  score->type[i].name, list->total[i]);
   fprintf (global.outfile, "Best %6s score                  : %10.2f\n",
  score->type[i].name, list->member[i][0]->score.total);
  }
 }
 fprintf (global.outfile, "%-35s : %10.2f\n", "Elapsed CPU time (sec)", time);

 if ((dock->rank_ligands)||(dock->combi.probe_flag))
  for (i = 0; i < SCORE_TOTAL; i++)
  {
   if (score->type[i] flag)
   {
    fprintf (global.outfile,
      "\nBest %s scorers: \n", score->type[i].name);
```

```
      for (j = 0; j < list->total[i]; j++)
        fprintf (global.outfile, "%2d: %7.2f %1d %s %s\n",
          j + 1,
          list->member[i][j]->score.total,
    list->member[i][j]->score.bumpcount,
          list->member[i][j]->info.name,
          (list->member[i][j]->move.reflected ? "(REFLECTED)" : ""));
     }
    }

/*  fclose (infofile); */
}

/* ////////////////////////////////////////////////////////////// */

int write_topscorers
(
  DOCK *dock,
  MATCH *match,
  SCORE *score,
  LABEL *label,
  LIST *list,
  MOLECULE *molecule
)
{
  int i, j, k, temp;
  int use_matrix = FALSE;
  long file_position;
  FILE *file;

/*
* ys
*/
  if (dock->combi.flag)
  {
    dock->ligand_file = dock->combi.scaffold_file;
    strcpy (dock->ligand_file_name, dock->combi.scaffold_file_name);
  }
/*
* Save the current position in the ligand input file
* 7/95 te
*/
  if (!dock->combi.flag)
    file_position = ftell (dock->ligand_file);

/*
* Loop through all requested scoring types
* 7/95 te
*/
  for (i = 0; i < SCORE_TOTAL; i++)
  {
    if (!score->type[i].flag)
      continue;

/*
*   Open up the output file
*   7/95 te
*/
    if (dock->rank_ligands)
      score->type[i].file =
        rfopen (score->type[i].file_name, "w", global.outfile);

    if (dock->rank_ligands && dock->combi.flag && !score->combi.merge)
      for (j = 0; j < dock->combi.max_sites; j++)
        score->combi.type[i].file[j] =
          rfopen (score->combi.type[i].file_name[j], "w", global.outfile);

    for (j = 0; j < list->total[i]; j++)
    {
/*
*    If coordinates are needed for output (output file is NOT ptr format),
*    then reread them and transform them
*    6/95 te
*/
      if (strcmp (strrchr (score->type[i].file_name, '.'), ".ptr"))
      {
        reset_molecule (molecule);
        copy_member (molecule, list->member[i][j]);

        if (dock->multiple_ligands || dock->combi.flag)
        {
          read_molecule
            (molecule, dock->ligand_file_name, dock->ligand_file);

    temp = ( strlen(list->member[i][j]->info.name) <
     strlen(molecule->info.name) ?
     strlen(list->member[i][j]->info.name) :
     strlen(molecule->info.name) );
          if (strncmp
            (list->member[i][j]->info.name, molecule->info.name, temp) != 0)
          {
            fprintf (global.outfile,
              "* * * Read incorrect ligand from input file:\n");
            fprintf (global.outfile, "      Intended : %s\n",
              list->member[i][j]->info.name);
            fprintf (global.outfile, "      Actual   : %s\n",
              molecule->info.name);
            exit (EXIT_FAILURE);
          }

          copy_info (molecule, list->member[i][j]);

          atom_neighbors (molecule);

          if (dock->combi.flag)
          {
            for (k = 0; k < molecule->total_sites; k++)
            {
              copy_member
                (&molecule->fragment_single[k],
                &list->member[i][j]->fragment_single[k]);

              read_molecule
                (&molecule->fragment_single[k], dock->combi.file_name[k],
                dock->combi.file[k]);

/* ags 11/97 need it to agree with change made in dock.c */
     vstrcat (&molecule->fragment_single[k].info.name, " ");

              if (strcmp
                (list->member[i][j]->fragment_single[k].info.name,
                molecule->fragment_single[k].info.name) != 0)
              {
                fprintf (global.outfile,
                  "* * * Read incorrect fragment from input file:\n");
                fprintf (global.outfile, "      Intended : %s\n",
                  list->member[i][j]->fragment_single[k].info.name);
                fprintf (global.outfile, "      Actual   : %s\n",
                  molecule->fragment_single[k].info.name);
                exit (EXIT_FAILURE);
              }

              if (molecule->fragment_single[k].move.anchored)
                molecule->move.anchored = TRUE;

              atom_neighbors (&molecule->fragment_single[k]);
            }
          }
        }

        if (molecule->move.moved || molecule->move.anchored)
          transform_molecule (dock, molecule, NULL, NULL);

        write_ligand
        (
          dock,
          score,
          label,
          molecule,
          score->type[i].file_name,
          score->type[i].file,
          ++(score->type[i].number_written)
        );
      }

      else
        write_ligand
        (
          dock,
          score,
          label,
          list->member[i][j],
          score->type[i].file_name,
          score->type[i].file,
          ++(score->type[i].number_written)
        );
    }

    if (dock->rank_ligands)
      fclose (score->type[i].file);

    if (dock->rank_ligands && dock->combi.flag && !score->combi.merge)
      for (j = 0; j < dock->combi.max_sites; j++)
        fclose (score->combi.type[i].file[j]);
  }

  if (!dock->combi.flag)
    fseek (dock->ligand_file, file_position, SEEK_SET);
}

/* ////////////////////////////////////////////////////////////// */

int write_restartinfo
(
  DOCK *dock,
  SCORE *score,
  LIST *list,
  int ligand_read_num,
  int ligand_dock_num,
  int ligand_skip_num,
  float clock_elapsed
)
{
  int i, j;
  long file_position;
  FILE *restart_file;

  void efwrite (void *, size_t, size_t, FILE *);

  file_position = ftell (dock->ligand_file);

  restart_file = efopen (dock->restart_file_name, "w", global.outfile);

  efwrite (&file_position, sizeof (long), 1, restart_file);
  efwrite (&ligand_read_num, sizeof (int), 1, restart_file);
  efwrite (&ligand_dock_num, sizeof (int), 1, restart_file);
  efwrite (&ligand_skip_num, sizeof (int), 1, restart_file);
  efwrite (&clock_elapsed, sizeof (float), 1, restart_file);

  save_list (list, restart_file);

  fclose (restart_file);
  return TRUE;
}
```

```
/* ///////////////////////////////////////////////////////// */

int read_restartinfo
(
  DOCK *dock,
  SCORE *score,
  LIST *list,
  int *ligand_read_num,
  int *ligand_dock_num,
  int *ligand_skip_num,
  float *clock_elapsed
)
{
  int i, j;
  long file_position;
  FILE *restart_file;

  void efread (void *, size_t, size_t, FILE *);

  restart_file = efopen (dock->restart_file_name, "r", global.outfile);

  efread (&file_position, sizeof (long), 1, restart_file);
  efread (ligand_read_num, sizeof (int), 1, restart_file);
  efread (ligand_dock_num, sizeof (int), 1, restart_file);
  efread (ligand_skip_num, sizeof (int), 1, restart_file);
  efread (clock_elapsed, sizeof (float), 1, restart_file);

  if (!load_list (list, restart_file))
  {
    fprintf (global.outfile, "* * * Error reading restart information \n");
    exit (EXIT_FAILURE);
  }

  fclose (restart_file);
  fseek (dock->ligand_file, file_position, SEEK_SET);
  return TRUE;
}
```

score.c

```
/*
Written by Todd Ewing
10/95
*/
/*
Modified by Yax Sun and Geoff Skillman
*/
#include "definitions.h"
#include "utility.h"
#include "molecule.h"
#include "global.h"
#include "grid_struc.h"
#include "list.h"
#include "dock.h"
#include "label.h"
#include "grid_score.h"
#include "score.h"
#include "match.h"
#include "fragment.h"
#include "io_ligand.h"
#include "orient.h"

int hash(char *key,  unsigned int size);

typedef struct simplex_struct
{
  DOCK *dock;
  SCORE *score,
  LABEL *label;
  MOLECULE *molecule,
  LIST *list;
} SIMPLEX;


/* /////////////////////////////////////////////////////////

Subroutine to calculate and optimize the score for a ligand orientation.
The best scoring orientation is updated.
7/95 te

///////////////////////////////////////////////////////// */

void evaluate_score
(
  DOCK *dock,
  SCORE *score,
  LABEL *label,
  MOLECULE *molecule,
  LIST *list,
  int  combi_flag
)
{
  int i, j, k;

  void minimize_ligand (DOCK *, SCORE *, LABEL *, MOLECULE *, LIST *);

/*
* Evaluate whether orientation bumps with receptor
* 3/96 te
*/
  score->pre_bump++;

  if (!combi_flag)
  {
    if (score->flag.bump &&
      (check_bump (&score->grid, &score->option, label, molecule) >
        score->option.bump_maximum))
      return;
  }

  score->post_bump++;

/*
* Loop through all scoring types
* 2/96 te
*/
  for (i = 0; i < SCORE_TOTAL; i++)
  {
    if (!score->type[i].flag)
      continue;

    molecule->score.type = i;

/*
*  Either minimize this orientation or just score it
*  2/96 te
*/
    if (score->type[i].minimize)
      minimize_ligand (dock, score, label, molecule, list);

    else
      get_ligand_score (dock, score, label, molecule, list);

/*
*  If the molecule has moved, then calculate the rmsd of current orientation
*/
    if (molecule->move.moved)
    {
      calc_rmsd (&label->vdw, molecule);

      if (dock->combi.flag)
        for (j = 0; j < dock->combi.molecule_compose->site_total; j++)
          for (k = 0, k < dock->combi.molecule_compose->fragment_total[j]; k++)
            calc_rmsd (&label->vdw, &dock->combi.molecule_compose->fragment[j][k]);
    }

/*
*  Write out the orientation to a file, if:
*
*  1. Ligands not ranked, AND
*  2. Want multiple orientations, AND
*  3. Orientations not ranked, AND
*  4. Other
*    A. Ligand has moved AND rmsd is within override, OR
*    B. No scoring is performed, OR
*    C. The score is below maximum cutoff
*
*  3/96 te
*/
    if
    (
      !dock->rank_ligands &&
      dock->multiple_orients &&
      !dock->rank_orients &&
      (
        ((molecule->move.moved) &&
          (molecule->move.rmsd <= score->rmsd_override)) ||
        !score->flag.any ||
        (molecule->score.total <= score->type[i].maximum)
      )
    )
    {
      write_ligand
      (
        dock,
        score,
        label,
        molecule,
        score->type[i].file_name,
        score->type[i].file,
        ++(score->type[i].number_written)
      );
    }
  }
}


/* ////////////////////////////////////////////////////////////// */

float calc_rmsd (LABEL_VDW *label_vdw, MOLECULE *molecule)
{
  int i, j, heavy_count = 0;

  molecule->move.rmsd = 0.0;

  for (i = 0; i < molecule->total.atoms; i++)
    if (label_vdw->member[molecule->atom[i].vdw_id].heavy_id)
    {
      for (j = 0; j < 3; j++)
        molecule->move.rmsd +=
          SQR (molecule->coord[i][j] - molecule->initial_coord[i][j]);

      heavy_count++;
    }

  molecule->move.rmsd /= (float) heavy_count;
  return molecule->move.rmsd = sqrt (molecule->move.rmsd);
}

/* ////////////////////////////////////////////////////////////// */

float calc_2rmsd (LABEL_VDW *label_vdw, MOLECULE *molecule1,MOLECULE *molecule2)
{
  int i, j, heavy_count = 0;
  float rmsd=0;
```

```
 for (i = 0; i < molecule1->total.atoms; i++)
  if (label_vdw->member[molecule1->atom[i].vdw_id].heavy_id)
  {
   for (j = 0; j < 3; j++)
    rmsd +=
     SQR (molecule1->coord[i][j] - molecule2->coord[i][j]);

   heavy_count++;
  }

 rmsd /= (float) heavy_count;
 return rmsd = sqrt (rmsd);
}

/* ////////////////////////////////////////////////////////// */

void minimize_ligand
(
 DOCK *dock,
 SCORE *score,
 LABEL *label,
 MOLECULE *molecule,
 LIST *list
)
{
 int i, size = 6;
 float *vertex = NULL;
 SIMPLEX simplex;

 float simplex_optimize (void *, float *, int, float, int *, int,
  float (*) (SIMPLEX *, float *));
 float simplex_score (SIMPLEX *, float *);

 if (score->minimize_torsion)
  size += molecule->total.flexes;

 ecalloc
 (
  (void **) &vertex,
  size,
  sizeof (float),
  "simplex array",
  global.outfile
 );

 simplex.dock = dock;
 simplex.score = score;
 simplex.label = label;
 simplex.molecule = molecule;
 simplex.list = list;

 molecule->score.total =
  simplex_optimize
  (
   (void *) &simplex,
   vertex,
   size,
   score->type[molecule->score.type].convergence,
   &molecule->score.iterations,
   score->iteration_max,
   simplex_score
  );

 simplex_score (&simplex, vertex);

 efree ((void **) &vertex);
}

/* ////////////////////////////////////////////////////////// */

float simplex_score (SIMPLEX *simplex, float *vertex)
{
 int i;

/*
* Convert simplex array to correct units
* 10/95 te
*/
 for (i = 0; i < 3; i++)
  simplex->molecule->move.final[i] =
   simplex->molecule->move.orient[i] +
   vertex[i] * simplex->score->initial_translation;

 for (i = 3; i < 6; i++)
  simplex->molecule->move.final[i] =
   simplex->molecule->move.orient[i] +
   vertex[i] * simplex->score->initial_rotation * PI / 180.0;

 if (simplex->score->minimize_torsion)
  for (i = 0; i < simplex->molecule->total.flexes; i++)
   simplex->molecule->flex[i].angle =
    vertex[i] * simplex->score->initial_torsion * PI / 180.0;

/*
* Transform the ligand to the new position
* 10/95 te
*/
 transform_molecule (simplex->dock, simplex->molecule, NULL, NULL);

/*
* Evaluate the score of this ligand position
* 10/95 te
*/
 return get_ligand_score
 (
  simplex->dock,
  simplex->score,
  simplex->label,
  simplex->molecule,
  simplex->list
 );
}

/* ////////////////////////////////////////////////////////// */

float get_single_ligand_score
(
 DOCK *dock,
 SCORE *score,
 LABEL *label,
 MOLECULE *molecule,
 LIST *list
)
{
/*
* Evaluate the score of the ligand in the current orientation
* 8/95 te 2/99/ags modified for plain-old single ligand
*/
 get_score
 (
  &score->grid,
  &score->option,
  label,
  molecule,
  molecule->score.type
 );

 molecule->score.total = molecule->score.sub_total;

 return molecule->score.total;
}


/* ////////////////////////////////////////////////////////// */

float get_ligand_score
(
 DOCK *dock,
 SCORE *score,
 LABEL *label,
 MOLECULE *molecule,
 LIST *list
)
{
 int i, j, k, m, number_held;
 int clash = FALSE;
 int previous_save, previous_member;
 int *index = NULL;
 int *index_base = NULL;
 int *index_dmsn = NULL;
 int index_max, index_update;
 float *fragment_score_maximum = NULL, ligand_score_maximum;
 STRING80 line;
 STRING200 long_name;
 float rotate_angle;

 XYZ rotation[3];

 XYZ     bond_vector;
 static LIST     *fragment_scored = NULL;
 static LIST*fragment_tmp_scored = NULL;
 FLEX     flex_anchor = {0};
/* MOLECULE fragment_tmp = {0}; */
 MOLECULE *fragment_tmp = NULL;
 MOLECULE fragment_tmp_best = {0};

 extern void get_index(int, int, int *, int *);
 extern void rotateaxis(float phi,float vect[3],float rot[3][3]);
 extern int  rotate_bonded_atoms (XYZ *, XYZ, int, int, MOLECULE *);

/*
* Evaluate the score of the ligand in the current orientation
* 8/95 te
*/
 get_score
 (
  &score->grid,
  &score->option,
  label,
  molecule,
  molecule->score.type
 );

/*
* ys
* 2/97
* RMSD calculation is done here because infomation will be store here in
* the list
*/
 if (molecule->move.moved && dock->combi.flag)
 {
  calc_rmsd (&label->vdw, molecule);

  if (dock->combi.flag)
   for (j = 0; j < dock->combi.molecule_compose->site_total; j++)
    for (k = 0; k < dock->combi.molecule_compose->fragment_total[j]; k++)
     calc_rmsd (&label->vdw, &dock->combi.molecule_compose->fragment[j][k]);
 }
/*
* ys
* 10/96
*/

/*
* ys
```

```
* 2/97
* DEBUG!!! for cathapsin D project only: RMSD override
*/
/*printf("%.2f\n",molecule->score.sub_total);*/
  if (dock->combi.flag &&
    (molecule->score.sub_total < BAD_SCORE) &&
    (molecule->move.rmsd < score->rmsd_max) )                          /*DEBUG*/
  {
/*
* now allocate the list to hold all the scored fragments
*/
/* ags 11/97 only malloc the first time through */
/* ags 12/97 add tmp list allocation */
   if(fragment_scored == NULL)
   {
    ecalloc
    (
     (void **) &fragment_scored,
     dock->combi.molecule_compose->site_total,
     sizeof (LIST),
     "scored_fragment_list",
     global.outfile
    );

    ecalloc
    (
     (void *) &fragment_tmp_scored,
     1,
     sizeof (LIST),
     "scored_tmp_fragment",
     global.outfile
    );
   }

   for (i = 0; i < SCORE_TOTAL; i ++)
   {
    for (j = 0; j < dock->combi.molecule_compose->site_total; j ++)
     fragment_scored[j].max[i] = (score->type[i].flag) ?
      dock->combi.max_fragments[j] : 0;

    fragment_tmp_scored->max[i] = (score->type[i].flag) ?
score->combi.greedy : 0;
   }

/* ags 11/97 only malloc the first time through otherwise nulify list*/
/* ags 12/97 add tmp_scored list */
   if(fragment_scored[0].member[SCORE_TMP] == NULL)
   {
    for (i = 0; i < dock->combi.molecule_compose->site_total; i ++)
     allocate_list (&fragment_scored[i]);

    allocate_list (fragment_tmp_scored);
   }
   else
   {
    for (i = 0; i < dock->combi.molecule_compose->site_total; i++)
reset_list (&fragment_scored[i]);

    reset_list (fragment_tmp_scored);
/*
    for (i = 0; i < dock->combi.molecule_compose->site_total; i++)
    {
fragment_scored[i].total[SCORE_TMP] = 0;
for(j = 0; j < fragment_scored[i].max[SCORE_TMP]; j++)
{
 fragment_scored[i].member[SCORE_TMP][j]->score.total = BAD_SCORE;
}
    }
    fragment_tmp_scored->total[SCORE_TMP] = 0;
    for(k=0;k < score->combi.greedy;k++)
{
 fragment_tmp_scored->member[SCORE_TMP][k]->score.total = BAD_SCORE;
}
*/
   }

/*
*  Update the orientations of all fragments to current scaffold's
*/
   get_matrix_from_angles_(rotation, &molecule->move.final[3]);

   for (i = 0; i < dock->combi.molecule_compose->site_total; i++)
    for (j = 0; j < dock->combi.molecule_compose->fragment_total[i]; j ++)
    {
     dock->combi.molecule_compose->fragment[i][j].move.moved = TRUE;

     transform_
     (
      &dock->combi.molecule_compose->fragment[i][j].total.atoms,
      dock->combi.molecule_compose->fragment[i][j].initial_coord,
      molecule->move.com,
      rotation,
      molecule->move.final,
      dock->combi.molecule_compose->fragment[i][j].coord
     );
    }

/*
*  Evaluate the score of all fragments in the current orientation
*/
   for (i = 0; i < dock->combi.molecule_compose->site_total; i++)
   {
    fragment_tmp_best.score.total = BAD_SCORE;
    for (j = 0; j < dock->combi.molecule_compose->fragment_total[i]; j++)
    {
/* ags 5/98 if frag + scaff weigh too much, just skip */
/*       if(dock->combi.molecule_compose->fragment[i][j].score.amw > 120){
   continue;
```

```
     }
*/
     flex_anchor.target = dock->combi.molecule_compose->
      fragment[i][j].anchor[0].atom[1];
     flex_anchor.origin = dock->combi.molecule_compose->
      fragment[i][j].anchor[0].atom[0];

     for (m = 0; m < 3; m ++)
      bond_vector[m] =
       dock->combi.molecule_compose->fragment[i][j]
        .coord[flex_anchor.target][m] -
       dock->combi.molecule_compose->fragment[i][j]
        .coord[flex_anchor.origin][m];

/* ags 11/97 don't copy just use fragment_tmp as a pointer to the fragment

     copy_molecule (&fragment_tmp,
      &dock->combi.molecule_compose->fragment[i][j]);
*/
fragment_tmp = &dock->combi.molecule_compose->fragment[i][j];

     for (k = 0; k < dock->combi.max_anchor_torsions; k ++)
     {
      fragment_tmp->score.total = BAD_SCORE;

/* ags 11/97 increment angle rather than starting over to avoid copying molecule */
   flex_anchor.angle = k * 2 * PI / dock->combi.max_anchor_torsions;
      rotate_angle = (k == 0 ? 0 : (2 * PI / dock->combi.max_anchor_torsions));

      rotateaxis
      (
       rotate_angle,
       bond_vector,
       rotation
      );

/* 11/97 ags don't need recursive function..rotate whole fragment for speed */
/*        for (m = 0; m < fragment_tmp->total.atoms; m ++)
       fragment_tmp->atom[m].flag = 0;

      rotate_bonded_atoms
      (
       rotation,
       fragment_tmp->coord[flex_anchor.target],
       flex_anchor.target,
       flex_anchor.origin,
       fragment_tmp
      ),
*/
      rotate_fragment  /* replacement routine for rotate_bonded_atoms */
      (
       rotation,
       fragment_tmp->coord[flex_anchor.target],
       flex_anchor.target,
       flex_anchor.origin,
       fragment_tmp
      );

      if (score->flag.bump &&
       (check_bump (&score->grid, &score->option, label, fragment_tmp)
       > score->option.bump_maximum))
       continue;

      fragment_tmp->score.total =
       get_score
       (
        &score->grid,
        &score->option,
        label,
        fragment_tmp,
        molecule->score.type
       );

      if (fragment_tmp_best.score.total == BAD_SCORE ||
       (fragment_tmp->score.total < fragment_tmp_best.score.total))
      {
       copy_molecule (&fragment_tmp_best, fragment_tmp);
       append_flex (&fragment_tmp_best, &flex_anchor);
      }

     } /* end k -- torsion loop */

/* ags 11/97 only update if we have a good score of a NEW fragment */
/* 12/97 ags change fragment update to tmp score list */

if(fragment_tmp_best.score.total < BAD_SCORE)
{
 update_list(fragment_tmp_scored, SCORE_TMP, &fragment_tmp_best, TRUE);
}
fragment_tmp_best.score.total = BAD_SCORE;

     if ((j+1 == dock->combi.molecule_compose->fragment_total[i])||
  (fragment_tmp->info.commentkey !=
dock->combi.molecule_compose->fragment[i][j+1].info.commentkey))
{

 merge_list(&fragment_scored[i], fragment_tmp_scored, TRUE);

 fragment_tmp_scored->total[SCORE_TMP] = 0;
 for(k=0;k < score->combi.greedy;k++)
 {
  fragment_tmp_scored->member[SCORE_TMP][k]->score.total = BAD_SCORE;
 }
}

    } /* end j -- fragment loop */
   } /* end i -- site loop */
```

```
/*
*   Allocate space for the index arrays
*/
   ecalloc
   (
    (void **) &index,
    dock->combi molecule_compose->site_total,
    sizeof (int),
    "combinatorial index array",
    global outfile
   );
   ecalloc
   (
    (void **) &index_dmsn,
    dock->combi molecule_compose->site_total,
    sizeof (int),
    "index dimension array",
    global outfile
   );
   ecalloc
   (
    (void **) &index_base,
    dock->combi molecule_compose->site_total,
    sizeof (int),
    "index base array",
    global outfile
   );
   ecalloc
   (
    (void **) &fragment_score_maximum,
    dock->combi molecule_compose->site_total,
    sizeof (float),
    "site score cutoff array",
    global outfile
   );

   index_max = 1;
   for (i = 0; i < dock->combi molecule_compose->site_total; i++)
   {
    fragment_score_maximum[i] = BAD_SCORE;
    index_max *= fragment_scored[i] total[SCORE_TMP];
    index_dmsn[i] = fragment_scored[i].total[SCORE_TMP];
   }

   for (i = 0; i < dock->combi molecule_compose->site_total; i ++)
   {
    index_base[i] = 1;
    for (j = i; j >= 0; j --)
      index_base[i] *= index_dmsn[j];
   }

   copy_molecule (&score->combi current, molecule);
   molecule->score total = BAD_SCORE;

/* ags 11/97 clash check fragments vs scaffold */

   if(score->combi check_clash)
   {
    for (i = 0; i < dock->combi molecule_compose->site_total, i++)
     {
      for (j = 0; j < fragment_scored[i] total[SCORE_TMP]; j ++)
      {
  fragment_scored[i] member[SCORE_TMP][j]->score bumpcount =
   molecule_clash(label, &score->combi current, i,
fragment_scored[i] member[SCORE_TMP][j], 0);
      }
     }
   }
/*
*   Loop through all combinations of best scoring fragments.
*   10/96 ys
*/

   for (i = 0; i < index_max; i ++)
   {
    get_index (i, dock->combi molecule_compose->site_total,
      index_dmsn, index);  /* converts i->index, an index to the fragments */

    score->combi current score total = molecule->score sub_total;
/* ags 3/98 add "rule of 5" scoring */
    if(score->lipinski){
      score->combi current score amw = molecule->score amw;
  /*   score->combi current score clogp = molecule->score clogp; */
      score->combi current score hbd = molecule->score hbd;
      score->combi current score hba = molecule->score hba;
    }



/* ags 11/97 if any frag clashes with scaffold skip it */
/*  sums score at same time */
    clash = FALSE;
    for (j = dock->combi molecule_compose->site_total -1;j>=0;j--)
    {
     if ( fragment_scored[j] member[SCORE_TMP][index[j]]->score bumpcount)
{
/* if frag bumps, increment its index */
 clash = TRUE;
 if(j>0)
 {
  i = index_base[j-1]-1;
  for(k=j; k < dock->combi molecule_compose->site_total;k++)
  {
   i += (index[k]*index_base[k-1]);
  }
 }
 break;
}
     score->combi current.score total +=
```

```
        fragment_scored[j].member[SCORE_TMP][index[j]]->score.total;
/* ags 3/98 add "rule of 5" scoring */
if(score->lipinski){
/*       score->combi.current.score.clogp +=
        fragment_scored[j] member[SCORE_TMP][index[j]]->score clogp;
*** not clogp yet */
       score->combi.current.score amw +=
        fragment_scored[j] member[SCORE_TMP][index[j]]->score.amw;
       score->combi current score hbd +=
        fragment_scored[j] member[SCORE_TMP][index[j]]->score.hbd;
       score->combi.current.score hba +=
        fragment_scored[j].member[SCORE_TMP][index[j]]->score.hba;
}

    }

    if(clash)
     continue;

/*
 *check to see if this molecule passed "rule of 5"
 *ags 3/98
 */
    if(score->lipinski){
     if(score->combi.current.score.amw > score->amw_max)
      continue;

     if(score->combi.current.score.hbd > score->hbd_max)
      continue;

     if(score->combi current.score.hba > score->hba_max)
      continue;
/*
     if(score->combi.current.score.clogp > score->clogp_max)
      continue;   ***not clogp yet ***/
    } /* end lipinski */

/* load "best" score for pseudo-minimization */
    if(score->combi current score.total < molecule->score.total)
molecule->score.total = score->combi.current.score.total;
/*
*    Check to see if this is a score worth considering
*/

    if ((score->combi.current.score total <
         score->type[molecule->score.type].maximum ) &&
        (list->total[molecule->score.type] <
         list->max[molecule->score type]  ||
         score->combi.current score.total <
           list->member[molecule->score.type]
            [list->total[molecule->score.type] - 1]->score.total))
    {
/*
*     Update the current ligand with the current set of fragments
*/
     strcpy (long_name, molecule->info.name);

     score->combi current.total sites =
      dock->combi.molecule_compose->site_total;

     for (j = 0; j < dock->combi.molecule_compose->site_total; j++)
     {
      strcat (long_name,
        fragment_scored[j].member[SCORE_TMP][index[j]]->info.name);

      copy_member (&score->combi current.fragment_single[j],
        fragment_scored[j] member[SCORE_TMP][index[j]]);
     }

     score->combi.current.info.fragment_single_allocated = TRUE;

/*
*     Check to see if this molecule has already been saved
*/
/*
     for (j = 0, previous_save = FALSE;
      j < list->total[molecule->score type]; j++)
     {
      if (!strcmp
        (list->member[molecule->score.type][j]->info.comment,
         score->combi.current.info.comment))
       {
        previous_save = TRUE;
        previous_member = j;
        break;
       }
     }
*/
/*
* ags 11/97 use numerical comparison
*/
score->combi current info.commentkey = hash(long_name, 100070);
     for (j = 0, previous_save = FALSE;
      j < list->total[molecule->score.type]; j++)
     {
      if (list->member[molecule->score.type][j]->info.commentkey ==
         score->combi.current.info.commentkey)
       {
        previous_save = TRUE;
        previous_member = j;
        break;
       }
     }
/*
*     if it has, then update the specific member
*     otherwise,update list normally
*/
     if (previous_save)
```

```
      {
       if(score->combi current score total <
        list->member[molecule->score type][previous_member]->score total)
       {
        if (score->combi check_clash)
        {
          if (combi_clash(dock, score, label, &score->combi current,
fragment_scored, index))
          {
           for(j = 0; j < dock->combi molecule_compose->site_total; j++)
            if (score->combi current fragment_single[j] score bumpcount)
             fragment scored[j] member[SCORE_TMP][index[j]]
              ->score bumpcount = TRUE,
          }
          else
    {
          update_member
           (list, molecule->score type, previous_member,
            &score->combi current),
    }
        }
        else
        {
         update_member
          (list, molecule->score type, previous_member,
           &score->combi current);
        }
       }
      }

      else
      {
       if (score->combi check_clash)
       {
        if (combi clash(dock, score, label, &score->combi current,
fragment_scored, index))
         {
          for(j = 0; j < dock->combi molecule_compose->site total, j++)
           if (score->combi current fragment single[j] score bumpcount)
            fragment scored[j] member[SCORE_TMP][index[j]]
             ->score bumpcount = TRUE,
         }
        else
   {
          update list
           (list, molecule->score type, &score->combi current, 0),
   }
       }
       else
  {
        update list
         (list, molecule->score type, &score->combi current, 0);
       }
      }
     } /*end if good score */

    /* if score exceeds the maximum, stop checking fragments at the
      current site, and adjust index to the next site (dimension) */
    else
    {
     m = 0;
     index update = 0,
     while ( m < dock->combi molecule_compose->site_total - 1 )
     {
      if (index[m] != 0)
      {
       i += index_base[m] - (index[m]*(m > 0 ? index_base[m-1] 1)) -1;
       index update = 1,
       break,
      }

      else
       m ++;
     }

     if (index update == 0)
      break.
    }
   }

   efree ((void **) &index);
   efree ((void **) &index dmsn);
   efree ((void **) &index_base),

/* 11/97 ags don't free these are now static
   for (i = 0, i < dock->combi molecule_compose->site_total, i ++)
    free list (&fragment_scored[i]),
   efree ((void **) &fragment scored);
*/
/*   free molecule (&fragment_tmp),   */
   free molecule (&fragment_tmp best);

/*
*  Print out the progress
*/
   if (score->minimize)
   {
    sprintf (line, "Best %s score at iteration %d",
     score->type[molecule->score type] name, molecule->score iterations++);
    fprintf (global outfile, "%-50s : %10.2f\n", line, molecule->score total);
    fflush (global outfile);
   }
  }

  else
   molecule->score total = molecule->score sub_total;

  return molecule->score total;
```

```
}

/* //////////////////////////////////////////////////////////// */

void output_score_info
(
 DOCK *dock,
 SCORE *score,
 MOLECULE *molecule,
 LIST *list,
 int number_docked,
 float time
)
{
 int i;
 STRING80 line;

 if (global.output_volume == 't')
 {
  if (number_docked == 1)
  {
   fprintf (global.outfile, "%s ", "Results: ");
   fprintf (global.outfile, "%s ", "name");

   if (dock->orient_ligand)
    fprintf (global.outfile, "%s ", "matches");

   if (score->flag.bump)
    fprintf (global.outfile, "%s ", "orientations");

   for (i = 0; i < SCORE_TOTAL; i++)
    if (score->type[i].flag)
    {
     fprintf (global.outfile, "%s ", score->type[i].name);

     if (score->type[i] minimize)
      fprintf (global.outfile, "%s ", "iterations");

     if (dock->orient_ligand || score->type[i].minimize)
      fprintf (global.outfile, "%s ", "rmsd");
    }

   fprintf (global.outfile, "%s\n", "time");
  }

  fprintf (global.outfile, "%s ", "Docked ");
  fprintf (global.outfile, "%s ", molecule->info.name);

  if (dock->orient_ligand)
   fprintf (global.outfile, "%d ", score->pre_bump);

  if (score->flag.bump)
   fprintf (global.outfile, "%d ", score->post_bump);

  for (i = 0; i < SCORE_TOTAL; i++)
   if (score->type[i].flag)
   {
    fprintf (global.outfile, "%.2f ", list->member[i][0]->score.total);

    if (score->type[i].minimize)
     fprintf (global.outfile, "%d ", list->member[i][0]->score.iterations);

    if (dock->orient_ligand || score->type[i].minimize)
     fprintf (global.outfile, "%.2f ", list->member[i][0]->move.rmsd);
   }

  fprintf (global.outfile, "%.2f\n", time);
 }

 else
 {
  fprintf (global.outfile,
   "\n________________________Docking_Results________________________\n");

  fprintf (global.outfile, "%-11s : %s\n", "Name", molecule->info name);
  fprintf (global.outfile, "%-11s : %s\n", "Description", molecule->info.comment);

  if (dock->orient_ligand)
  {
   fprintf (global.outfile, "%-50s : %10d\n", "Orientations tried",
    score->pre_bump);

   if (score->flag.bump)
    fprintf (global.outfile, "%-50s : %10d\n", "Orientations scored",
     score->post_bump);
  }

  fprintf (global outfile, "\n");

  for (i = 0; i < SCORE_TOTAL; i++)
   if ((score->type[i].flag)&&(list->total[i]))
   {
    sprintf (line, "Best %s score", score->type[i] name);
    fprintf (global outfile, "%-50s : %10.2f\n", line,
     list->member[i][0]->score.total),

    if (score->type[i].minimize)
    {
     fprintf (global outfile, "%-50s : %10d\n", "Minimizer iterations",
      list->member[i][0]->score iterations);
    }

    if (dock->orient_ligand || score->type[i] minimize)
    {
     sprintf (line, "RMSD of best %s scorer (A)",
      score->type[i].name);
     fprintf (global.outfile, "%-50s : %10.2f\n", line,
      list->member[i][0]->move.rmsd);
```

```
      }

      if
      (
        !dock->rank_ligands &&
       dock->multiple_orients &&
        !dock->rank_orients
      )
        fprintf (global.outfile, "%-50s : %10d\n", "Orientations written",
          score->type[i].number_written);

      fprintf (global.outfile, "\n");
     }

   fprintf (global.outfile, "%-50s : %10.2f\n", "Elapsed cpu time (sec)", time);

   fprintf (global.outfile, "\n\n");
  }

  fflush (global.outfile);
}

/* /////////////////////////////////////////////////////////// */

int hash(char *Key,  unsigned int size)
{
   unsigned int hashvalue = 0;

   while(*Key != '\0')
hashvalue = hashvalue * 37 + *Key++;

   return hashvalue%size;
}
```