Title

Elucidating essential targets in pharmacologically relevant system models

Author

Anderson, Abraham,

Publication Date

2002

Elucidating Essential Targets in Pharmacologically Relevant System Models

by

Abraham Anderson

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Bioengineering

in the

GRADUATE DIVISIONS

of the

UNIVERSITY OF CALIFORNIA SAN FRANCISCO

and

UNIVERSITY OF CALIFORNIA BERKELEY

[redacted]

Date University Librarian

Degree Conferred: ....

Dedication

To my immediate and extended family who
have provided guidance and encouragement
many times and in a myriad of ways.

Acknowledgements

During my time in the joint bioengineering graduate group there have been many who helped me accomplish my goals. Below are just a few of those I am grateful to:

C. Anthony Hunt: For allowing me to freely explore my ideas and for providing insight into several aspects of the pharmaceutical industry. As my research advisor, he provided a comfortable and flexible research environment

Sarah Nelson: For providing guidance as my academic advisor and providing a unique perspective on my thesis work.

Marie Mayne: For helping me navigate the initial stages of my degree program and lining up funding sources for my stipend.

Deborah Harris: For having an answer for all of my administrative questions and for keeping me abreast of all the things I needed to do to progress in the program.

Robert Stroud: For allowing me the opportunity to gain practical experience in computational drug design.

Raj Bhatnager: For allowing me to participate in tissue engineering research, a field that continues to hold my interest.

Richard Ho: For valuable discussions and for providing a venue for practical experience in industrial biocomputing and bioinformatics. Together we developed several prototype informatics tools; a process that has positively influenced my career plans.

I would also like to thank the members of my dissertation committee and qualifying exam committee for helping to guide my investigations and reviewing my work. They are: C. Anthony Hunt, Patricia Babbitt, Adam Arkin, Maurice Cohen, Donna Hudson, John Canny, and Stan Glantz.

This work was partly supported by grant # LSI-10041 from the California Life Science Informatics Technology Program, the R.W. Johnson Pharmaceutical Research Institute (La Jolla, CA), and by grant # r25-gm56847 from the NIH National Institute of General Medical Sciences. The Hunt Lab in the Department of Biopharmaceutical Sciences is a dynamic and vibrant research environment and has contributed greatly to my academic and research progress.

Abstract

Elucidating Essential Targets in Pharmacologically Relevant System Models

By

Abraham A. Anderson

Large-scale and complex biological system models can be used to simulate and mimic normal and pathologic body function. Putative pharmacological agents can be "screened" in silico for greatest effect by adjusting model parameters at postulated and known drug target sites to match experimental data. Once the simulation is set in motion, the effect can be monitored and the drug's effectiveness evaluated. This document describes how these models can be used to find potentially optimal intervention sites via quantitative and qualitative graph theoretic techniques. A major benefit of this method is its computational objectivity in analyzing large and highly connected systems. It is also much faster than a complete, classical sensitivity analysis of the model. Several systems are analyzed for modular substructure and the elements of those systems are prioritized as potential therapeutic targets. These are blood coagulation, human obesity, and bacterial metabolism. Predictions are validated with information about known therapeutic targets, essential genes and metabolites. For quantitative models, a sensitivity analysis is done to further validate prioritizations.

As we emerge from the genomic era, during which whole genomes have been completely sequenced, data mining is being used to elucidate interactions between genes and proteins. The resulting relational models are expected to be quantitatively fleshed out, and used to adequately predict biological phenomena. Biological and

pharmacological system models are not new, but as a consequence of high throughput, robotic data generation technologies, they are necessarily rapidly expanding in size and getting more complex. A single researcher can no longer conceptualize even a fraction of the dynamical interactions within such models; much less judge the full effect each element may have on the system. Short of a tedious full-scale sensitivity analyses, guesses about which element or set of elements is likely to have a desired, widespread effect will be based on trial-and-error, and thus biased. The techniques employed hearken from computer network planning and parallel processor load balancing, and should return comprehensive and *utterly objective results*. These results may contain conceptual and computational artifacts and so were qualitatively evaluated and iteratively refined. Essentially, these techniques filter the system and present those elements that serve to ***hold the system together***, and/or prevent chaotic behavior. Such critical elements are logical targets for evaluation as potential therapeutic intervention loci. Thus, models once used to test hypotheses, can serve double-duty by generating hypotheses. This sort of computer aided target location coupled with current methods for evaluation, are expected to give investigators the added edge of objectivity married with speed.

Scientists and engineers require new techniques and tools for exploring, visualizing, and understanding the limits and relative properties of large complex biological systems. The approach described here may be such a technique.

Approved by: C. Anthony Hunt, Ph.D.

C Anthony Hunt 8/22/02

Research Advisor date

Contents

Elucidating Essential Targets in Pharmacologically Relevant System Models i
Dedication ii
Dedication iii
Acknowledgements iv
Abstract vi
Contents viii
List of Tables xi
List of Figures xiii
Chapter 1: Motivating Factors 1
Chapter 2: Background 7
Methods For Therapeutic Target Discovery 7
Defining Therapeutic Targets 7
Target Discovery and Prioritization 8
Chapter 3: Network-Based Applications & Analyses which may Provide Clues to Finding Potential Therapeutic targets 27
Sources of Relational Biological Data 27
Visualization, Analysis, Modeling 28
Benefiting from a Topological Perspective 29
Visualization and a Venue for Thought Experiments 29
Quick Analysis Method for Quantitative Models 30
Topological Elements & System-Wide Influences 32
Modularity 32
Metabolic Control Theory 33
Sensitivity Analysis with Signal Flow Graphs 36
Scale-Free Networks 37
Phenotype Phase Plane Analysis or Network Flow 38
Analyzing Biological Network Topology 39
Chapter 4: Motivating Hypotheses 41
Chapter 5: Methods 43
Graph Models of Biological Systems 43
Data Management 43
Connected Components & Strongly Connected Components 43
Articulation Points & Bridges 44
Centrality Measures 44
Closeness 44
Betweenness 45
Graph Clustering 45
Graph Partitioning 46
Balanced Partitioning 46
Unbalanced Partitioning 47
Finding an Optimal Partitioning 48
Chapter 6: Predictions of Therapeutic Targets and Variable Sensitivity via Prioritizations from Partitioning & Centrality Analyses 49
OVERVIEW 49

INTRODUCTION 49
RESULTS 51
Blood Coagulation 51
Obesity PhysioLab™ v1.1 55
DISCUSSION 56
METHODS 62
Summary Selection Method 62
Validation 62
Chapter 7: Effect of Increasing a Model's Quality & Size on Target Prioritizations 65
OVERVIEW 65
INTRODUCTION 66
RESULTS 70
DISCUSSION 77
SYSTEMS & METHODS 81
Chapter 8: Software Tools Developed For Topological Analyses 83
OVERVIEW 83
INTRODUCTION 83
PROGRAM DESCRIPTION 84
Performance 85
Chapter 9: Test Case on Public Data: *E. coli* Metabolism 89
OVERVIEW 89
INTRODUCTION 90
RESULTS 92
DISCUSSION 101
METHODS 106
Chapter 10: Conclusion 111
References 120
APPENDIX A: Tables 125
APPENDIX B: Website References 132
APPENDIX C: User Manual 133
Introduction 133
Requirements 134
X-files 134
Python™ files 134
METIS™ files 134
Setting up the system for initial use. 134
Setting up your input files for analysis. 134
Console Functions 135
Automated Analysis Tool 135
Individual Tools 136
Examples 138
GUI Interface 139
Displaying a graph 139
Choosing a layout for the graph 139
Selecting Nodes 140
Labels 141

Analysis Tools ... 141
Displaying analysis results ... 141
Examples ... 143
Index ... 144
Appendix D: Developer Manual ... 145
Codebase ... 146
Analysis Tools ... 146
CC.py ... 146
SCC.py ... 146
Centrality Calculations ... 147
METIS™ Interface ... 147
Spectral Graph Partitioning ... 147
Graph Clustering ... 148
The GUI ... 149
Layout ... 149
Data Visualization ... 150
Appendix E: Computer Executable Code ... 151

List of Tables

Table 2-1. Methods for mapping biomolecular interaction*. 9
Table 6-1. Entities of Blood Coagulation Model 53
Table 6-2. Blood Coagulation Disorders 53
Table 6-3. Significant Relations || Bridges & Spans 53
Table 7-1. Parametric experiments' results sorted by decreasing fraction of weight lost (||loss||) vs placebo. Experiments with significant (>5%) weight loss colored grey. Other experiments with significant (>5%) weight gain (||gain||) or significant (>threshold for each prediction method: 0.162 & 0.22) average-fat-mass sensitivity (Sm) colored grey. 76
Table 8-1. Time taken to perform all analyses for several systems, and metadata for each system. 85
Table 9-1. Articulation points from the global graph with their centralities: Directed Betweenness Centrality (DBC), Undirected Betweenness Centrality (UBC), Directed Closeness Centrality (DCC), and Undirected Closeness Centrality (UCC). The sort is based on decreasing DBC, where larger values are preferred. 30 are shown here, but the full listing is provided in Table A-8 (see Appendix A). 94
Table 9-2. Sixteen key metabolites for E. coli metabolism. A + indicates that the cooresponding meatabolite had a significant score from one or more prioritization metrics; a – indicates otherwise. An EC number indicates that the specific metabolite's score was not deemed significant, but the score of an enzyme linked to it was. Results are shown for analysis of the global and filtered graph. 96
Table 9-3. The key metabolites that had significant indicators from the global and filtered analysis are shown below with the significant scores for each metric. Each metric's significance threshold is indicated above the metric's name. Nodes that are articulation point s (APs) are indicated. UCC, UBC, & DBC are defined in Table 9-1. 96
Table 9-4. Distribution Statistics for Essential and Nonessential Enzymes. Values in parentheses are for the filtered graph others are for the global graph. D = maximum difference between essential and nonessential enzyme distributions. Q = probability of observing a difference greater than D. Ne = effective number of observations for the calculation of Q. S = shift of essential gene score distribution away from the nonessential gene score distribution. UCC, UBC, DCC & DBC are defined in Table 9-1. 97
Table A-5. The best scoring molecules for each metric when applied to the global graph of E. coli metabolism. The criteria for significant a closeness score is less than 1.5 standard deviations below the mean; and for betweenness, greater than 5 standard deviations above the mean. The betweenness centrality metrics are Directed Betweenness Centrality (DBC), and Undirected Betweenness Centrality (UBC). The closeness centrality metrics are Directed Closeness Centrality (DCC), and Undirected Closeness Centrality (UCC). 125
Table A-6. The best scoring molecules for each metric when applied to the filtered graph of E. coli metabolism. The criteria for significant a closeness score is less than 1.5 standard deviations below the mean; and for betweenness, greater than 5 standard deviations above the mean. The betweenness centrality metrics are Directed

Betweenness Centrality (DBC), and Undirected Betweenness Centrality (UBC). The closeness centrality metrics are Directed Closeness Centrality (DCC), and Undirected Closeness Centrality (UCC). 127
Table A-7. Articulation points for the filtered graph, listed with their centrality scores. The sort is based on decreasing UBC, where larger values are preferred. 128
Table A-8. Articulation points from the global graph with their centralities. The sort is based on decreasing DBC, where larger values are preferred. 130

List of Figures

Figure 3-1. Enzyme(circle) and metabolite(square) network. 34

Figure 3-2. Directed control graph. Numbered nodes are enzymes. 34

Figure 3-3. Disintegration modes due to attack. Figure redrawn from Albert et.al......... 37

Figure 6-1. Blood coagulation network shown with MDS layout. Larger nodes have better betweenness centrality when calculated for the undirected graph.................... 54

Figure 6-2. Observed Sensitivity (S) Fractions... 58

Figure 6-3. Observed Sensitivity (S) Fractions... 58

Figure 6-4. OB v1.1: Effect diagram of obesity model v1.1 displaying portions of analysis results. MDS = multidimensional scaling.. 60

Figure 7-1. Fraction of nodes in each SCC. Inset shows SCCs 96 to 160. 71

Figure 7-2. Finding an optimal number of partitions with METIS................................. 71

Figure 7-3. Fraction of known obesity drug targets predicted exactly and including graph neighbors of predicted targets, using the Standard method (a) and the Metric-based method (b)... 73

Figure 7-4. Increasing likelihood of observing a significant sensitivity (Sm) with a.) increasing betweenness and b.) decreasing closeness centrality, and c.) increasing modeled-metric. 74

Figure 8-1. Screenshot of GUI showing selection box, fisheye zoom, data representation, and graph layout... 87

Figure 8-2. Screenshot showing GUI, the status console, a partial view of the imported-data table, and two node selection boxes.. 88

Figure 9-1. Scale free degree distribution, with fitted power functions. 93

Figure 9-2. Distributions of centralities for essential and nonessential enzymes. The abscissas are oriented so that better centrality values are leftward. 'cumu e/et', 'cumu n/nt', and 'd' represent the cumulative fraction of essential and nonessential enzymes and the difference between the two distributions. 97

Figure 9-3. Finding an optimal partitioning. [s] denotes data from the spectral partitioner. [m] denotes data from the METIS partitioner... 99

Figure 9-4. Subgraph 23 from k = 24, containing reactions shared by arginine and proline metabolism; glycolysis / gluconeogenesis; and glycine, serine, and threonine metabolism. Larger nodes have higher directed betweenness centrality. Arrows indicate articulation points.. 100

Figure 9-5. Partition 5 from k = 24. Larger nodes have higher directed betweenness centrality. Components represent subpathways common to 2 or more pathways. A) urea cycle and alanine and aspartate metabolism. B) fructose and mannose metabolism, pyruvate metabolism, and glycerolipid metabolism. C) galactose metabolism, and starch and sucrose metabolism... 102

Figure 9-6. The steps involved in identifying critical objects and modularity in a network (graph) using said network's topology. Raw data on biological interaction or dependence is modeled by a graph or network. Next a variety of techniques are applied on the search for modular substructures. If found, the objects involved in linking separate modules are selected and prioritized according to several measures of criticality in a network. Above, AP stands for articulation point, and SCC stands for strongly connected component. In case modules are not found satisfactorily, all objects in the network are prioritized, and the highest priority nodes are hypothesized to be critical for information or mass flow the system being modeled. To the right: steps are labeled; the potential applicability of each phase's results are indicated; and the entire process labeled as Comparative Topology, as different systems can be objectively compared by their modularity and critical objects.................................107

Chapter 1: Motivating Factors

The ideas that prompted this investigation stem from graph partitioning techniques used in workload distribution for parallel processing. With these techniques, the goal is to divide work evenly among several processors while reducing interprocessor communication. Several techniques are being utilized to partition these networks and work is still ongoing. In general, graph partitioning is being used to solve optimization problems in the design of very large-scale integrated circuits, storing and accessing spatial databases on disks, transportation management, and data mining[2]. A direct method of partitioning a graph uses the Laplacian of the adjacency matrix. It is known that the Laplacian provides a minimization solution the total length of connections between adjacent nodes. The solution induces clustering of connected nodes by partitioning the underlying graph[3]. Seary and Richards present a survey of some early attempts to understand network structure from the Standard spectral representation and introduces the Laplacian spectrum as a method for directly obtaining useful results[4]. They also introduce the *Normal spectrum* and show that it has similar properties with the Laplacian. They present examples that suggest that the Normal spectrum is useful for role-based and group-based network analyses. They state, "the signs and magnitudes of eigenvalues indicate the type and importance of the partitioning induced by the corresponding eigenvectors." There are many tools available to partition graphs. The METIS toolkit and function library will be used in this project[5]. METIS uses a fast, multilevel, recursive, partitioning algorithm.

If biological system representations are similarly treated, elements could be clustered into groups that have minimal intercluster communications and reactions. *I* hypothesize that those elements that communicate between system groups uniquely effect proper system functioning. If validly represented, these critical elements should be ideal intervention sites within the system.

An interesting use of clustering and partitioning techniques has emerged in quantitative anthropology. Doreian demonstrated complementary qualitative and quantitative tools in delineating the structure of social networks[6, 7]. He used the ideas of centrality and multidimensional scaling (MDS) of graph distances to describe graph structure and delineate key network members. An adjacency matrix representing social relationships was developed and used to generate another matrix of graph distances. This distance matrix was the input to a MDS algorithm. The output produced was a Euclidean projection in two dimensions. The approach resulted in a network partition from which important qualitative statements could be made. In a systematic way they were able to determine those nodes that *integrate* the graph. Additionally, nodes that provided the least amount of integration had the smallest centrality scores.

Social Network Analysis (SNA) is the study of the interactions between multiple actors for the purpose of discerning repeated patterns of interaction and the role of specific actors in the overall group. As SNA is a quantitative analysis method developed in the field of social and behavioral sciences, actors are usually individuals, countries, or organizations. Thus interactions are interpersonal, international, or interorganizational. For analyses on the level of an actor, several metrics have been developed to quantify an actor's importance, prestige, or influence in the group. One set of metrics is collectively described as centrality metrics. Over the years several researchers have contributed their own quantification of the centrality concept. Degree centrality is the simplest definition of centrality. It prioritizes actors according to the number of their direct interactions with other actors in the network. Closeness centrality quantifies how close an actor is to all other actors in the network via tabulation of internode distances. Betweenness centrality prioritizes actors higher if they lie between many other actors and on the shortest routes between pairs of these actors. Information centrality generalizes the concept of betweenness centrality by considering paths other than the shortest and scales their contribution according to each path's length. Some of these metrics have been converted for analysis of groups as well as individual actors, thus quantifying the levels of influence actor's have as a group on other groups or, when the group contains all actors, group centrality can be used to evaluate the significance of an individual actor's centrality. There are other group level analyses in the field of SNA that aim to provide quantitative measures for deciding which actors constitute a group. Such methods measure subgroup cohesion via statistical methods, considerations of reachability within the group, and matrix manipulations. Once cohesive subgroups are identified, researchers go on to evaluate differences between members of groups and nonmembers, and overlap between subgroups. Also, different networks can be compared by their individual organization into subgroups.

System models in biology take on many forms. Some are explicitly mathematical whereas others are purely relational with informative diagrams representing various levels of abstraction. Knowledge maps and effector diagrams both use arcs and nodes to qualitatively represent the interaction between system elements. These elements can be single proteins or molecules and even abstract concepts. The Ecocyc database is a good example of a knowledge map built on genomic and proteomic elements from studies of

E. coli[8]. The final product is a searchable database of reaction pathway data. This framework has also been extended to overlay gene and protein expression data in an intuitive manner. More quantitative models can be seen in the various PhysioLabs made by Entelos[9]. Each PhysioLab models a human physiological system of special therapeutic interest to the pharmaceutical community. The models represent a homeostatic state upon which clinical trials can be simulated. The elements within the model are both conceptual and explicit, as necessary or as supported by data. They are related by simple linear and non-linear differential equations that represent diffusion, conversion, and element interaction. The complexity and strength of these models has evolved from the large number of interconnected elements. So far PhysioLabs can only be used to simulate different disease states and test hypotheses. Any sensitivity analysis must be done manually, one element at a time.

There are an increasing number of such modeling efforts. It is not possible to review them here. As an example, however, a smaller scale version of such system models is under careful development by Levine et al. to model onset of capillary formation initiating angiogenesis, and to relate that to cancer growth[10].

Chapter two introduces techniques for studying biological systems and discovering potential therapeutic targets and/or biomarkers.

Chapter three begins with an overview of current uses of network representations in biology. Several network based systems analysis techniques are presented. Aspects common to each of these techniques and to other non-biological network analysis techniques are highlighted.

Chapter four presents hypotheses concerning the topology of biological systems and how it may be used in the prediction of novel therapeutic targets.

Chapter five presents methods developed to test the hypotheses presented in chapter three. In this chapter a heuristic incorporating these methods is described.

Chapter six, seven, and nine present applications of the methods to several complicated biological systems. In each application individual methods are validated against known therapeutic targets, *in silico* experimentation, and pathway annotations.

Chapter six presents the results of an application of the individual methods to a qualitative model of blood coagulation and a quantitative model of human obesity. During which, each method is evaluated and presented as part of a general heuristic for modularity assessment and target prediction and prioritization.

Chapter seven presents the results of an application of the heuristic to an improved version of the obesity model. A primary issue here is the effect of changes in a model on the quality of the analysis results, and there are many additions and changes in the newer model of obesity.

Chapter eight contains a description of the software tool kit developed to implement the heuristic. The tool kit analyzes network topology and presents the results in a data-rich graphical fashion.

Chapter nine presents the results of an application of the heuristic to the type of network model most accessible for researchers. The metabolism of *E. coli* is modeled from a public database and data quality issues are dealt with; results are validated in light of prior knowledge of essential genes and important metabolites.

Chapter ten summarizes the hypotheses that motivated the development of the topological methods, the computational techniques used to implement each method, and the results of various implementations. This chapter ends with suggestions for improvement of the techniques and their place in therapeutic target discovery efforts.

Chapter 2: Background

Methods For Therapeutic Target Discovery

Defining Therapeutic Targets

Therapeutic targets encompass any biological molecular process(es) that maximally influence(s) a specified physiological objective function, or that changes the phenotype of the subject or patient. The process of target validation and selection identifies those potential targets that have the most influence on the physiological process to be modulated. Many targets of successful drugs are G-protein coupled receptors (GPCRs). These are cell surface receptors, allowing cells to act based on environmental signals. One reason for the abundance of GPCR targets might be that they are relatively easy to access on cell surfaces. It is harder to design a drug to both access and then interact with highly sequestered target molecules.

For drugs, developed to influence a target, to be commercially successful, the biological problem that they potentially alleviate must be common or a chronic disease. Otherwise, the validation of a target and development of drugs may not be worth the investment. Also, the disease that the target is associated with should be well understood and have known surrogate markers for disease progression. This helps the proper identification of patients, leading to shorter clinical trials. Intervention at the target must offer an advantage over current therapies. Such advantages may be increased efficacy, convenient dosing, or fewer side effects. Having a well understood target or homologue to it, and inhibitors or agonists for comparison helps in the validation process. Many current drugs have similar targets and some targets are indicated for multiple diseases.

Initially, alternate indications make clinical trials more complicated, and federal approval harder.

Target Discovery and Prioritization

Biochemical Means

Many therapeutic targets are actually found indirectly, by using anecdotal knowledge from previous pharmacological studies. For example, in the multigenic disorder obesity, many of the early drugs used to control obesity were already on market as approved CNS drugs[11]. Prior to 1995 pharmaceutical R&D companies developed drugs for targets of which 70% had a similar class of drug already on the market[12, 13]. There are many ways to discover the initial set of genes and proteins that are involved in a disease state. Some of the most general approaches are based on differential display observations. That is, the observation of a difference between two separate mRNA or protein expression profiles may indicate molecules that cause the phenotype difference between the two environments producing the expression profiles. Expression of mRNA can be observed with electrophoresis techniques and more recently with higher throughput silicon chip expression systems. Expression of proteins can also be measured with 1D or 2D electrophoresis and protein-chips are becoming available. Another approach to finding the initial set of phenotype-associated molecules is genetic linkage evaluation, which identifies genetic markers or single nucleotide polymorphisms (SNPs) that are statistically linked to a diseased population. Once such markers are found, the specific gene is identified by positional cloning; genes overlapping the marker are sequenced and checked for improved linkage. After doing so, one has a rough idea of which genes are involved in the phenotype. Knock-out or knock-in experiments can be undertaken and if the phenotype is a unigenic disorder, then one has the target. Since most disorders a multigenic in nature, selecting targets for pharmaceutical intervention is complicated.

A single protein may interact with many other molecules: proteins, DNA, RNA, small molecules. In order to fully appreciate the complexity of a multigenic disorder or, on a smaller scale, a signaling pathway, the system's molecular interactions must be mapped. Once mapped, a system's various routes of interaction can be analyzed for critical sites of vulnerability, which may suggest a strategy for a new pharmaceutical intervention. Later in this chapter *I* present several approaches to search networks for targets. First, I discuss some old and new techniques for discovering/mapping biomolecular interactions.

Table 2-1. Methods for mapping biomolecular interaction*.

	DNA	RNA	Protein
DNA	Southern Blot Expression-Correlation?	Northern Blot Complex Trap Footprinting Gel Shift	Complex Trap Footprinting Gel Shift
RNA		Northern Blot Y2H?	Complex Trap Gel Shift
Protein			Y2H Immuno Trap Western Blot

*This list is not a comprehensive experimental list and it should be noted that data-mining techniques are also used to extrapolate relations between molecules from database text[14, 15].

Expression Correlation: Statistical correlation between the expression of genes by transcription implies a undefined functional relationship between them[16]. Time-lagged correlations of expression time series data can imply the causal nature of relationship[17].

Footprinting: A strand of DNA that is believed to have a region to which another molecule binds is enzymatically digested. The products are analyzed electrophoretically. The molecule of interest is allowed to bind to the DNA prior to digestion. Differences in the electrophoretic pattern indicate the location of the binding site; the binding molecule protects the oligo from digestion within its *footprint*[18].

Gel Shift: The electrophoretic mobility of an oligonucleotide changes when it changes conformation or is bound to another molecule. This mobility change is indicated by a shift in the normal distance traveled within the gel.

[Southern, Northern, Western]-Blot: Several target molecules are run on a gel, after which a probe molecule is washed over the gel. Targets can be DNA, RNA, or protein. For convenience the molecules are transferred to a sheet of nitrocellulose prior to probing. Unbound probes are washed away and the bound probes are located by radio or photo labels. For example, a Western blot experiment is done to detect specific proteins. First proteins are separated by polyacrylamide-gel electrophoresis, then transferred to a sheet of nitrocellulose and incubated with an antibody designed to bind a specific protein. Unbound antibodies are removed, and any bound antibodies are detected by coupled fluorescent dye. Probes can alternatively be labeled with a radioactive isotope or detectable enzyme.

Immuno Trap: There are many ways of using antibodies in assays for molecular interaction. For example, an antibody, linked to a surface and also bound to molecule B, has a solution of molecules washed over it. Any molecules in the solution that can bind to B are pulled out of solution, or if B is an enzyme, a detectable reaction occurs (ELISA[19]).

Y2H: Yeast-2-Hybridization indicates protein-protein interactions by transcriptional rescue. A transcription activation factor is divided into its two main domains, a DNA binding domain and a transcription activation domain. To see if proteins A and B bind together, A is fused to the DNA-binding domain and B is fused to the activation domain. If A and B do not bind there is no transcription of the marker gene[20].

Complex Trap: The complex between two or more molecules is trapped by the addition of formaldehyde. Antibodies for a trapped protein are used to isolate it and anything bound to it. A similar technique ChIP coupled with DNA microarrays has been proposed for genome-wide detection of transcription factors[21].

The methods just described allow initial delineation of the complicated molecular interactions involved in a disease system. Once the network of interactions is discovered, the network can be analyzed for regions susceptible to attack. These critical regions may explain aberrant phenotypes and/or suggest intervention points for future therapeutic development.

Computational Means

Qualitative Approaches

Genetic Network Inference

High-throughput DNA expression array data has been used to unravel functional relationships between genes. Though they imply little about actual molecular mechanisms, they may reveal genes critical for a biological process. Several researchers have developed Bayesian networks to describe the causal relationships behind gene expression, which is key for genetic manipulation[16, 22]. The Bayes networks do not contain cycles as a result of their definition, but the description of biological cycles and feedback pathways is essential for description or modeling of oscillation and metabolic control. A reformulation of the Bayes network is the Dynamic Bayes Network[23]. This models feedback, but without directed edges, and requires a lot of data to construct. In a different approach called Correlation Metric Construction, Arkin et. al. use time-lagged correlation of metabolite concentration time series to uncover the underlying metabolic pathway[17]. The pathways discovered do contain cycles unlike Bayesian methods, but this method may not scale well to systems with many metabolite concentrations to measure. Even with the scaling issue it is a good alternative to isolating and measuring detailed kinetics for large systems. D'haeseleer gives a good review of genetic network inference methods including Boolean network representations[24].

Petri Nets

A simulation alternative to systems of differential equations, SANs, Stochastic Activity Networks, or Petri Nets have recently been employed in data mining efforts as a platform for functional predictions and interpretation of genomic expression data. Since Petri Nets have been in use for quite a while in engineering endeavors, they have a mature suite of software for their management and construction. Küffner et. al. have used Petri Nets to enumerate all valid metabolic pathways and they suggest comparisons between cell types, disease states, and whole organisms in a technique called DMD or Differential Metabolic Display[25]. After compiling the metabolic pathway information from several public databases, they reveal 500K paths of length less than 9 that accomplish the conversion of glucose to pyruvate. The Petri Net representation of a system's pathways can be used to identify gaps in pathways, interpret expression data, and predict protein function. Once a system is represented by a Petri Net simulations can be run to model the effects of changes to the system. Liebman and Mounts performed a sensitivity analysis of the blood coagulation system and recreated hemophilic states[26].

Scale-Free Networks

A network's pattern of connections is very important for larger-scale effects such as resistance to attack or random error and communication delay. Most theoretical study of network properties assumes the probability of a connection between two nodes follows a gaussian distribution. In reality, many networks have a degree distribution that follows a power law, where degree is a count of a node's immediate neighbors. This topological scaling property is common to many metabolic networks, like organisms in the WIT database, and non-biological networks like the Internet[27]. These scale-free networks have only a few highly connected nodes, and any 2 nodes can be connected by a short path, regardless of the network's size. The scale-free nature makes networks robust to random errors such as genetic errors, but they have decreased attack survivability, which is good for drug design[1]. In Barbasi's study, network attack was the informed deletion of nodes with many links to the rest of the network, but it is unclear whether ranking therapeutic targets by nodal degree is an effective screen for targets in a large and possibly incomplete network.

Modular Organization in Biology

The various biological control mechanisms that ensure signal fidelity, homeostasis and signal amplification likely take on a modular system of organization. A module in biology is a set of molecules that interact together to perform a specific function[28, 29]. The transition from input to output only depends on the molecules within the module. This isolation stems from chemical specificity and physical sequestering, as within organelles. A module's function is to reliably perform a biological task and evolutionary pressures have certainly optimized this. Higher-level functions can be built by incorporating and connecting modules together. Phenotypes may change by altering interaction between individual modules, while keeping a module's core function robust. The limited connectivity between modules, avoids widespread effects to evolutionary changes. In other words, it's easier to reconnect modules than high connectivity proteins like histones, which do not change very often.

Fidelity and redundancy in all biological processes is required for an organism to maintain successful reproduction and robustness to environmental changes. Drug design based on targets within modules are frequently frustrated by the system's redundancy[29]. Restricting the target pool to links between modules would reduce the target validation space and focus on those elements that, by definition, enable complicated phenotypes. To date there are no techniques aimed at finding this sort of modular design in biological systems, but with the strong efforts in functional proteomics[30, 31] and recasting and analysis of biological information as networks[32], someone is bound to take up the task of module discovery and exploitation.

Quantitative Approaches

Saturable kinetics is at the core of much pharmacokinetic and pharmacodynamic modeling. Michaelis-Menten kinetics was developed to understand reaction rate changes due to changes in substrate concentration, prior to substrate depletion and product accumulation. Another inherent assumption is that the turnover rates are much smaller than the enzyme-substrate dissociation rates. In cases where this last assumption does not hold, Briggs-Haldane kinetics is used. Here the steady state assumption—that the intermediate concentrations do not change much relative to the enzyme, substrate, and product concentrations—is used to derive the rate/substrate relationship. The steady-state assumptions that make these systems of equations easier to solve have also been part of the reasoning about what makes a good target. A good therapeutic target might be a molecule that has the most influence on the rate of a pathway. At steady state the overall rate of a pathway is assumed to be no faster than the slowest step in the pathway[33]. This assumption led many in a search for the 'rate-limiting-step' as a possible therapeutic target or in chemical manufacturing, a way to increase production. Unfortunately, after much work in finding possible rate-limiting steps, improvement of the rate here did little to improve the pathway's overall flux[34]. In pathways of unsaturated enzymes, flux depends nonlinearly on all kinetic parameters[35]. Also, at steady state, all reactions are assumed to be at the same rate, so there is no slowest rate[33]. In order to better understand and ultimately make predictions about system behavior, additional complicated phenomena should be considered in modeling: multiple enzymes and substrates, molecular trafficking and signaling, enzyme internalization and presentation, adaptation, sorting and degradation. Many accurate models have been developed to describe the complicated behavior of biological systems, such as transcription and substrate cycling[36, 37]. It should be noted that an infinite number of accurate models can be developed to describe the same system's behavior, but there is no guarantee that accurate predictions can be made about real world changes to the system. The system's topology is sometimes much more important than the kinetic parameters in understanding complicated phenomena[38].

There are various methods in developing mathematical models, but basically one lists the system interactions and transfer-processes, obtain parameters for systems of usually differential equations, iteratively solve the system and refine parameters. Manually obtaining solutions is usually desired, but for slight increases in parameters of complexity, only numerical solutions are possible. In linear systems where the function to be integrated is a sum of constant coefficient linear equations, the solution is sum of solutions for the homogenous and particular equations. The homogeneous solution is typically solved by the separation of variables technique or Eigen methods. The particular solution is solved by substitution. In nonlinear systems, where the integrating function is a sum of nonlinear functions, superposition of solutions does not apply and solutions are obtained by substitution. As opposed to manual methods, numerical integration is most often done for complicated systems. A common iterative method is Runge-Cutta, where each new value of the integrated function is a function of the current value and a several slopes of the function within a specified step-size. For systems of partial differential equations, the manual methods of solution involve the assumption the function that partial derivative are being taken from is a product of several new functions—one for each partial derivative. The transformed equation can now be solved by separation of variables and independent solutions for each new function. Numerical methods usually involve mesh representation of the function space and mesh interpolation with boundary conditions. Each cell in the mesh is iteratively updated with a new value of the function. A cell's new value is roughly the average of its neighbors' current values plus a cell-specific function.

Metabolic Control Theory, MCT

The idea of 'a single rate-limiting step that controls a pathway' needed some 'modification.' Instead of a rate-limiting step, in MCT, all molecules in the system control pathway flux. The flux control coefficient, Eq 2-1, is defined as the sensitivity of the pathway flux, J, to the rate of the reaction catalyzed by an enzyme, E. It is therefore not an enzyme property, but depends on the metabolic state. Changes to the system cause a redistribution of flux control. Elasticity, Eq 2-2, is defined as the sensitivity of the rate of a reaction catalyzed by an enzyme to changes

$$C_{xase}^{Jydh} = \frac{\partial \ln J_{ydh}}{\partial \ln E_{xase}} \approx \frac{\partial J_{ydh}}{J_{ydh}} \Big/ \frac{\partial E_{xase}}{E_{xase}} \qquad (2\text{-}1)$$

$$\varepsilon_S^{xase} = \frac{\partial \ln |\upsilon_{xase}|}{\partial \ln S} \qquad (2\text{-}2)$$

$$R_P^{J_{ydh}} = \frac{\partial \ln J_{ydh}}{\partial \ln P} = C_{xase}^{J_{ydh}} \varepsilon_P^{xase} \qquad (2\text{-}3)$$

$$\sum_{i=1}^{n} C_i^J = 1 \qquad (2\text{-}4)$$

in substrate concentrations. The response coefficient is a generalized sensitivity function, Eq 2-3. There are two main theorems in MCT, the summation and connectivity theorem. Since all enzymes, i, share flux control, the sum of flux control coefficients is unity, Eq 2-4. Due to the net balance of a metabolites control, the sum of flux control coefficients for enzymes affected by the metabolite, S, and the appropriate elasticity terms equal zero, Eq 2-5. The multi-site response of any parameter, P, can be formulated in the same fashion, Eq 2-6, but the sum is not constrained to zero.

$$\sum_{i=1}^{n} C_i^J \varepsilon_S^i = 0 \qquad (2\text{-}5)$$

$$R_P^J = \sum_{i=1}^{n} C_i^J \varepsilon_P^i \qquad (2\text{-}6)$$

To use MCT to describe control of a pathway, elasticity terms can be measured or derived from rate equations. Flux-control coefficients are either measured or algebraically solved for. When working with large systems it is useful to note that the summation and connectivity theorems form a solvable system of equations, but the measurement of a large amount of elasticities can be daunting. In general MCT is a good basis for analyzing metabolic systems. Acerenza showed that large changes in flux can be designed not by selecting the single enzyme with largest flux-control, but a group of enzymes that share most of the flux-control and increasing their individual fluxes together[39].

Electrical Circuit Analogues

As parameter measurement is inhibitory for large and complicated systems, Sen described an electrical analogue for MCT problems[40]. The main use of this alternate representation is in determining the relative importance of enzymes and simplification of pathway structure, which eases the burden of later parameter measurement. In the electrical circuit analogue, current resistances are connected in series or parallel with a voltage or current source. Enzymes are associated with resistances, $R \sim \varepsilon$, and flux-control coefficients are associated with currents, determined by Ohm's Law (V=I*R). The current-analogue circuit is setup with all resistances in parallel to a voltage source of unity. The alternate voltage-analogue circuit is setup with conductances in series, and the relative flux control is equivalent to a specific resistor's voltage fraction/drop. These circuits provide a visual framework for studying the regulatory behavior of metabolic pathways. Sen notes that more than one current-analogue or voltage-analogue circuit can represent the control structure of a given pathway.

Graph Theoretical Approach

Sen has also investigated the regulatory properties of metabolic pathways by using a graph-theoretical approach[41]. Like the electric circuit analogues, this graph theoretic method is used to study the relative importance of enzymes that control a pathway's flux, and studying the cause and effect relationships among them. The control structure of the pathway is represented by a weighted directed graph, constructed without writing summation and connectivity theorems explicitly[42]. Elasticities are edge weights and control coefficients are formed as tree products—

also called 'path gains'. Control coefficients can be calculated symbolically or numerically from the elasticities. The flux control coefficient for any enzyme is N/D. D = Sum of the loop gain products for each 1-connection. A 1-connection is a subgraph that includes all nodes plus edges that form a cycle involving any node. N = Sum of path gains to a specific enzyme's node. The gain of a path is the product of edge weights along the path.

King & Altman Method

Circa 1956, King and Altman popularized another graph theoretical method that used a directed graph for the analysis of steady-state enzyme kinetics[43]. A node represents each enzyme state and the transition rate between each state becomes the weight for a directed edge between their nodes. The fractional concentration of each state is defined as the sum of all spanning tree's (a subgraph that touches all nodes and without cycles) tree products that contain a particular enzyme state, all divided by a denominator—which is the sum of numerators for each enzyme state. The fractional concentrations can be inserted into any rate equations, providing an alternative to direct solution of the rate equation.

Mason Graphs

Mason graphs are a special kind of signal flow graph. Signal flow graphs can be used to graphically represent any system of equations and subsequently study sensitivities or solve optimization problems. A Mason graph is a directed graph where the nodes are system's variables, and the edges are weighted by the partial derivative of the target with respect to the source node. Paths through the graph are used to calculate transmittances, T, or transfer functions between distant nodes. Once a transfer function (or transmittance) is determined, its derivative can be taken with respect to any system parameter. This is defined as the sensitivity, S, of the output to that parameter. I have not been able to find any direct applications of this theory to metabolic control or pathway analysis, but the graph methods defined by AK. Sen are the closest match. Even so, there are many papers on Mason theory for engineering optimization problems and several software packages to handle the graph manipulations[44]. The significant difference in engineering systems and biological ones is that an engineering system is usually planned and all parameters are known to a high degree, parameters for biological systems are unknown—it's the researcher's task to discover them. Biological parameters are not always known and the amount of parameter measurement required for large systems makes the direct application of engineering control methods difficult.

Stoichiometric Network Analysis

The steady-state internal metabolite stoichiometry matrix has been used in many ways to analyze the pathway structure of metabolic networks. Simpson et al. describe why enumeration of all independent pathways is important to the study of bioreaction networks[38]. Firstly, it identifies the degrees of freedom in a network that uniquely determine all pathway fluxes. Secondly, each reaction carries the flux corresponding to one or more of the independent pathways. The relative importance of each individual reaction can be assessed. Thirdly, if fluxes of reactions in a independent pathway leading to a desired product are known, it is possible to devise a strategy to amplify the flux to an arbitrary level. This point is based on work by Acerenza; see [39]. Lastly, the independent pathways can be used to identify critical branch points where metabolic pathways converge or diverge. An independent pathway is defined by Simpson as the smallest set of reactions connecting a single network output with the necessary network inputs in a way that permits the levels of internal species to reach a steady state[45]. The stoichiometric matrix is n x m where n is the number of explicit steady state metabolites and m is the number of explicit reactions. Each element of this matrix is the stoichiometric coefficient of metabolite n in reaction m. Simpson describes the independent pathways as the Eigen vectors of the stoichiometric matrix. Contemporary work done by Palsson uses the basis vectors of the stoichiometric matrix to define 'extreme pathways' which are similar to Simpson's 'independent pathways' except they are linearly independent[46]. The set of pathways are not linearly independent if the size of this set is greater than the dimension of the null space. Therefore, Palsson subjectively chooses a subset to arrive at the 'extreme pathways'. Another technique to identify characteristic pathways structure was developed by Schuster et al. and goes by the name 'elementary flux modes.'[47] They are rationalized as the minimal set of enzymes for the system that operates at steady state. As opposed to 'extreme pathways'—a basis for the flux space—they always form an objective and unique path set. They can be used to define any flux, compare similar systems, and find non-redundant pathways. Convex analysis is used to calculate these pathways, by restricting the flux vectors to be positive in orientation and finding the vector that span this convex null space. In general all the previously mentioned techniques can be used in a similar manner, but 'elementary flux modes' best lends itself to automation and scaling to larger systems. As for finding drug targets, Schuster et al. suggest that given a well-annotated genome, like *M. tuberculosis*, the metabolic routes remaining after a knockout can be calculated and in this manner find vulnerable sites in the network. The techniques for these and other pathway calculation are contained in a software package called METATOOL[48].

Sensitivity Analysis

This is general concept that is used to determine model quality, accuracy, influential factors, and study factor interaction. Sensitivity analysis is generally done for each output variable. The model is executed for combinations of values sampled from an input variable distribution. The number of calculations involved can be quite large, so qualitative screening is done to reduce the number of test inputs prior to the actual quantitative sensitivity analysis. If the model is small enough—this depends on your computational apparatus and personal ability—the sensitivity to all parameters can be checked. The synergistic effects of multiple parameter variation should also be evaluated, but adds exponentially more calculations.

Sensitivity analysis can be used to gain insight into the operation of many systems or, on a more basic level, study the effects of uncertainty in the variables of a closed mathematical function. One of the more unique applications is in the construction of a Bayesian belief network[49]. Instead of directly constructing large belief networks, smaller ones can be constructed and improved. They are improved by finding critical locations in the network and creating a new network by replacing it with additional highly informed elements (additional elements that would be in a much larger network). Sampei et al. at Chiba University in Japan developed a fully qualitative method for sensitivity analysis[50]. Their method can be used on systems modeled in the rule based causal format. It is also used for performance evaluation of discrete event dynamic systems.

The need for a quick, objective heuristic for finding therapeutic targets

As I pass the genomic era, during which whole genomes have been completely sequenced, data mining is being used to postulate complicated interactions between genes and proteins. The relational models that develop are expected to be quantitatively fleshed-out and used to accurately predict or reproduce biological phenomena. Biological and pharmacological system models are not new, but as a consequence of high throughput, robotic data generation technologies, they are rapidly expanding in size and complexity. A single researcher can no longer conceptualize even a fraction of the dynamic interactions within such models, much less judge the full effect each element may have on the system. Short of a tedious full-scale sensitivity analyses, guesses about which element or set of elements is likely to have a desired, widespread effect will be based on trial-and-error, and thus biased. Currently there are no 'standard' methods for reasoning about the characteristics of complicated networks in biology. The separate methods of Schuster, Stephanopoulis, and Palsson are recent attempts to formalize the representation of biological networks from a pathway-based perspective, but they do not directly attempt to predict possible therapeutic targets. There are a few indications of how to find possible therapeutic targets and they mostly involve the network topology. Here are four examples.

After studying the blood coagulation pathway with Petri-Net models, Mounts and Liebman (1997) suggest that, "critical-path and bottleneck analysis will allow predictions of therapeutic intervention positions by focusing on the structure and connectivity of the pathway." In the study of characteristic pathways in ATP metabolism, Stephanopoulis (1999) concludes that, "When considering large sets of reactions, however, it becomes nearly impossible to reach salient conclusions based upon kinetics and feedback, without first considering the intrinsic structure of the reaction network." In one of the initial papers detailing the use of Correlation Metric Construction, for the reengineering of metabolic networks, Arkin (1997) states that, "The correlation matrix also gives a rough idea of how 'central' each species is to the dynamics of the network." He goes on to say that, "[This provides a way to] predict likely points of control." These three researchers, studying metabolic control, all reach the general conclusion that the potential for control in a pathway is heavily influenced by the topology of the network as a whole. In a different area of study, Bayesian network models of genetic influence, Friedman (2000) relates the network topology to macro-scale phenotype with the observation that, "The most striking feature of the high confidence order relations, is the existence of *dominant* [or high degree] genes. ... these genes are potential causal sources of the cell cycle process." All of these observations are just that, observations. A strategy that both evaluates large and highly connected biological networks and returns candidate critical system elements is expected to be valued for many reasons: it should shed light of the inherent biological organization in complicated metabolic networks or protein interaction networks; a topological strategy computationally cheap enough to allow for objective predictions of drug targets could serve as a hypothesis generation engine able to work in tandem with an *in silico* or *in vitro* validation engine. Optimizing therapeutic target selection and validation in this manner would eventually increase the quality of disease intervention and, with the ability to consider the entire network topology, facilitate prediction of multi-drug therapies. Such a method could initially be general in nature and applicable to the multitude of qualitative networks plus the small but growing number of large and complicated quantitative simulation networks. The actual methodology should be well informed by work in non-biological sciences such as computer science and possibly social science. In many areas of study graph partitioning and data clustering is a recurring task and computational scientists have developed a multitude of tools to do such work. A quantitative method used many times in social science research is Social Network Analysis. This is the study of a social network from a topologically based perspective with the aim of: shedding light on social network organization, and developing methods to measure an actor's influence over others or prestige in the group. These, and possibly other areas of study outside molecular biology, are expected to provide clues in developing a method to predict therapeutic targets based on network topology.

Chapter 3: Network-Based Applications & Analyses which may Provide Clues to Finding Potential Therapeutic targets

Sources of Relational Biological Data

Currently, the main use for online biological databases is as a primary data retrieval tool, functioning much like a dictionary. However, a number of online databases and their private sector counterparts[51-54] that store relational biological information have been used for more complicated data mining efforts. For example, a systematic statistical approach, based on structural and sequential databases, has been used to accelerate the pace of discovering new protein folds[55]. Another example is the use of archived scientific literature to extract intergenic relationships to aid gene expression analysis[14]. Yet, there remain large databases of relational biological data, like protein-protein or more general interactions (or reactions) between molecules, which have not been used at a level beyond simple data retrieval. These databases could additionally be used for predicting molecules with potential for significant system-wide metabolic control. Specifically, databases like the PathCalling Yeast Interaction Database contain significant relational data that represents an almost complete biological interaction network[56, 57]. This protein interaction network's topology could be used far beyond its current primary focus on single entity and local interaction queries if the database usage were extended to analyze organizational structure and identify features that indicate the system's control mechanisms. The data sources certainly exist and a broader use for them merits consideration.

Visualization, Analysis, Modeling

Some groups maintaining relational biological databases have realized the inadequacies of standard data visualization methods, and they have also recognized the benefits of network visualizations. The EcoCyc database developers, for example, have created an appealing representation of molecular reaction pathways and integrated access to the underlying relational data[58]. Visually 'appealing' presentations of data allow viewers to absorb the information more efficiently and intuit higher-level relationships implied by the data because the predominant way people interpret their environment takes place by sight. By contrast, sound, touch, smell, and taste present more difficult data mappings. Thus, the representation of biological information is increasingly being done via network visualizations such as with protein interaction maps for the yeast and *H. pylori* databases[59]. A few other groups are also developing automatic network layout algorithms that intended to optimally convey a network's topological structure in a familiar biological or chemical visual syntax[60]. On a macro level, these representations often provide enlightenment about the system's overall organization, but there is still confusion when discerning micro level features. Given a network of molecular interactions, one is still at a loss to select a single molecule (or group of molecules) as key to the selection of therapeutic targets or to understand the system's control structure.

Analyses of large biological networks are occasionally present in the scientific literature. Barabasi et. al., for example, analyze the vulnerability of biological networks with certain unique degree[i] distributions[1]. In a later article, they describe correlations between known lethal yeast gene deletions and a gene's degree[61]. While their work

[i] Degree is the count of a node's neighbors in a network.

focuses on a single molecule, others are working to gain a clearer understanding through pathway perspectives. Differential Metabolic Display (DMD) has been developed to take advantage of the network management, layout, analysis, and simulation tools currently used in stochastic modeling, specifically of Petri Nets[25]. DMD begins by storing enzyme reaction information in a Petri Net format and uses custom algorithms to extract all paths through the network between two metabolites. These paths can be compared with gene expression data to assess if they exist at all or whether an organism is using them. Additionally, it is possible to compare and contrast paths active in two different organisms. Though this differential metabolic display can assess a differentially active pathway or set of enzymes, it does not prioritize the results. And the results can be quite numerous. Yet another pathway-based perspective is being worked on contemporaneously by different groups[45-47]. The stoichiometric matrix of a reaction network is used to specify a unique set of pathways that can be combined additively to represent any metabolic state of the network. The results, however, require additional experiments or simulations to discern therapeutic targets with significant metabolic control. Again, as with DMD, a large network of relations is processed and a group of pathways are presented, but there is no accompanying prioritization theory or guide to selecting a therapeutic target.

Benefiting from a Topological Perspective

Visualization and a Venue for Thought Experiments

Understanding large amounts of relational data is greatly enhanced by network representations and visualizations. Currently, productive use of network-based interfaces to data are seen in the KEGG[62] enzyme reaction database, the GeneNet[63] database, and the ProNet[51] protein interaction databases. Presenting relational data in a visual manner is quite useful, but when coupled with a theory for discerning the important characteristics of a system and an accompanying graphical analysis methodology, data visualizations enables new understanding of a system.

The right data representation can facilitate useful thought experiments, which allow one to infer a system's responses to proposed interventions. After developing a graph theoretical approach for Metabolic Control Analysis, Sen demonstrated a method for conducting thought experiments using his network representations[42]. The metabolic system was first represented as a graph (Figure 3-2), where nodes represented enzymes and arcs linking nodes represented elasticity coefficients. One example of a thought experiment followed this basic pattern: represent the lack of product inhibition by nullifying an elasticity coefficient; graphically recalculate the flux control coefficients of the system's enzymes; finally, ascertain the significance of the shift in flux control. This type of thought experimentation would be useful in working with large disease system models where simulation time is prohibitive for flexible analysis. Some things are required before using this kind of thought experimentation on large models of disease: an appropriate network representation, and a metric for describing a molecule's potential for system wide influence.

Quick Analysis Method for Quantitative Models

Currently, quantitative system models are used to provide information to assist in the validation of a given drug target and drug mechanism. Pairing this strong validative capability with an automated target prediction engine would allow for computer guided target discovery. Definitely faster than human-guided target discovery, it comes with a few caveats. When a researcher goes through the process of computational target validation, s/he discerns whether the model is properly representing reality and most times the model is updated as a result. A fully automated hypothesis generation and validation engine would not have the benefit of model updates during the target discovery process, but should provide objectively selected drug targets in a shorter amount of time, at a lower cost.

Most quantitative biological system models are nonlinear systems of differential equations like PhysioLab™ from Entelos®[9] or stochastic systems like SANs or Petri Nets[64]. When the system is based on continuously differentiable equations, sensitivity analysis (SA) is normally used to find maximally influential parameters and improve a model's quality, and accuracy. SA is more often used when the system is small or computation time is unlimited, as the model executed for combinations of values sampled from an input distribution.

Decreasing the model's simulation time would make SA an available option for larger systems. If fast, qualitative SA (QSA) methods were developed concurrently, SA would be an analysis option for many more quantitative and qualitative models of molecular interaction. Far, Nakamichi, and Sampei presented QSA methods for application to systems modeled in rule based causal format[50]. Their QSA method involved following the propagation of a truth assignment through the discrete event dynamic system. Each logical variable was perturbed either singly or as sets and having two concurrent contradicting logical states identified other variables as sensitive to the perturbed set. The technique was completely based upon a directed graph and graph traversal heuristics.

Topological Elements & System-Wide Influences

Given the potential benefits of using the as yet untapped topological information in biological networks and system models to indicate potential therapeutic targets and the desire to expand the use of SA to large quantitative models and qualitative models in general, I will show that network topology can directly be used to highlight critical system parameters. Network analysis techniques from engineering disciplines and social network analysis can be used to highlight tight subsystems of molecular organization. The molecules that link these subsystems together are expected to serve as prime candidates for therapeutic target validation leading to later drug design. What follows are five indications that the processes that link metabolic subsystems are critical to the phenotype of their macrosystem. In technical terms, a network can be separated into subsystems with a minimal number of links between them. When found by network partitioning or network flow analysis, these links are called a 'cut-set.' The mathematical sensitivity for a specific disease indicator to network parameters is most likely maximal for those parameters in the cut set.

Modularity

The various biological control mechanisms that ensure signal fidelity, homeostasis and signal amplification likely have evolved to have a modular system of organization. A module in biology is a set of molecules that interact together to perform a specific function[28, 29]. The transition from input to output only depends on the molecules within the module. This isolation stems from chemical specificity and physical sequestering, as within organelles. A module's function is to reliably perform a unique biological task and evolutionary pressures have certainly optimized this.

Incorporating and connecting modules can build higher-level functions. Phenotypes may change by altering the connections between individual modules, while keeping a module's core function robust. The limited connectivity between modules, avoids widespread effects to evolutionary changes. In other words, it's easier to evolutionarily reconnect modules than high connectivity proteins like histones, which do not change very often. Fidelity and redundancy in all biological processes is required for an organism to maintain successful reproduction and robustness to environmental changes. Drug design based on targets within modules are frequently frustrated by the system's redundancy[28]. Restricting the target pool to links between modules would reduce the target validation space and focus on those elements that, by definition, enable complicated phenotypes.

Metabolic Control Theory

Analysis of metabolic control starts by defining sensitivities of a metabolic flux to the concentration of different enzymes (control coefficient) and the sensitivity of enzymes' catalytic rates to concentrations of metabolites (elasticity)[33]. Identifying enzymes with large flux control coefficients has been very powerful in the design of large metabolic responses with multi-site modulation[39]. The analysis of large systems with feedback is harder due to difficulties in experimental measurement of the control coefficients and elasticities. A. K. Sen presented a topological technique for metabolic control analysis that evaluates all coefficients symbolically[41]. The technique allows one to compare control coefficients for different enzymes. Effects of enzyme inhibition and saturation can be evaluated by modifying the network of interactions. If the output of a system is most sensitive to reactions that link different subsystems, then control

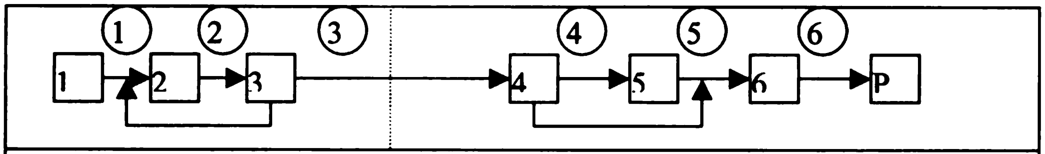


Figure 3-1 Enzyme(circle) and metabolite(square) network.

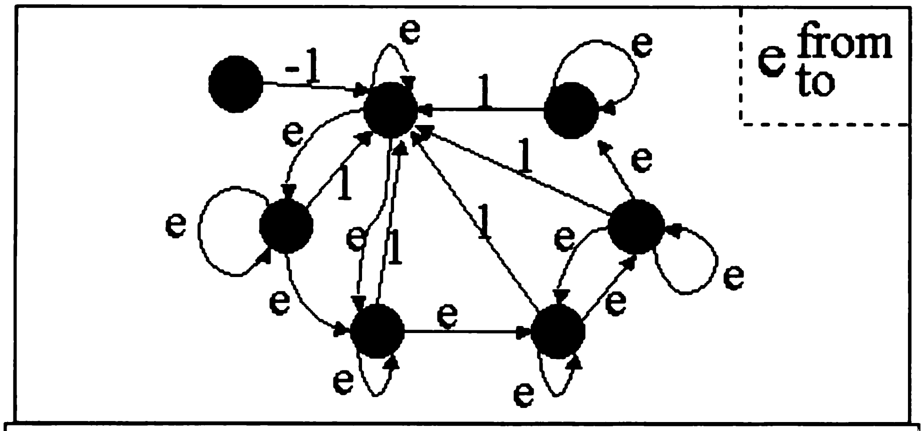


Figure 3-2 Directed control graph. Numbered nodes are enzymes.

coefficient(s) for the enzyme(s) participating in these linking reactions should be greater than for other enzymes in the system.

Sen's topological approach can be used to show that this is the case for a simple system where a module is characterized by a feedback loop and the reactions leading to the production of the feedback metabolite, and two such modules are linked by having a common metabolite. Under these circumstances the enzyme producing the common metabolite has the highest flux control coefficient, and would be considered in a set of potential drug targets for said system.

Figure 3-1 depicts a system with two separated control structures, a feedback reaction and a feed-forward reaction. The circles are enzymes and the boxes are metabolites. This system is first converted to a graph with edges weighted by elasticities, and a node for each enzyme plus a source node adjacent to enzyme 1. The e's on each edge, in figure 3-2, represent the elasticity of the target node on the source node. The flux control coefficient for any enzyme is N/D. D = Sum of the loop gain products for each 1-connection. A 1-connection is a sub-graph that includes all nodes plus edges that form a cycle involving any node—there are many 1-connections for this graph, one example is a graph with all nodes and all self loops (an edge leaving and returning to the same node). N = Sum of path gains to a specific enzyme's node. The gain of a path is the product of edge weights along the path. The flux control coefficient for enzyme 1 is shown here: $C_{1}^{J} = {}^{1\varepsilon_{2}^{2}\varepsilon_{3}^{3}\varepsilon_{4}^{4}\varepsilon_{5}^{5}\varepsilon_{6}^{6} - 1\varepsilon_{2}^{2}\varepsilon_{3}^{3}\varepsilon_{5}^{4}\varepsilon_{4}^{5}\varepsilon_{6}^{6}}\!\big/_{D}$. If there is no product inhibition for enzyme1 at the beginning of the feedback loop, then $\varepsilon_{2}^{1} = 0$ and there is no path to enzyme 2 in the graph, therefore, enzyme 2's flux control is 0. In general, when there is no product inhibition of the enzyme leading into a feedback loop, the flux control is only shared by enzymes at the beginning and end of the loop[33]. A feedback system can be considered as a separable subsystem in a metabolic network, and I have just seen that it is likely that enzymes that link these subgroups share most of the metabolic flux control.

Given the definition of N, it is proportional to the number of paths from the source node to an enzyme. An enzyme's N and also C, the flux control coefficient, will be higher if the enzyme lies on many paths from the source node. In social network analysis, betweenness centrality is a direct measure of this property[65]. So partitioning and betweenness scores can be used to find enzymes with the majority of flux control. And these enzymes with high flux control have been shown to be ideal drug targets[66].

Sensitivity Analysis with Signal Flow Graphs

A signal flow graph (SFG), or Mason graph is a graph that represents any system of equations[44]. The nodes are system’s variables, and the edges are weighted by the partial derivative of the target with respect to the source node. Paths through the graph are used to calculate transmittances, T, or transfer functions between distant nodes. Once a transfer function is determined, its derivative can be taken with respect to any system parameter. This is defined as the sensitivity, S, of the output to that parameter. I hypothesized that a biological system’s phenotype or output would be most sensitive to parameters involved in spanning separable subsystems. By Mason’s Rule, Tj,i = {Contribution of input j to output i} / input i = Sum of [Pk*Dk / D] for all paths k from j to i; D = determinant of the system = 1 - Sum of cycle gains + Sum of [Sum of [(-1)^k*Product of disjoint cycle gains taken k at a time]]; Pk = gain of kth direct path from input to output; Dk = D – gains of cycles that touch Pk. For example, if we have a SFG with three cycles: L1, L2, L3; and one path, P, from j to i, that touches L1; Tji = P(1-L2-L3+L2,L3)/(1-L1-L2-L3+L1L2+L1L3+L2L3-L1L2L3). If our network has one input and one output, then it is reasonable to say that parameters associated with many direct paths would yield greater S than those on fewer direct paths. This is measured by betweenness centrality. Also, parameters associated with cycles, but not the direct path, would appear in D many times and S = [DN’-ND’]/ D^2, this would be smaller than N’/D, the S for parameters only in direct paths. n’/D >? [DN’-ND’]/D^2; {n’ = dN/dp.on.direct.path, N’ = dN/dp.in.cycle} → if, N’<<1 & N’,D’,N,D > 0 then, n’/N > -D’/D; or, the output is more sensitive to parameters that occur in direct paths from input, when compared to parameters in cycles and off the direct path.

Scale-Free Networks

The distribution of node degree, (the number of neighbors), has a significant effect on a network's susceptibility to random and informed attack. To illustrate one can compare biological and Internet networks. The latter have a power-law distribution, with random networks of exponential or gaussian distributions. If a network's diameter, (average length of a network's shortest paths), and largest cluster size, (number of connected nodes), are tracked while nodes are removed from the network randomly, it has been shown that scale-free networks' diameter remains very low, while exponential networks exhibit a steady increase in diameter[1, 27]. The largest cluster size changes very little for scale-free networks under random attack, but clusters rapidly fragment for exponential networks. This all changes for informed attack, where nodes are removed in order of descending degree. Scale-free networks have very low survivability to this kind of attack, which is a good thing for drug design. Since important nodes in this study are defined as those with high degree, then I can extend the concept of importance with social network analysis measures of centrality. Degree is a measure of network centrality and so is betweenness, the number of shortest paths across a network that a node lies on. Betweenness is also an indicator of a net's articulation points[65], which if removed, cut a network into separate components.

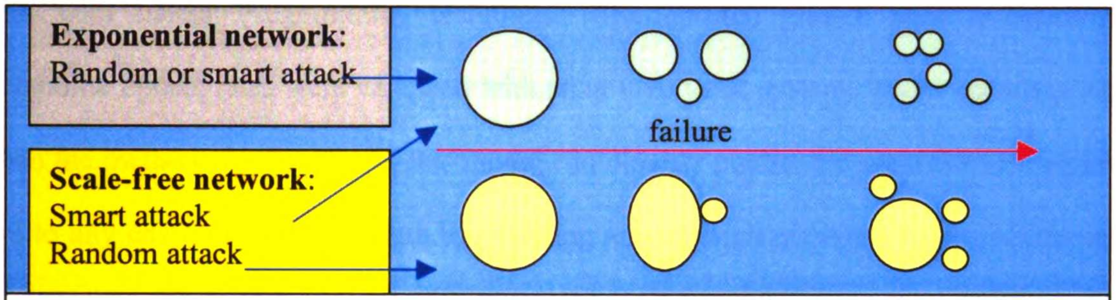


Figure 3-3 Disintegration modes due to attack. Figure redrawn from Albert et.al. [1]

Phenotype Phase Plane Analysis or Network Flow

The analytical method called Phenotype Phase Plane Analysis has been used to show that yeast maximizes its growth rate via optimizing its metabolism[67]. It was done by predicting growth rate due to substrate uptake rates and finding the optimal growth rates with Linear Programming techniques. The optimal growth rates at specified metabolite uptake rates were validated with an in vivo yeast system. While not suggested when the method was presented, this model can directly predict the sensitivity of growth rate to any enzyme's catalysis rate by allowing an enzyme's maximum rate to vary and by taking the slope of the growth rate with respect to the enzyme rate. It follows that the enzyme for which the growth rate is most sensitive can be found by allowing all rates to vary sequentially and in combination. Additionally, the maximum network flux can also be found by finding the min-cut and calculating its capacity—essentially, this is network flow analysis or bottleneck analysis[68]. If the sensitivity of the growth rate to the enzymes controlling the min-cut were zero, then changing the capacity of the min-cut would not change the max growth rate. Since the min-cut's capacity directly gives an upper bound on the max growth rate, then it must have non-zero sensitivity and, since growth rate is always optimized to this maximum capacity, then no other rate would have a larger sensitivity. This min-cut is the set of edges with minimum flow capacity that, when deleted, separates the input from the output. One accepted method of finding a network's min-cut is with graph partitioning techniques[69]. This again implies that elements that link separable network subsystems are quite important for understanding and controlling a system's behavior.

Analyzing Biological Network Topology

Each of the concepts above reinforces the important role of a system's topology to system behavior. Topological analyses require steps that identify network modularity, that select elements linking separate modules, and that develop a prioritization metric for the selected elements. Identifying modules is similar to the task of partitioning networks. Finding the best partitioning of a graph is an NP-complete problem, but there are many algorithms that provide satisfactory partitions in short order. Since the graph-partitioning problem in computer science is most often used to optimally share work among a number of computer processors, the partitions are balanced. When the partitions are balanced each partition contains roughly the same number of nodes. Such balancing may not be desirable if the partitions are used to represent biological modules. Fortunately, not all graph-partitioning methods produce balanced partitions. Spectral partitioning methods, for example, produce *natural*, unbalanced partitions[70].

Graph clustering in general is another potential set of techniques for grouping network elements and discovering natural orderings within networks. One of the many clustering algorithms paired with an optimization function could reveal modular organization without a balancing constraint. Several social network analyses (SNA) have used clustering techniques to discover cohesive subgroups within social networks[71]. Cohesive subgroups in SNA represent subsets of nodes that have relatively strong or frequent interactions among them, an idea comparable to modularity in biological networks.

The prioritization metric used to specify potential therapeutic targets should be one that complements the partitioning or clustering algorithm. If therapeutic targets are most effective when they act as gatekeepers for interactions / communications between a system's inputs and an output / disease biomarker, then there are several centrality metrics used in SNA which seek to quantify this. Betweenness centrality is based on the idea that interactions between two nonadjacent actors might depend in a special way upon the actors who lie on paths between them, and more so for paths of shorter length[72, 73]. This metric and other centrality metrics can be formulated for a single node or a group of nodes in directed or undirected graphs. Another metric, information centrality, extends the rationale of betweenness centrality to include longer paths in addition to the shortest paths[74]. When a network can be organized into multiple cohesive subgroups, either by partitioning or clustering, the nodes that lie between groups may have high betweenness centrality. Betweenness centrality has been linked to the idea of "cut-points, " which are nodes that when deleted break the network into separate components. Indeed, one method for finding cut-points is to delete nodes in order of their betweenness centrality and count the number of components generated. This was done for an AIDS network and not all nodes with high betweenness centrality served as cut-points[74]. It is likely that no single centrality measure or other single metric indicating a potential drug target will be sufficient, and a combination of metrics will be more useful—similarly for algorithms that reveal network organization.

Chapter 4: Motivating Hypotheses

The aim of this research is to develop and engineer objective analysis techniques for large and complicated biological systems, for the purpose of assisting in therapeutic target discovery and prioritization. The techniques are applied to graph models of biological systems of interest. In developing the techniques several concepts from graph theory and social network analysis are used to test the following hypotheses.

1.) A modular system is most sensitive to the changes in elements integrating modules together. *For a biological system, modeled by a system of equations, a subset of variables is generally chosen to represent the overall behavior of the system. This subset of variables is sometimes called a biomarker subset. This hypothesis implies that the sensitivity of the biomarkers to integrating variables is greater than to nonintegrators. Thus, these integrators may be good therapeutic targets or surrogate markers for the system's phenotype.*

2.) In a system modeled as a graph, those elements integrating modules together have higher betweenness centrality than other nodes. *Given a set of integrating elements, each node's betweenness centrality can be used as a relative rank. In the case where a proper assignment of nodes into modules is difficult to obtain, betweenness centrality can be used to prioritize nodes, with high-ranking nodes being more likely to be an integrating node in the as yet unfound modular organization.*

3.) In a system modeled as a graph, high betweenness centrality is a property of nodes that are more likely to be influential in the system. *In general, a ranking of* *nodes according to their betweenness centralities will prioritize a system's elements in a manner reflecting their likelihood to be effective therapeutic targets.*

In order to test these hypotheses, graph modeling a biological system of interest must be organized into modules and nodes must be prioritized. The modules in a graph are approximated by optimal graph partitions, graph clusters, and the graph's strongly connected components. The nodes are prioritized according to node centralities. Both betweenness and closeness centralities are evaluated. Also, knowledge of the graph's articulation points and bridges is evaluated for contribution to target discovery and prioritization. The following chapter describes in greater detail the aforementioned network analysis techniques.

Chapter 5: Methods

Graph Models of Biological Systems

Data Management

In preparation for analyses all entities within the system to be analyzed are accounted for and given a label and an index ID within a database. Each direct causal relationship in the system is similarly accounted for and stored within another table containing three fields: the entity IDs comprising the binary relation and an indexed ID for said relation. With this accounting the system is translated into a standard graph form, with entities being the nodes of a graph and the relations being the edges of said graph.

Box 1. Finding SCCs Given G(V,E)

```
1.  DFS(G); keep last seen time f(v) for each node v.
2.  get Gt =G transposed.
3.  DFS(Gt), in DFS search each v in order of decreasing f(v).
4.  output each tree found in step 3 as a new SCC.
```

Box 2. Finding Bridges Given G(V,E)

```
compute low(v) for each v.
for each v do:
  for each child u of v do:
    if low(u) = pre(u) then:
      (v,u) is a bridge.
---Computing low(v)---------------------
explore(v):
  low(v) ← pre(v) # pre=time first seen v
  for each e(v,w):
    if w not visited yet:
      explore(w)
      low(v)←min(low(v),low(w))
    else: low(v) ← min(low(v),pre(w))
```

Box 3. Finding Articulation Points Given G(V,E)

```
compute low(v) for v ⊂ V.
for each v do:
  if v is the root & has >1 child:
    v is an articulation point.
  else: for each child u of v do:
    if low(u)≥pre(v):
      v is an articulation point.
```

Connected Components & Strongly Connected Components

A component of a graph is a set of nodes that are connected to each other by an undirected path. A strongly connected component (SCC) is a set of nodes that are connected to each other by a directed path. Thus, a SCC fulfills the property of a module that it has a stronger connection to internal vs. external elements. The pseudo-code used to find a graph's SCCs is detailed in Box 1. It mainly consists of two depth-first-search topological orderings of the graph's nodes. During the second ordering stage the SCCs are output[75].

Articulation Points & Bridges

Our operating hypothesis is that nodes incident with edges that connect different modules are more likely to be essential to the system's phenotype. A stricter indicator of importance is incidence with one of the graph's bridges. A bridge is defined as an edge that, if removed, results in an increase in the number of components. The pseudo-code for finding a graph's bridges is detailed in Box 2. A nodal equivalent to the bridge is an articulation point, a node that, if removed, results in an increase in the number of components. The pseudo-code for finding an articulation point is described in Box 3. Spans are edges that connect different SCCs. They are identified by looking at each of the graph's edges and determining whether or not the two incident nodes have different SCC assignments.

Centrality Measures

Closeness

The two centrality measures used in developing the target prioritization heuristic: closeness and betweenness centralities[65]. Closeness centrality indicates the likelihood that a node maximally influences many other nodes. It is calculated as the sum of graph distances between the node in question and all other nodes. The graph distances are computed with Dijkstra's shortest path algorithm[75]. Because there may be more than one 'shortest' path between two nodes, Dijkstra's algorithm was modified to handle multiple 'shortest' paths. Lower closeness scores indicate more central nodes.

Betweenness

Betweenness centrality indicates the potential for a node to regulate or influence interactions between other nodes, and makes use of the modified shortest-paths algorithm. After the shortest paths between all node pairs are found, for every node an accounting is made of the number of shortest-paths in which it appears, scaled by the inverse length of each path. Higher betweenness scores indicate more central nodes.

Graph Clustering

Graph clustering is used to place graph nodes into groups that are maximally isolated from the rest of the graph, and are maximally connected internally. This is achieved by a hierarchical clustering algorithm, where clusters are assessed as they grow. If a cluster is found to have the properties previously stated, then its growth is halted and no longer considered in the rest of the clustering process. The closest two clusters are merged at each phase to form a new cluster. This process repeats until there is one cluster. The distance between clusters is determined using three measures: minimal internode distance, maximal internode distance, and average internode distance

Each newly formed cluster is assessed for its cohesiveness or modularity. There are three choices for determining cohesivity: density ratio, and two versions of the intracluster edge observation likelihood. The density ratio is ratio of the cluster density and the overall graph density. Density is the observed number of edges divided by the maximum possible number of edges. A lower threshold of 7 is suggested for a suitably cohesive cluster if the density ratio is used.

The other option for assessing cohesivity calculates the probability of observing at

$$\sum_{k=q}^{\min(1+L,1+gs(gs-1)/2)} \binom{L}{k}\binom{g(g-1)/2-L}{gs(gs-1)/2-k} \div \binom{g(g-1)/2}{gs(gs-1)/2} \qquad (5\text{-}1)$$

least the number of edges within a cluster, expression 5-1. In expression 5-1: L = number of Total Edges, g = number of Total Nodes, gs = number of Cluster Nodes. The threshold for this measure is at most 0.00025. This value is graph dependent. In order to calculate this probability, the combinatorics had to be rewritten, hence the two versions. Initially the division of factorials caused an overflow error. This division was algebraically simplified and rewritten in a new combinatorial function, correcting the overflow problem. The factorial F(a) is calculated with Sterling's formula for a > 10. The error in Sterling's for a=10 is 0.8%. For a <=10 the factorial is calculated by recursive multiplication.

Graph Partitioning

Balanced Partitioning

A graph partitioning achieving a minimal cut is expected to have partitions that fit the definition of a 'module,' because of the isolation provided to nodes in a partition by the relatively few edges connecting the partition to the rest of the network. The partitioning algorithm used here is METIS[2, 5]. METIS is a fast multilevel recursive graph partitioner. It operates by first coarsening the graph, then making an initial partition, and finally uncoarsening the graph, while at each level refining the partition so as to achieve a minimal number of edges cut by the partitioning. The precompiled *pmetis.exe* code accepts as input a graph to be partitioned and the number of partitions desired. It outputs a list of partition assignments for each node in a flatfile named after the input graph. METIS and many other public graph partition tools were designed for computer science researchers needing to not only partition work efficiently among processors, but to attempt to balance the amount of work partitioned among processors. For example, if a finite-element model is optimally divided in two sections having 400 nodes and 20 nodes, the processor working on the model section with 400 nodes will be overworked compared to the processor with the smaller 20 node model. Even though, the partitioning has minimized interprocessor communication, computing power is not efficiently being used. This is why METIS includes an extra requirement that partitions be as similar in size as possible.

Unbalanced Partitioning

As opposed to the *balanced* partition determined by METIS™, a spectral graph partitioner was developed to generate natural or *unbalanced* partitions. The eigenspace for a special adjacency matrix is calculated. There is a choice in the type of matrix used: unsigned adjacency matrix, degree matrix, Laplacian matrix, and Disconnection matrix. Currently the Disconnection matrix is used in calculating the eigenspace, but it is quite similar to the Laplacian matrix, another good choice. Once the eigenspace is calculated the 3 *maximal* (in general this means closest to 0) eigenvalues are selected along with their respective eigenvectors. In order to bisect a graph, the eigenvector for the maximal eigenvalue is selected and each node is associated with a scalar from this vector. The nodes are then assigned to two groups according to the sign of their associated scalar. To create an arbitrary number of partitions, this method uses recursive sectioning. Because the three eigenvectors are used, eight sections can be created per iteration. A partitioning scheduler was developed to decide whether 8, 4, 3, or 2 new partitions are needed at each step in the process of attaining the desired number of partitions. Sometimes, there are several sections and not all of them need to be partitioned. Some sections will partition better than others. The quality of a partitioning is taken to be the number of edges cut—smaller is better. The best section to partition is the one with the best maximal eigenvalue. Briefly, the entire method is as follows: decide how many partition(s) needed this round, sort sections in order of maximal eigenvalues, chose the best sections to partition and partition, repeat till desired number of sections is achieved.

Finding an Optimal Partitioning

When using a graph partitioner to find modular structure, one must select the number of partitions that it should produce. Since this number (*k*) is not known *a priori,* successive partitions are generated for different values of *k*. When one observes a significant negative deviation from the trend of edges cut vs. k, the value of k at the deviation is believed to represent the best partitioning.

Chapter 6: Predictions of Therapeutic Targets and Variable Sensitivity via Prioritizations from Partitioning & Centrality Analyses

OVERVIEW

A topological approach is presented for the analysis of control and system organization in quantitative and qualitative biological network models. In this approach, the network model is represented by an unweighted graph. From a heuristic inspection of the graph's topology, modular organization is ascertained, and each graph element is assessed for it's potential significance in system wide influence or control. The benefits of this approach are: increased understanding of biological system organization, objective discovery and prioritization of potential therapeutic targets, qualitative sensitivity analysis of very large system models, and applicability to a wide range of qualitative and qualitative systems regardless of size. Results are obtained for a blood coagulation model and a much larger model of human obesity.

INTRODUCTION

Available to any researcher is a vast array of biological data that is accessible within the classic scientific literature and via the Internet—much of it being relational but accessed and analyzed in a piecemeal manner[52, 54]. There are attempts to organize this data into molecular or genetic networks to illustrate specific signaling pathways and networks and, on a larger scale, genomic interactions[59, 76]. Once created, what is one to do with the enormity of these complicated biological networks? A good initial use is the visualization of gene expression or metabolic pathway activity[8]. Another is the construction of predictive models of biological systems. Lately there is much work aimed at using *in silico* simulations of biology in order to validate pharmaceutical targets for the control of disease and modulation of phenotype[9, 77]. These quantitative models can be quite large and require some care in their implementation, but there are other large models used to aid therapeutic target development. Qualitative models of genetic and protein interaction are currently used in conjunction with high throughput screening to discover potential therapeutic targets[51, 78]. This kind of discovery is more difficult to do with large, quantitative models of biology, because of the computational cost or subjectiveness of the search. Networks can be visualized, but it is hard to determine which aspects may be most significant simply by looking at them—methodologies for *useful* graphical layout of biological networks is a currently being researched by a few groups[60].

Methodologies are in use in fields of scientific endeavor not directly associated with bioengineering, biotechnology, biocomputing, or bioinformatics that can provide some guidance as to the kind of network analysis that might be needed to develop to improve one's understanding of a system's organization well enough to identify likely therapeutic targets. One set of analyses used in Social Network Analysis concentrates on developing metrics for an individual's importance in the network. Some of these metrics are called *centrality* measures. These centrality measures have been reframed in the light of metabolic control theory and the hypotheses of modular organization in biology in order to provide new concepts and tools for biological network analysis[79].

The following are analyses of a qualitative model of blood coagulation and a separate quantitative model of obesity. Both models are assessed for modular organization and integrating nodes identified and further prioritized. The prioritizations are evaluated by comparison to known drug targets or biomarkers for these systems. The prioritizations for the quantitative model's variables are validated by comparison to the results of calculating the sensitivity of obesity progression to changes in variables of different prioritization. The blood coagulation model is taken from an effect diagram used by Mounts and Liebman[26] to illustrate the steps involved in the coagulation process. The much larger obesity model is abstracted from a quantitative system of equations used to model the progression of obesity in virtual patients. This model was developed by Entelos and distributed under the name Obesity PhysioLab[9]. Its main use is the evaluation of new therapies to treat obesity, but here I use it to predict new therapeutic targets.

RESULTS

Blood Coagulation

The model of blood coagulation as analyzed by Mounts and Liebman was analyzed for modularity using graph-partitioning techniques. The node list is shown in Table 6-1. The optimal number of partitions was found to be six. The nodes were then ranked by the fraction of their neighbors in another partition, averaged over all k-partitionings. The top ranking nodes [tf-f7a, Ca, f9a, f8a-f9a, f10a, thrombin] were located on the periphery of and within the central pathway and they either participate in feedback [f10a] or represent branch points [thrombin]. When considering the optimal partitioning of the graph into 6 parts, the preceding nodes were also ranked highly, but a new node [f5a-f10a] had the highest rank. This node is associated with the conversion of prothrombin to thrombin. Once the most influential nodes were selected by their roles in connecting partitions, *betweenness* and *closeness* centrality scores for all nodes were calculated for both directed and undirected version of the model's graph. The top ranking betweenness and closeness scorers were some of the same nodes identified previously by the simpler 'fractional' measure, except for Factor VIII $_{\text{activated}}$ (f8a) which had a significant closeness centrality score. Table 6-2 shows that those entities identified as significant are directly associated with the major coagulation disorders, and they are well documented in the disease literature. The blood coagulation graph is shown in Figure 6-1 with nodes sized to indicate their undirected betweenness centrality score.

Table 6-1. Entities of Blood Coagulation Model

id	Name	Abbreviation
1	"vascular injury"	
2	"tissue factor"	Tf
3	"factor 7"	F7
4	"tf-f7"	
5	"tf-f7a"	
6	"Ca"	
7	"factor 9"	F9
8	"factor 10"	F10
9	"hmwk"	
10	"prekallikrein"	
11	"kallikrein"	
12	"factor 12"	F12
13	"factor 12a"	F12a
14	"factor 11"	F11
15	"factor 11a"	F11
17	"factor 9a"	F9a
18	"factor 8a"	F8a
19	"factor 8"	F8
20	"platelet"	
21	"f8a-f9a"	
22	"factor 10a"	F10a
23	"factor 5"	F5
24	"factor 5a"	F5a
25	"f5a-f10a"	
26	"prothrombin"	
27	"thrombin"	
28	"fibrinogen"	
29	"fibrin"	
30	"factor 13"	F13
31	"factor 13a"	F13a
32	"crosslinked fibrin"	cFibrin

Table 6-2. Blood Coagulation Disorders

Disorder	Cause (predicted targets in red)	Treatment
Hemophilia A	defect in F8	Treated w/ rF8
Hemophilia B	defect in F9	Treated w/ plasma
Parahemophilia	F5 deficiency	Plasma Treatment
Stuart-Prower factor deficiency	F10	Replacement F10
Hypoproconvertinemia	F7 deficiency	Lethal
Hypoprothrombinemia	Thrombin deficiency	Treat w/ Vitamin K or corticosteroids and high-dose Ig

Table 6-3. Significant Relations || Bridges & Spans

Bridge	
From-node (parent)	To-node (child)
Ca	FIXa
Thrombin	Fibrin
Fibrin	CFibrin
FXIIIa	CFibrin
TF-FVIIa	FIXa
Spans	
FXIIa	FXIa
Thrombin	Fibrin
Thrombin	FXIIIa

In a further effort to identify modular structure in the model and critical linking entities, the graph's *strongly connected components* (SCC), *spans*, *bridges*, and *articulation points* were identified. The two main SCCs found were the intrinsic cycle, and thrombin feedback into the common pathway. SCCs with only one node were ignored. Three articulation points were found: tf-f8a, f9a, and thrombin. The bridges and spans are listed

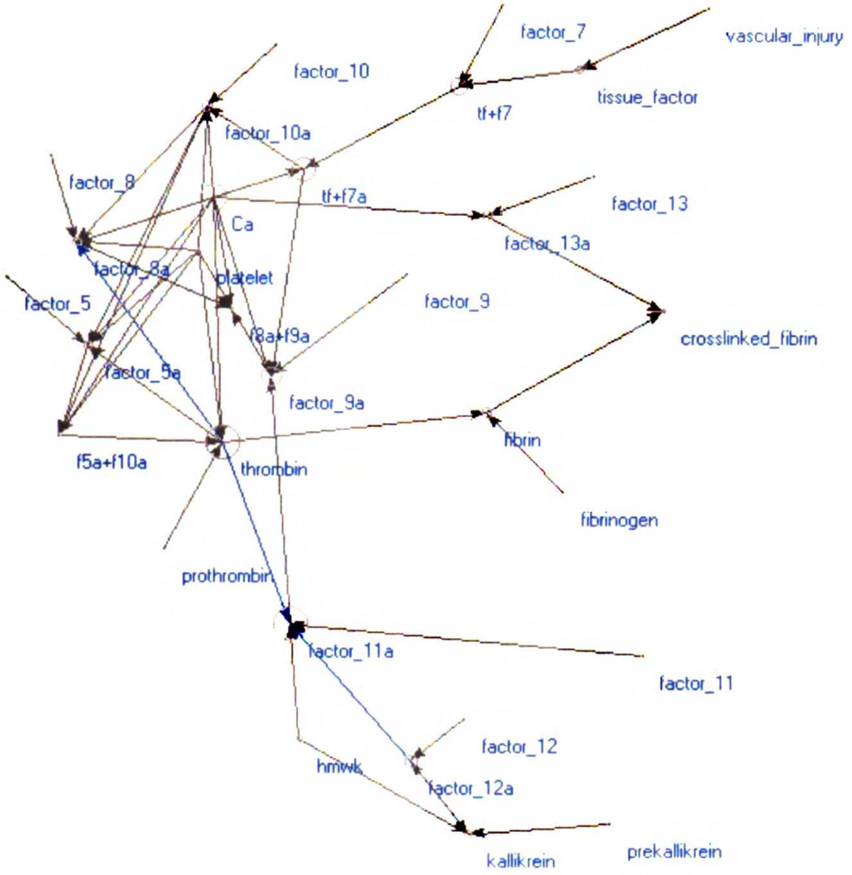


Figure 6-1 Blood coagulation network shown with MDS layout. Larger nodes have better betweenness centrality when calculated for the undirected graph.

in Table 6-3. The SCC algorithm finds cycles within pathways. These cycles roughly correspond with the partitioning results but preserve and highlight biological organization. Species identified as articulation points have high centrality values, and are likely to be critical species. The factors that correspond to the SCC spans and bridges, if inhibited would destroy the coagulation pathway. They are found on the periphery of the SCCs.

A summary metric was created that includes the various topological measures used in this study to identify critical network entities (see the following Methods section).

This summary metric was used to rank the nodes using weights from a directed graph along with a separate ranking using weights from an undirected graph. The summary metric, based on the directed graph, reinforced the importance of thrombin, tissue-factor, Factor VII, and Factor VIII. Fibrin and Factor XI were newly selected targets. The summary metric, based on the undirected graph, reinforced the same nodes, including fibrin, but did not highlight Factor XI.

Obesity PhysioLab™ v1.1

The obesity model was translated into a graph where there was a node for each variable in the model and a directed edge between each pair of nodes that had a direct causal relationship in the model. No unique partitioning was found using the method of consecutive partitioning. The search for SCCs revealed 109 SCCs. The majority of SCCs contain a single node; one contained 3 nodes; another 2 nodes, and another with 57 nodes. 102 bridges, 61 spans, and 25 articulation points were found. The nodes in the 90th percentile of *directed betweenness centrality* scores, nodes that are incident to spans or bridges, and the articulation points were selected for anecdotal validation as therapeutic targets. When checked for coverage of the known obesity therapeutic targets, 60%, 80%, and 69% of the on-market, clinical, and pre-clinical targets were directly matched. When the immediate neighbors of predicted targets were also considered, then 100% of known targets were matched.

The nodes with high betweenness centrality and the articulation points were used to design and test *in silico* therapies for simulation by the PhysioLab™. These "trials" simulated 6 months of therapy with a weight stabilized 90kg individual. The measured sensitivities of adipose fat mass and rate of fat change are shown to have a direct correlation, and fat mass sensitivity correlates well with % fat lost and %fat gained, but rate sensitivity only correlates well with the rate sensitivity. With this observation in mind, fat mass sensitivity was used to evaluate the predictive power of each topology measure. Five targets resulted in weight loss of ~5% or greater. The maximum weight loss was ~61%. All Five with demonstrated weight loss had sensitivity magnitudes of ~0.2 or greater. There were seven other targets with good sensitivities, but insignificant weight loss

Undirected and directed closeness and betweenness centrality scores were assessed for their ability to predict the measured sensitivity of adipose mass to interventions on the various targets. Figure 6-6 shows that as betweenness centrality score increases, the fraction of sensitivities greater than or equal to 0.2 increases. This is the trend for both undirected (a) and directed (b) scores. Figure 6-7a shows the same trend for closeness centrality, but the directed score (b) has a similar fraction of good sensitivities (> 0.2) for the entire range of closeness scores. Notice that the likelihood of getting a good sensitivity score increases for smaller values of undirected closeness centrality. This result is expected given the definition of closeness centrality.

DISCUSSION

If a graph (representing a system of interacting entities) can be organized into modules, said modules being groups of nodes that are better connected with each other than to the rest of the graph, then those nodes that connect different modules together are more central than other nodes. This centrality concept is demonstrated by the blood coagulation results, where nodes shown to have many neighbors in other partitions are the most central, as measured by both betweenness centrality and closeness centrality. Such a relationship may not always be the case, and is affected by the module-defining scheme and the individual graph's characteristic topology. When one module-defining approach fails, then centrality alone may be used to find nodes that may connect the as yet unidentified modules, a situation that is made more likely when the centrality score distribution has a long tail. It was also shown that the most central nodes were associated with blood disorders and the less central nodes had fewer direct links to single gene disorders.

Figure 6-2 Observed Sensitivity (S) Fractions

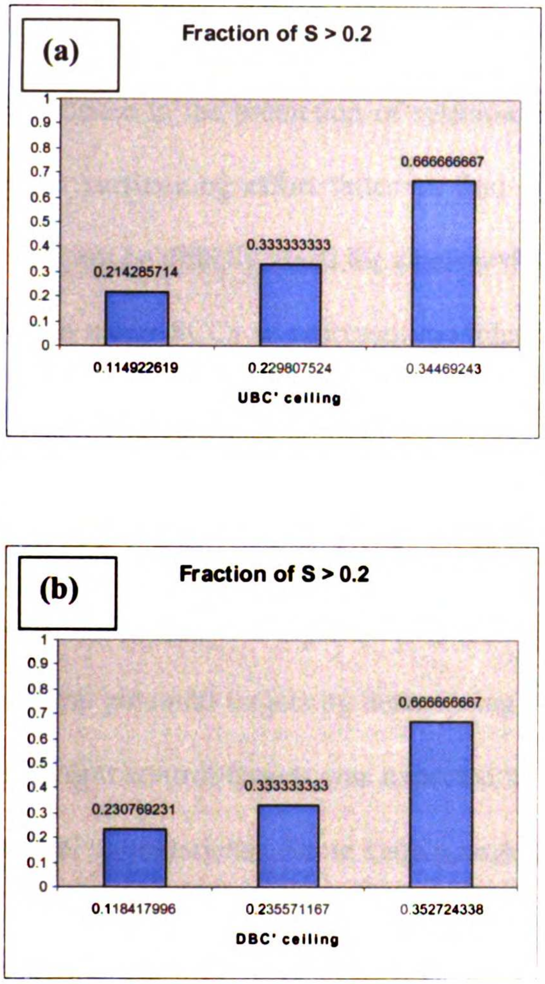
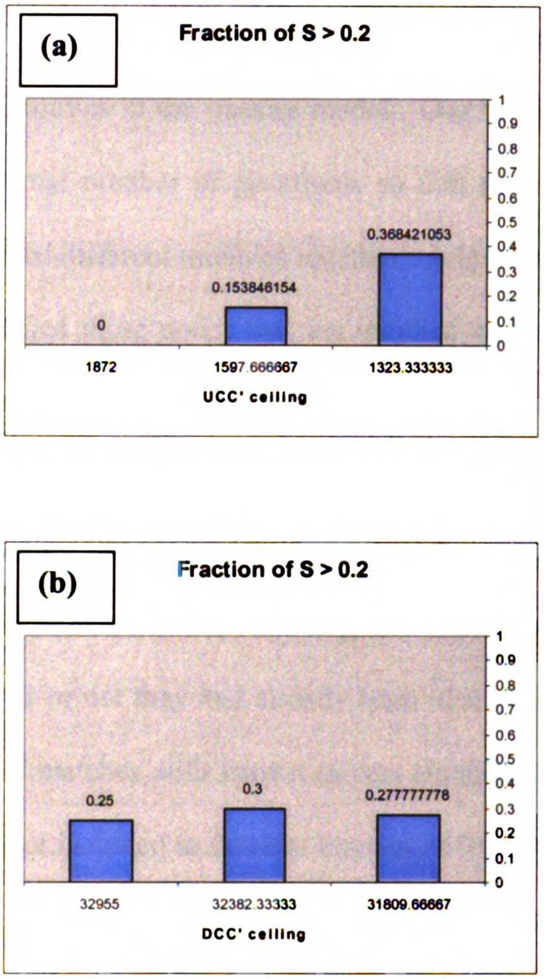


Figure 6-3 Observed Sensitivity (S) Fractions



The range of normalized centrality scores was divided equally into 3 subranges, and the fraction of targets in each range with an associated sensitivity (S) of 0.2 or greater was recorded. The ceiling of each subrange is indicated on the axis of abcissas. UBC' = normalized undirected betweenness centrality; DBC' = normalized directed betweenness centrality; UCC' = normalized undirected closeness centrality; DCC' = normalized directed closeness centrality.

As an example of the importance of the module-defining scheme, fibrin was not selected initially as being especially critical to the coagulation system when just the graph partitioner was used to locate modules. However, fibrin was selected when SCCs plus their spans were used to define modularity along with bridge and articulation point information. The use of multiple methods for identifying modularity provides flexibility and increases one's confidence in selecting system-critical entities.

Having multiple methods for finding modular organization increased our confidence in the prediction of system-critical entities in the obesity model. Our initial graph partitioning effort failed to find an optimal number of partitions, so that nodes could not be directly assed for whether they linked different modules together. A broader search using SCCs to represent modules identified those nodes that are incident with a span, and are therefore connecting different "modules". Furthermore, the identification of the graph's bridges and articulation points indicated topologically critical nodes. This information was combined with a node's centrality scores to identify entities that were most likely to be critical, and so would be good candidate therapeutic targets. Validation of these potential targets by determining whether or not they had already been identified as weight control targets was expected to reveal matches with known targets along with several novel targets. Some known targets are not included in Obesity PhysioLab™ v1.1 so they were not included in the validation. On average 70% of known therapeutic targets were directly predicted as being topologically critical. When the predicted node's neighbors were also counted, the match increased to 100%. There is a good reason for including a node's neighbors in the prediction of significant nodes. Some of the nodes identified as likely good targets cannot be directly adjusted. The node's neighbor must be adjusted to affect the desired.

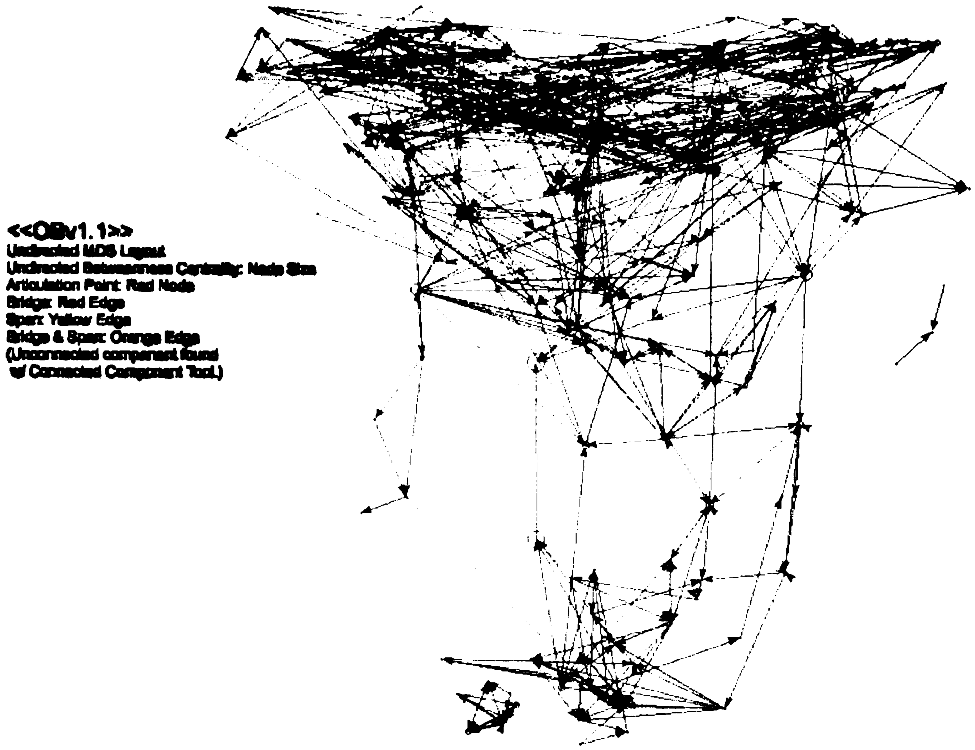

<<OBv1.1>>
Undirected MDS Layout
Undirected Betweenness Centrality: Node Size
Articulation Point: Red Node
Bridge: Red Edge
Span: Yellow Edge
Bridge & Span: Orange Edge
(Unconnected component found
w/ Connected Component Tool.)

Figure 6-4 OB v1.1: Effect diagram of obesity model v1.1 displaying portions of analysis results. MDS = multidimensional scaling.

Having a critical topological location and better centrality score increases the likelihood that the system will be sensitive to modulation of a particular node. If a node has high betweenness centrality, it has the potential to influence the interaction between many other nodes. If a node's closeness centrality is low then it is very close to other nodes in the network, and is expected to have a better chance of affecting a greater proportion of the system and therefore the system phenotype. As a potentially critical gatekeeper of influence and information flow through (directed graph) or within (undirected graph) a biological system, nodes with high betweenness (and for some systems, closeness) are expected to have a higher potential to affect a biological systems behavior or phenotype. To verify that presumption, nodes having various centralities were first identified. Each one was then used as a point for intervention in the obesity model and the impact on a simple obesity biomarker was recorded. Betweenness centrality, calculated either for directed or undirected graphs, is positively correlated with the likelihood of a node being critical to the system's phenotype. Directed closeness centrality is negatively correlated, yet undirected closeness is not.

It is important to note that this analysis used an unweighted graph. All interactions between entities in the network are treated equally. Such equality certainly is not the case in reality. That consideration is expected to have a great deal to do with the accuracy of any predictions.

The algorithms used to identify a system's modularity and to calculate centralities can also incorporate valued relations—weighted edges. The incorporation of edge weights would change centrality scores and therefore different modules will be identified. The concept of edge weight will need to be a focus for future research. Choice of weights will be different depending on the source of information for each particular representation of a biological network. For example, a genetic network compiled from microarray measured gene expression[16] or concurrence in the scientific literature[14] may have edge weights based on the confidence associated with one gene influencing another. A metabolic network could have edge weights representing a reaction's transition time, but implementing this becomes complicated for nonlinear reactions with multiple molecular species. With this in mind, the work presented here is a demonstration of usefulness in representing biological systems by unweighted graphs and analyzing their topology and node centralities.

METHODS

Summary Selection Method

This score (s) is calculated as in Eq 6-1. A, B, C, and D are equal to one. β & σ are group betweenness centrality scores defined in Eq 6-2 for bridges and spans, respectively—a group in this case is defined as the two nodes incident with the same edge. C'(n) is the normalized betweenness centrality score, Eq 6-3, and g is the number of nodes in graph. The node with the maximum centrality score in a group in denoted by n*, and conversely n` has the smallest score. a equals one if it is an articulation point, and zero otherwise. b equals one if it is adjacent to a span, and zero otherwise. c equals one if it is adjacent to a bridge, and zero otherwise. d equals C'(n). I define significant scoring nodes to be those with scores greater than two standard deviations away from the mean of the largest mode in the distribution of scores. This score was calculated twice, once with all centrality scores based on a directed graph, and another time using scores based on an undirected graph.

$$s = Aa + Bb\sigma + Cc\beta + Dd \qquad (6\text{-}1)$$

$$\sigma, \beta = C'(n^*) - C'(n`) \qquad (6\text{-}2)$$

$$C'(n) = \frac{2C(n)}{(g-1)(g-2)} \qquad (6\text{-}3)$$

Validation

Anecdotal: Blood

Each molecular entity in the blood coagulation model was used to search the OMIM database for appearances in record titles and record text. OMIM is a database of human diseases, syndromes, and the genes associated with them[80]. It is assumed that if an entity has many such appearances or 'hits,' it is more significant to proper function of the blood coagulation pathway. The heuristic's target predictions are validated by comparison to the results from OMIM. Also predictions were checked for direct association with known blood diseases in the coagulation disorder literature.

Anecdotal: Obesity

The targets for those weight control drugs that are documented in public literature were compiled and classified as to their stage within the drug development process. Those targets that could be modeled by the obesity model were selected and joined with a list of associated entities from the model. The heuristic's predictions were compared to this list of modeled targets. The heuristic was expected to predict many of the documented obesity targets and to identify some novel targets. A known target is considered covered if the heuristic predicts one of the modeled entities associated with it. The fraction of known targets directly covered was determined. Another coverage fraction was calculated by considering matches with a predicted target and its immediate neighbors in the graph. The neighbors of a predicted target were considered for the reasons already presented.

In silico Validation for Obesity

The Obesity v1.1 PhysioLab™ from Entelos® was used to perform simulations of interventions inspired by the target predictions provided by the heuristic. Several other simulations were done for model entities that were predicted not to be good targets. The main aspects of the virtual patient's profile were an obese 90kg individual with voluntary eating for a simulated study-duration of 6 months. Parametric experiments were done for a range of parameter values around the 'normal' value given by Entelos®. The variables recorded for later analysis were maximal, minimal, and actual body weight, adipocyte fat mass, muscle mass, other tissue mass, and rate of fat change. From these variables were calculated: percent fat lost and gained due to parameter changes, sensitivity of fat mass and sensitivity of rate of fat change to parameter changes, Eq 6-4.

$$S_b^a = \frac{b*\Delta a}{a*\Delta b} \approx \frac{\partial \ln(a)}{\partial \ln(b)} \qquad (6\text{-}4)$$

The accepted criteria for success in a weight control therapy are: moderate, sustained weight loss as fat mass of about 5% (FDA[81]) to 10% (CPMP[82]) vs. placebo; a rate of weight loss greater than 0.5 lb per week; and improvement in comorbidities that often accompany obesity. In these validation trials, loss of fat mass and sensitivity of fat mass were used to indicate relative success.

Chapter 7: Effect of Increasing a Model's Quality & Size on Target Prioritizations

OVERVIEW

Knowledge of a system's organization aids in the search for therapeutic targets that maximally affect the system. The topology of a system can be used to reveal the extent to which it is organized into modular subsystems and used to prioritize its elements according to their potential to be good therapeutic targets. The following section describes tests of the hypothesis that both elements which link different subsystems and network centrality metrics indicate elements to which the system is sensitive. This testing was done with an *in silico* model for validating obesity drugs and their targets. This model is a subsequent version of the obesity model analyzed previously, and it models obesity in much greater detail. Predictions were tested by their coverage of known and currently investigated therapeutic targets for obesity. Betweenness and closeness centrality measures are found to be predictive of biomarker-sensitive variables (targets) and that the predictions have a high coverage of known drug targets for obesity. The increased size of the model and the addition of new graph elements does not significantly affect the quality of target prioritizations. This work is an example of how a good strategy for therapeutic target prioritization can be developed with an analysis of system topology, focusing on subsystem organization and centrality.

INTRODUCTION

The available quantity of small models of various biological mechanisms directly and indirectly related to obesity, may be sufficient to assemble a larger model of the entire biology of obesity. Entelos® (Menlo Park, CA) believe that there is sufficient data to design an accurate top-down model of obesity. There are a few other validated models of biology that attempt to reconstruct the entire system in order to allow *in silico* experimentation [83, 84]. This paper will focus on the model system from Entelos® called the Obesity PhysioLab™ v3.0, hereafter referred to as OPL. It has proven to be accurate enough for some of the major and minor pharmaceutical companies to license it for internal research.

The OPL is a large, complicated system of equations modeling the processes that influence obesity. One of its uses is the validation of therapies to affect weight loss. Another important use is as a knowledge base of the biological processes involved in weight control, as recorded in the public literature. Users believe that it is sufficiently accurate as an *in silico* platform for the initial validation of potential drug targets and other therapeutic interventions. It is under continuous improvement, but its very size and complexity makes it difficult for a researcher to objectively suggest new potential targets. Sensitivity analysis of the model could provide an objective analysis of the whole system, not just those parts that are most familiar to a user. A one-to-one analysis with all model parameters and indicators of obesity would require a prohibitive amount of time. On an IBM Intellistation M-Pro, with 733Mhz dual Pentium processors and 512 MB RAM, a "one-year" *in silico* trial with data taken at minute intervals is complete in approximately 4.4 hours (Technical Observation 2001). With better technology the duration will be shorter. Because obesity is generally a multifactor disease, a many-to-many sensitivity analysis would be more appropriate. Such an analysis would take exponentially longer than the one-to-one procedure. A qualitative analysis that is not constrained by the time to simulate has the potential to allow a global and objective analysis of the system for potential points of therapeutic intervention.

Any model's underlying topology of variable interactions can, theoretically, be used to identify potential therapeutic targets, and can be used to aid evaluation of multiple target interventions. As mentioned, the topological approach of Sen compares control coefficients of metabolic pathways without measuring elasticity coefficients [41, 42]. Far *et al.* developed a qualitative sensitivity analysis methodology for systems modeled in the rule based causal format [50]. The methodology allows one to quickly determine which parameters cause instability in output variables. Both methodologies allow one to identify those elements of the system that have the greatest effect on system performance. Unfortunately each approach is quite specific to model formalisms.

More generally applicable studies have been done to reveal a biological system's organization and points of control. A few groups have studied the scale free characteristic of several biological networks. Jeong *et al.* found that a few network elements with an unusually high number of neighboring elements are key to the robustness of these networks to uninformed attack [1]. The number of neighboring elements of a node is termed degree in graph theory. Jeong *et al.* show that for an E. coli protein interaction network, a protein's degree is predictive of gene-knock-out lethality [61]. Prioritization of potential drug targets by their degree may not be very useful for disease models. The approach may return "unsuitable" targets. Some high degree elements will be nonspecific to a disease being studied. Interventions on these targets may cause unwanted side effects if perturbed.

Lauffenburger hypothesizes that biological systems are an organization of modular subunits [29]. A module is envisioned to be a set of elements with many interactions among themselves and few interactions with the rest of the system. This modular organization is hypothesized to account for some of the robustness to error, damage or attack seen in biological systems [28]. If biological systems are organized as networks of individually error-robust modules, then the interactions that link modules are at risk of being missed by any degree-prioritization. Yet these same interactions may be among the better candidate targets. Fell *et al.* have conducted a study of metabolic network organization in *E. coli.* They used a different measure to prioritize network elements [85]. The metric used is called closeness centrality in Social Network Analysis [65]. It is described in the Experimental Protocol section, and measures the distance of each network element from all others. Lower closeness scores indicate more central objects or modules. Elements closest to all others are ranked higher than more distant elements. This metric, like one based on degree, indicates elements important to many other elements but not those with potential for regulation of interaction between modules.

The above approach identifies modular organization within the model and then prioritizes the identified model objects according to their potential to act as gatekeepers of interactions between other model objects. Graph partitioning algorithms and graph clustering algorithms are used to identify modular organization within the OPL model. Betweenness centrality is described in the experimental protocol section. It is used to help prioritize elements according to their potential for maximally influencing the model's behavior.

Minimal inter-group interaction, or modularity, is a property sought by graph partitioning algorithms. The goal is to divide a graph into k subgroups with a minimal number of links between them. If modular structure cannot be found directly, then I use an alternative graph theoretical concept of a SCC (strongly connected component). It can be used to represent the connectedness of a module. If for all nodes x and w in a component (a connected set of nodes), there exists a directed path from x to w and from w to x that component is said to be strongly connected. Articulation points and bridges are also identified. They are potentially important, because their deletion from the network results in the creation of two or more separated components of the network [86]. Such a deletion would be expected to dramatically change the phenotype of the larger system. Such change is exactly what is desired of a good therapeutic target.

Betweenness centrality is used as a measure of the probability that a network element is controlling significant information flow between other elements. Information is assumed to flow along the shortest route between two elements, and more centrally located elements are expected to occur in a great deal of these shortest routes through the network [65]. It is important to note that there can be multiple shortest routes between two nodes. Betweenness centrality is a count of an element's occurrence in the shortest paths between each pair of other elements, scaled by the length of each path. In subsequent sections 'd' subscript for betweenness indicates that the paths are from a directed graph; 'u' would indicate an undirected graph.

Objects with the potential to maximally influence the major indicators of disease progression are selected from the prioritized model objects and then checked against a summary of known obesity drug targets. Subsequently, the sensitivity of disease biomarkers to an intervention on each predicted target is calculated. The resulting data provides an indication of the sensitivity of the OPL model for each predicted target. The data also provides a means for comparing how well each metric performed in finding good targets.

RESULTS

I have a heuristic for general network analysis to determine the extent of modular organization and to identify key integrative network elements. With large, complicated networks no single approach can be expected to work, so the heuristic uses multiple complementary approaches in tandem for determining system organization and to prioritize elements. Predictions of critical elements were validated by a comparison with known drug targets for obesity and those hypothesized targets at various levels of commercial investigation. The predicted target's prioritization is hypothesized to indicate the likelihood of OPL obesity biomarkers to be very sensitive to the target. The target prioritization was validated with *in silico* clinical trials done with the OPL v3.0.

Modularity of model: The model was partitioned with the METIS© multilevel graph partitioning software package [5]. The optimal number, *k*, of partitions was sought by obtaining partitionings for *k* between 1 and 21, and looking for a significant deviation from the trend of *edge-cut* vs. *k*. If two nodes are incident to an edge and each node is in a different partition, then that edge is *cut*. The *edge-cut* for a *k*-partition is a sum of edges

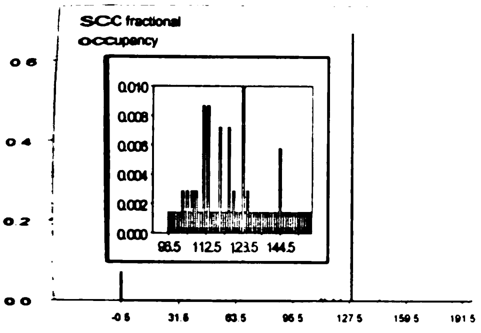


Figure 7-1. Fraction of nodes in each SCC. Inset shows SCCs 96 to 160.

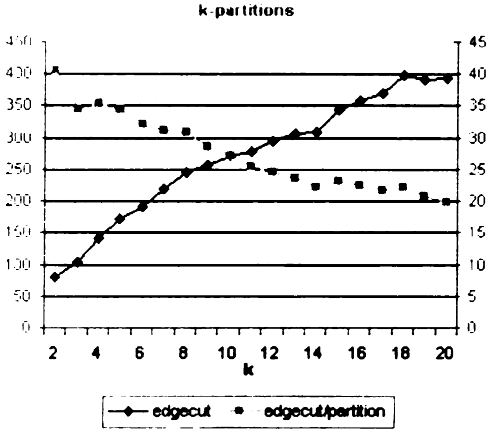


Figure 7-2. Finding an optimal number of partitions with METIS.

that were *cut*. Figure 7-1 shows that no optimal value for *k* was found with this methodology. Next the SCCs of the graph were found (Figure 7-2). A total of 208 SCCs were found: 192 with a single node each, 9 with two nodes each, two with six nodes each, two with five nodes each, one with 601 nodes, one with 65 nodes, and one with 4 nodes. The great majority of the graph is contained in a single well-connected SCC.

Key model elements: Two different methods were used to prioritize the model's variables in order of their likelihood of being a good drug target. The first method, called the standard method, prioritizes variables by their betweenness$_d$ centrality and whether or not the variable is an articulation point in the graph representation of the model. The articulation points of the graph along with those variables having a betweenness$_d$ score larger than the mean plus one standard deviation were selected as the best targets. So doing yielded 203 potential therapeutic targets. The second method, called the metric-based method, is given by Eq 7-1. It is a mathematical function that relates model sensitivity to a target's various centrality scores and the indicators from graph theory.

The variables with a predicted sensitivity above a single standard deviation threshold were selected as the best targets. This yielded 71 potential therapeutic targets.

$$\begin{aligned}Sm^*_i &= 0.32+[AP]_i(-0.11)\\ &+[bgcsd]_i(-0.51)+[sgcsd]_i(-0.86)\\ &+[UCC]_i(-0.30)+[UBC]_i{}^*(1.20)\end{aligned} \qquad (7\text{-}1)$$

Equation 7-1. Calculation of Sm*$_i$, the predicted average-fat-mass sensitivity to changes in i. AP = [0; variable I is not an articulation point, 1; otherwise], bgcsd = group directed-betweenness centrality* score for a bridge incident to variable i, sgcsd = group directed-betweenness centrality* score for a span incident to variable i, UCC’ = normalized undirected closeness centrality, UBC’ = normalized betweenness centrality. *[65].

Coverage of known targets for obesity: A database of known targets for obesity was compiled (57 targets) and those targets that are represented by variables in the OPL selected to validate the predicted targets (26 targets). The coverage of this modeled subset of known obesity targets was determined firstly, by whether they are predicted by the heuristic and secondly, by whether one of their neighbors in the graph is predicted. The targets selected by the standard method uniformly covered almost all of the known and currently investigated targets for controlling obesity, Figure 7-3a. The targets selected by the metric-based method covered many of the on-market targets and fewer of the targets undergoing early stage clinical validation, Figure 7-3b.

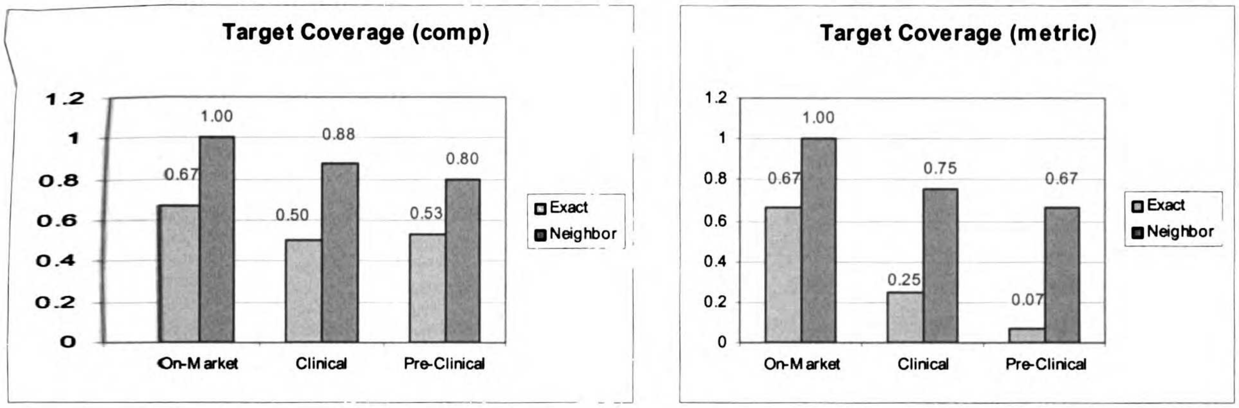


Figure 7-3. Fraction of known obesity drug targets predicted exactly and including graph neighbors of predicted targets, using the Standard method (a) and the Metric-based method (b).

Metrics that reveal likelihood of high biomarker sensitivity: A fraction of the targets highlighted by both methods were selected randomly for *in silico* validation. Parametric experiments were done with the OPL for each of these potential therapeutic targets. The indicators for disease progression used to validate each target were: maximum % of body-weight lost and gained for each experiment, the sensitivity of average-fat-mass to experiment parameters (Sm), and the sensitivity of rate-of-stored-triglyceride-change to experiment parameters (Sr).

Results from standard method: *Sm* positively correlates with the magnitudes of % weight gain and % weight loss, having R^2 values of 0.630 and 0.628 respectively; *Sr* positively correlates with the magnitudes of % weight gain and % weight loss, having R^2 values of 0.129 and 0.182 respectively; *Sm* = 0.162 is the lower threshold for % weight loss 5% and greater according to the linear equation fitting *Sm* to % weight loss, Eq 7-2; the fraction of *Sm* greater than or equal to 0.162 increases for larger values of betweenness$_d$ centrality, Figure 7-4a; the fraction of *Sm* greater than or equal to 0.162 increases for smaller values of closeness$_u$ centrality, Figure 7-4b.

Results from metric-based method: *Sm* positively correlates with the magnitudes of % weight gain and % weight loss, having R^2 values of 0.457 and 0.431 respectively; *Sr* positively correlates with the magnitudes of % weight gain and % weight loss, having R^2 values of 0.138 and 0.039 respectively; *Sm* = 0.22 is the lower threshold for % weight loss 5% and greater according to the linear equation fitting *Sm* to % weight loss, Eq 7-3; the fraction of *Sm* greater than or equal to 0.22 increases for larger values of the calculated metric, Figure 7-4c. Table 7-1 summarizes PhysioLab™ parametric experiments for both target sets.

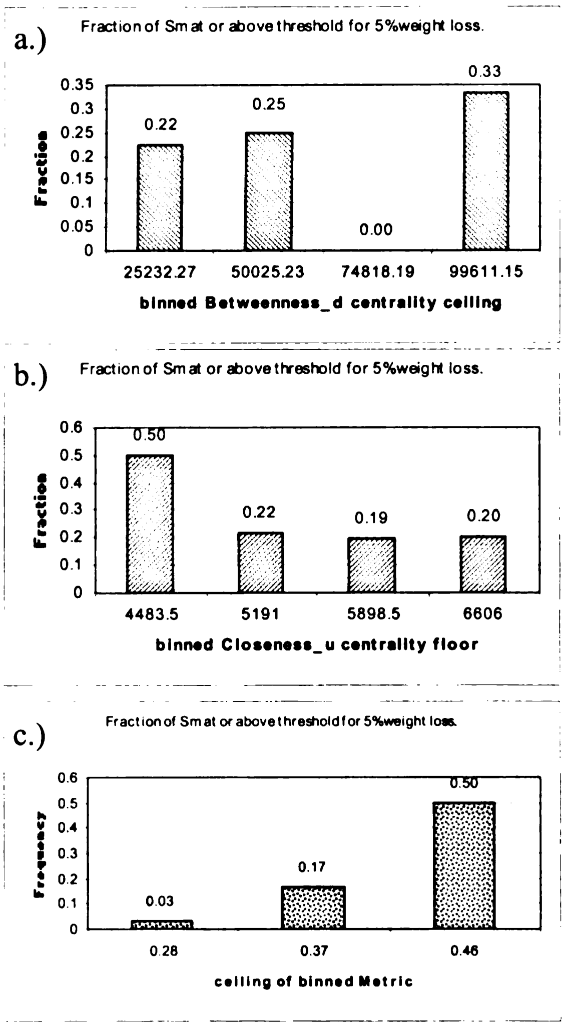


Figure 7-4. Increasing likelihood of observing a significant sensitivity (Sm) with a.) increasing betweenness and b.) decreasing closeness centrality, and c.) increasing modeled-metric.

$$\% \text{ Weight loss} = 0.294\,Sm + 0.003 \qquad (7\text{-}2)$$

Equation 7-2. Model of % weight loss relation to Sm from Standard Method's experiments.

$$\% \text{ Weight loss} = 0.202\,Sm + 0.006 \qquad (7\text{-}3)$$

Equation 7-3. Model of % weight loss relation to Sm from Metric-based Method's experiments.

$$\% \text{ Weight loss} = 0.1493\,Sr + 0.0086 \qquad (7\text{-}4)$$

Equation 7-4. Model of % weight loss relation to Sr from Standard Method's experiments.

$$\% \text{ Weight loss} = 0.055\,Sr + 0.0096 \qquad (7\text{-}5)$$

Equation 7-5. Model of % weight loss relation to Sr from Metric-based Method's experiments.

Table 7-1. Parametric experiments' results sorted by decreasing fraction of weight lost (||loss||) vs placebo. Experiments with significant (>5%) weight loss colored grey. Other experiments with significant (>5%) weight gain (||gain||) or significant (>threshold for each prediction method: 0.162 & 0.22) average-fat-mass sensitivity (Sm) colored grey.

Experiment results for Standard-method's predictions.

	||loss||	||gain||	||Sm||
1	30.8[illegible]	21.04%	0.45
2	28.[illegible]	18.14%	0.93
3	20.3[illegible]	21.04%	0.52
4	20.3[illegible]	19.51%	0.27
5	[illegible]	2.31%	0.17
6	11.2[illegible]	26.41%	0.54
7	11.1[illegible]	7.30%	0.31
8	11.0[illegible]	0.00%	0.03
9	11.0[illegible]	0.00%	0.03
10	10.6[illegible]	1.45%	0.07
11	10.6[illegible]	10.29%	0.21
12	8.2[illegible]	4.43%	0.10
13	7.9[illegible]	21.38%	0.10
14	7.2[illegible]	21.58%	0.26
15	6.7[illegible]	13.43%	0.47
16	6.2[illegible]	6.69%	0.03
17	6.2[illegible]	0.00%	0.09
18	[illegible]	6.62%	0.07
19	[illegible]	16.72%	0.30
20	4.85%	[illegible]	0.13
21	4.84%	[illegible]	0.07
22	4.82%	[illegible]	0.12
23	4.46%	[illegible]	0.11
24	3.83%	4.47%	0.08
25	3.39%	[illegible]	[illegible]
26	3.39%	0.00%	0.08
27	3.36%	[illegible]	[illegible]
28	1.57%	[illegible]	[illegible]
29	1.43%	2.85%	0.07
30	1.34%	1.67%	0.02
31	1.10%	[illegible]	0.04
32	1.05%	[illegible]	0.05
33	1.04%	1.06%	0.04
34	1.04%	1.06%	0.04
35	1.00%	0.58%	0.02
36	0.91%	0.46%	0.03
37	0.74%	7.21%	0.04
38	0.72%	0.34%	0.02
39	0.64%	0.52%	0.01
40	0.61%	2.93%	0.01
41	0.61%	0.01%	0.01
42	0.56%	0.00%	0.00

Experiment results for Metric-based-method's predictions.

Name	||loss||	||gain||	||Sm||
1	11.2[illegible]	26.41%	0.54
2	11.1[illegible]	7.30%	0.31
3	11.0[illegible]	0.00%	0.03
4	10.9[illegible]	8.06%	0.05
5	10.6[illegible]	10.29%	0.21
6	[illegible]	6.69%	0.03
7	4.46%	[illegible]	0.11
8	3.77%	1.67%	0.03
9	3.58%	3.46%	[illegible]
10	1.82%	0.09%	0.04
11	1.67%	0.43%	0.03
12	1.57%	[illegible]	0.13
13	1.43%	2.85%	0.07
14	1.25%	1.87%	0.04
15	0.90%	4.35%	0.08
16	0.74%	7.21%	0.04
17	0.67%	1.20%	0.04
18	0.61%	2.93%	0.01
19	0.61%	0.01%	0.01
20	0.56%	0.00%	0.00
21	0.56%	0.01%	0.00
22	0.56%	0.04%	0.00
23	0.52%	[illegible]	0.04
24	0.42%	0.23%	0.01
25	0.35%	0.50%	0.00
26	0.30%	0.33%	0.03
27	0.30%	0.55%	0.01
28	0.29%	2.01%	0.01
29	0.25%	0.92%	0.01
30	0.18%	0.24%	0.00
31	0.09%	0.74%	0.01
32	0.09%	0.01%	0.00
33	0.06%	0.02%	0.00
34	0.01%	4.88%	0.01
35	0.01%	0.05%	0.00
36	0.00%	0.02%	0.00
37	0.00%	0.02%	0.00
38	0.00%	1.21%	0.04
39	0.00%	8.85%	0.01

43	0.56%	0.01%	0.00
44	0.50%	4.88%	0.02
45	0.40%	0.02%	0.00
46	0.36%	0.18%	0.00
47	0.25%	0.92%	0.01
48	0.16%	0.70%	0.02
49	0.16%	8.87%	0.07
50	0.16%	0.75%	0.01
51	0.15%	0.34%	0.01
52	0.13%	0.81%	0.01
53	0.11%	0.13%	0.01
54	0.10%	0.46%	0.01
55	0.06%	0.04%	0.00
56	0.06%	0.67%	0.01
57	0.05%	0.16%	0.00
58	0.04%	[illegible]	0.06
59	0.03%	0.01%	0.00
60	0.02%	0.66%	0.00
61	0.01%	0.02%	0.00
62	0.01%	0.02%	0.00
63	0.01%	0.01%	0.00
64	0.01%	0.01%	0.00
65	0.01%	[illegible]	0.01
66	0.01%	0.06%	0.00
67	0.00%	0.03%	0.00
68	0.00%	[illegible]	0.01
69	0.00%	[illegible]	[illegible]
70	0.00%	[illegible]	0.01

DISCUSSION

I have studied the topological organization of an obesity model and prioritized model elements as potential drug targets. Experiments have been formulated to test the predictive value of each of the heuristic's internal metrics. For the set of predicted therapeutic targets, I show how many are already known and in clinical trials. The process began by determining whether or not the model could be organized into a modular structure. No best number of partitions was found. The density of relationships between model variables may mean that there is no modular structure to be found. The result that the majority of the model's variables are contained in a single SCC seems to indicate the difficulty in finding a modular structure.

Alternatively, there might be some modularity, but the METIS partitioner is not the best tool to find it. METIS has a partition-balancing algorithm that strives to make the number of nodes in each partition equal. This is desired for partitioning work for parallel computing, but there is no reason to believe biological modules are balanced. An unbalanced partitioning algorithm, like recursive spectral partitioning, might have more success in finding modular structure, as it finds the partitioning with the minimum number of cut edges regardless of the size of the resulting partitions.

The next step in the heuristic is prioritizing nodes according to information from the topological organization phase and node centrality calculations. As stated earlier there are two techniques being compared: a standard and metric-based method. The metric-based method results in a smaller number of predicted and validated targets compared with the standard method. The targets predicted by each method were first validated by assessing their coverage of drug targets associated with pre-clinical, clinical, and on-market weight control drugs. It is important to note that a fraction of the known obesity targets could not be directly modeled by the OPL and were not included when assessing the predictions. The coverage determined by exact matches was approximately 55% for all target pools using the standard method. When the coverage rule was relaxed to include neighbors of predicted target, the coverage was complete for all known, modelable drug targets. The reason for including a predicted target's neighbors in the network when assessing a match to known drug targets was that, when it came time to actually implement an intervention in the OPL for a specific predicted target, the neighbors were more often than not adjusted as well. This was due to the inconsistent mutability of certain parameters across all OPL variables.

The coverage results for predictions from the metric based methods show the same improvement when neighbors are considered, but the coverage seems to improve for target populations toward the end of the development pipeline. Two possible reasons for this bias are: the OPL incorporates more information about on-market and better-known targets and their associated pathways, and/or the bias is expected because the aim of the drug development process is to enrich the pool of target candidates as they move towards FDA approval.

The aim of each method is to predict targets to which the model is highly sensitive. High sensitivity is assumed to be a property of quality targets. Both sensitivity prediction methods have been shown to cover a majority of the known targets for obesity. The OPL can be used directly to assess how well the methods predict the actual sensitivity values. For the standard method, mainly betweenness$_d$ and closeness$_u$ were used to prioritize targets for experimental validation. Figures 7-4a and 7-4b show that better centrality values are predictive of *good* sensitivities—*good* sensitivities are those above the 5% weight loss threshold. Nineteen target interventions showed weight loss of approximately 5% and higher. There are four more interventions with high *Sm* values but no significant *in silico* weight loss. The latter may indicate target interventions deserving of further investigation and a broader exploration of parametric space, because the weight loss trend (indicated by Sm) is significant. The metric calculated in the metric-based method is also shown to be predictive of targets with good sensitivity values, Figure 7-4c.

There are six interventions with significant *in silico* weight loss, and one other with a promising *Sm* value.

Whether to use the metric based method or not may be moot, because its predictive model requires a training set of sensitivity values. To get such a set one must perform several *in silico* parametric experiments. If the standard method is used to prioritize the model's variables prior to a random selection of variables to test, then it is quite likely that many good targets will be found during the process. It is questionable whether researchers will be willing to proceed to the construction of a model, such as the one used in the metric-based method, or simply further validate the good targets found by the standard method.

The approach used here demonstrates that a system's topology has considerable value even in the absence of quantitative information. In general, a mathematical model's variable interdependencies are represented by a graph, and this graph is organized into subsystems with a minimal amount of interaction between them. This minimal set is an initial choice for critical interactions to the system's overall behavior. An independent but complementary ranking of variables by their centrality is used to prioritize this linking set and every other model variable. Analyses of this type yield useful insights (have high value) and are faster than more robust quantitative analyses. Usually robust methods, such as classical sensitivity analyses, require a screening phase to focus the analysis on a smaller, manageable set of variables. This screening process is typically subjective. However topological methods, such as those presented here, can provide objective screens for later detailed analyses, including classical sensitivity analysis. Future computational advances may allow similar analysis using graphs that include quantitative and kinetic information, or even direct sensitivity calculations with acceptable temporal costs. For now the main use for the approaches described here will be to extract further information from qualitative systems. Such systems are plentiful, whereas large fully parameterized quantitative models are more rare.

Much can be learned about a system by analysis of its topology, but experimental validation of apparent insights is an essential next step. Methods such as these are not the last step in target selection. However they should be one of the first steps in prioritization of system elements for further detailed validation, including *in silico* validation. More work is required to address integration of available kinetic information into graph analysis methodologies, and to better understand the benefits to comparative studies between different systems.

SYSTEMS & METHODS

Graph mapping: The OPL version 3, from Entelos® is visually transcribed into a Microsoft® Access™ database for preprocessing. Each variable's name and type is recorded and given a new integer node id. Each arc representing a relationship between variables is recorded as a unique integer edge id. For arcs that are modified by another variable, the arc is represented by a node id and a new edge is recorded for each variable modifying that arc. This process results in a list of directed edges comprising an unique edge id, a node id from the node incident to the beginning of the directed edge, and another node id from the node incident to the end of the directed edge; a list of nodes comprising an unique node id, a unique node label, the node's type: function node or state node, and the node's origin (obesity model or new). It is important to note that there are many variables in the OPL that are not immediately visible to a user. These were not included in the transcribed graph. This exclusion underscores the point that there are multiple options for transcribing a quantitative model to a graph representation.

Select active edges: Each node in the graph represented a variable in the model that has an active and inactive state during the simulation. This state of a variable in the simulation is recorded with each node representing it. All active nodes are written to a comma-delimited flat file with the node's id and name. Each edge incident to 2 active nodes are considered active and written to a comma-delimited flat file with the node ids incident to the beginning and end of the directed edge, and the edge's id.

OPL experiments: The patient profile used for all experiments is an equilibrated 100kg individual with light activity. For each target to be evaluated a parametric experiment is executed. Parametric experiments are a batch of experiments, each having the parameter(s) of interest set to a range of values according to a linear or log interpolation between the initial and final values for the parameter(s). The experiments simulated the patient's physiology over a one-year interval, and data is sampled at 1440-second intervals. Saved experimental data is used to calculate the maximum fraction of weight loss and gained compared to the weight of a control patient after a one-year interval. Sensitivity of various biomarkers to the intervention is calculated as the ratio of the difference in the natural logarithm of the biomarker's value and the difference in the natural logarithm of the changing parameter's value.

Chapter 8: Software Tools Developed For Topological Analyses

OVERVIEW

Knowledge of a system's organization aids in the search for therapeutic targets that maximally affects the system. The topology of a system can be used to reveal the extent to which it is organized in modular subsystems and used to prioritize its elements according to their potential to be good therapeutic targets. This software developed identifies modularity of a system by implementing various graph partitioning and clustering algorithms. It also assesses the criticality of system elements with additional centrality calculations. The application is written in Python/Tkinter and displays analyses results in a multifunctional graphical user interface.

INTRODUCTION

To facilitate the topological analyses, an application has been developed in Python™. The aim of the analyses is the identification of modularity. This knowledge is expected to uniquely aid the understanding of complicated biological systems and the prediction and prioritization of therapeutic targets. To better understand results, a network visualization facility was added to the initial application embodiment. This graphical interface was written in Tkinter™ and provides an additional interface to the network analysis application. This application has been applied to several network models of biological interaction: molecular interactions involved in blood coagulation, metabolic and signaling interactions involved in obesity, and metabolic interactions conducted by *Esherichia coli*. The methodology involved in these analyses is inspired by prior graph theoretical and social network analytical research.

PROGRAM DESCRIPTION

Input to this application is a network model represented in a graph format. The application can accept input from a command line or via its graphical user interface (GUI). Once an input file is specified, its graph can be viewed in the GUI and/or a battery of graph analyses can be conducted. The graph layout method is not finalized, but currently a multidimensional scaling (MDS) approach and a mass-spring-system relaxation approach are available in the GUI. Briefly, the network analyses are of two related types: modularity assessments and critical-node prioritization-metric calculations. Results are then viewable by projection onto the currently displayed graph. Individual results can be viewed alone or overlaid and the graph layout can be manipulated in the GUI. Elements can be removed from the system and reanalysis initiated for deeper investigation. All results are persistent on disk as readable files.

Graph Layout

The graph is initially displayed with nodes positioned randomly. Two layout options are available: MDS or mass-spring based. The MDS algorithm calculates the eigenvectors and eigenvalues for the graph's distance matrix, and then selects the two eigenvectors associated with the two largest eigenvalues as principal axes. The mass-spring algorithm treats each node as equal masses with a negative attraction force between them; graph edges are treated as springs with equal relaxation lengths. During the relaxation process the system's kinetic parameters—spring relaxation length, mass repulsion, spring constant—can be updated manually. The resulting balance of mass repulsion and spring contraction usually results in a visually pleasing layout.

Additional graph visualization and modification tools include: a magnification lens, node selection, node label toggling, a node activation-toggling menu, and an analysis results table.

Analyses

Several analyses are available from the GUI, or command line. Their results are automatically stored in individual files and sent to the GUI for visualization as reflected in node size and color, and edge color. In order of batch execution they are as follows: identifying connected components, strongly connected components, articulation points, bridges and spans; calculation of betweenness and closeness centrality values for each node; and hierarchical node clustering. Currently, there are two additional analyses that are completed but not visualized. Partitioning the graph into k parts for a range of user-selected k's is done by an extension to METIS™. Similarly, spectral graph partitioning is conducted with custom designed Python™ code. After each type of partitioning, nodes on the edge of each partition are identified.

Performance

Table 8-1. Time taken to perform all analyses for several systems, and metadata for each system.

Graph	**Diameter {D(U)†}**	**Geodesics**	**Nodes**	**Time (sec)‡**
Blood	7 (9)	124 (1402)	28 (31)	0.05 (0.25)
Ob1	21 (12)	110331 (407520)	289 (381)	*
Ob3	32 (15)	993681 (5092384)	837 (902)	937 (2043)
E.coli	19 (*)	4417426 (52583606)	1121 (1502)	2143 (*)

†The values without parentheses are those from the analysis of a directed graph and those in parentheses are for the undirected graph.

‡The time posted is that taken to complete all analysis calculations except graph partitioning and clustering.

*The batch analysis job failed when computer memory demands increased, requiring individual reapplication of each subanalysis. This prevents proper timing of the batch job.

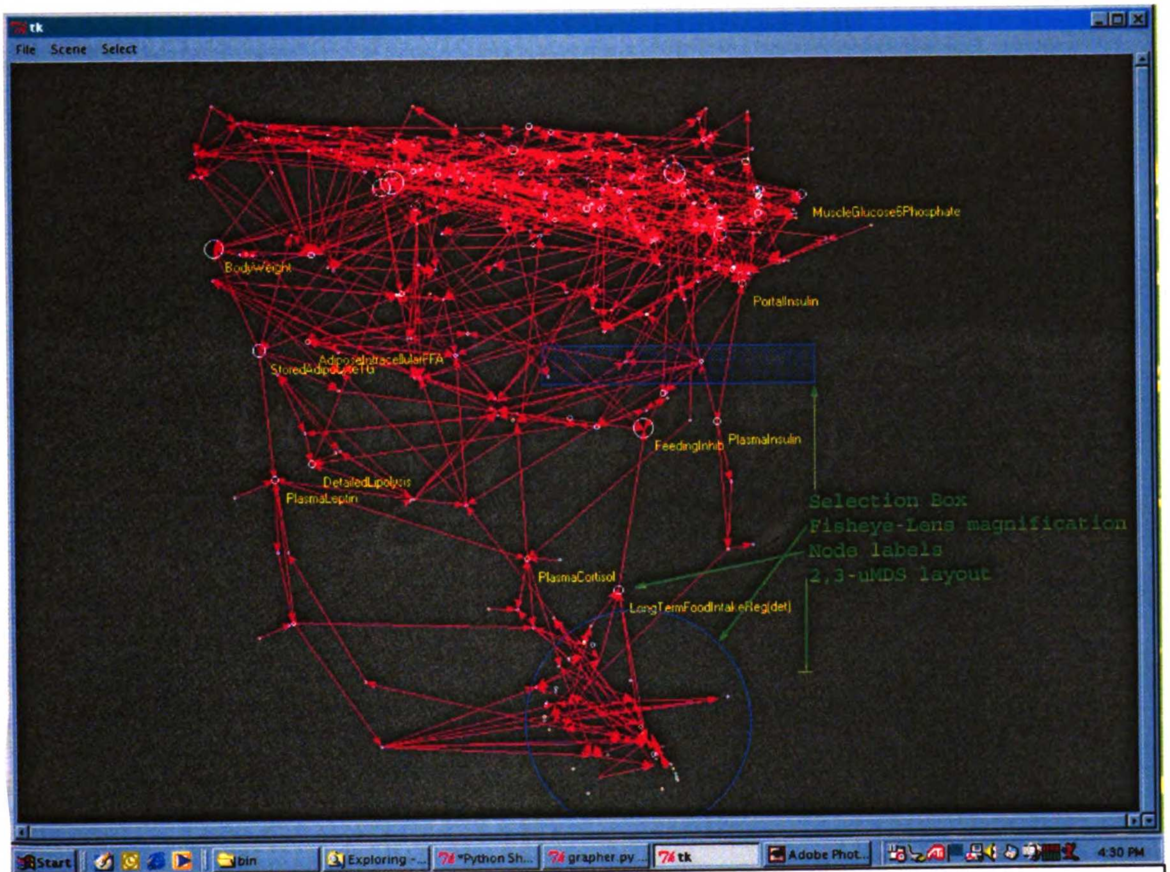
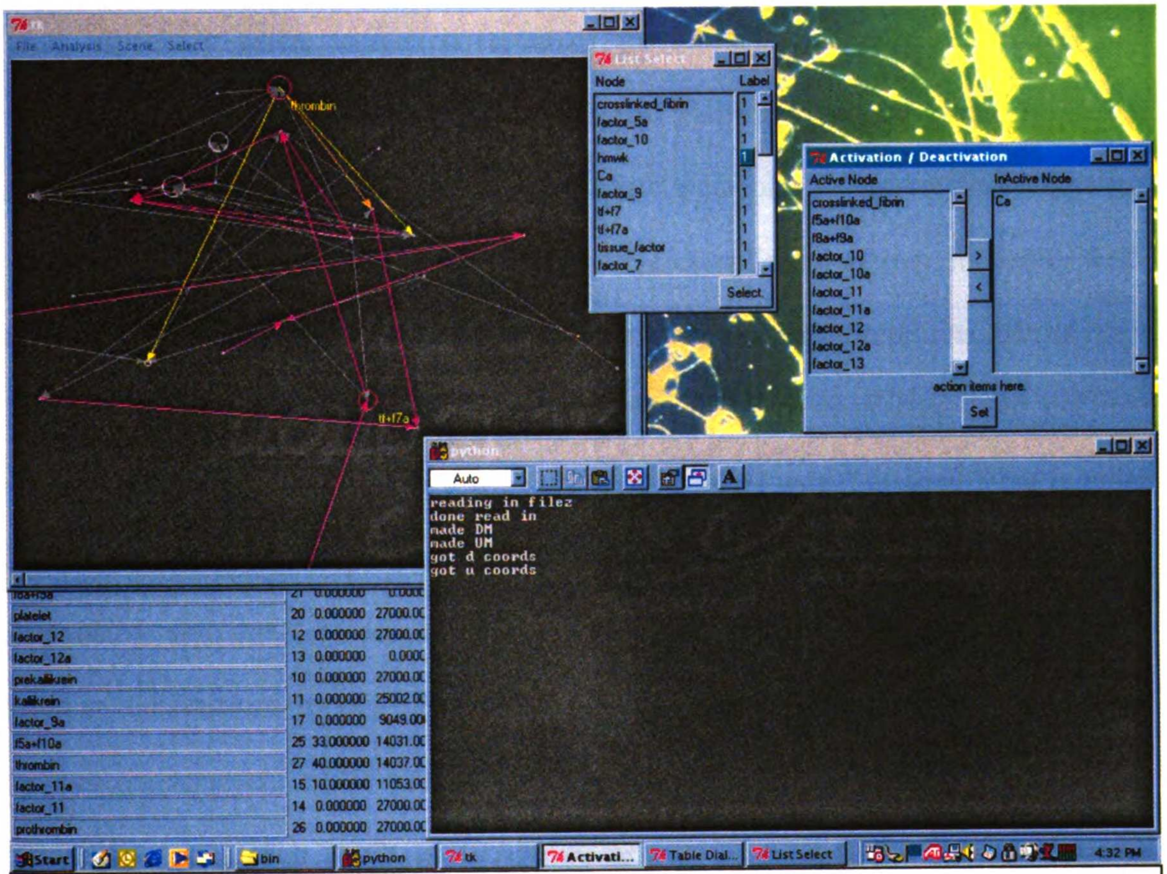


Figure 8-1. Screenshot of GUI showing selection box, fisheye zoom, data representation, and graph layout.



```
reading in filez
done read in
made DM
made UM
got d coords
got u coords
```

f8a+f9a	21	0.000000	0.000C
platelet	20	0.000000	27000.0C
factor_12	12	0.000000	27000.0C
factor_12a	13	0.000000	0.000C
prekallikrein	10	0.000000	27000.0C
kallikrein	11	0.000000	25002.0C
factor_9a	17	0.000000	9049.00
f5a+f10a	25	33.000000	14031.0C
thrombin	27	40.000000	14037.0C
factor_11a	15	10.000000	11053.0C
factor_11	14	0.000000	27000.0C
prothrombin	26	0.000000	27000.0C

Node	Label
crosslinked_fibrin	1
factor_5a	1
factor_10	1
hmwk	1
Ca	1
factor_9	1
tf+f7	1
tf+f7a	1
tissue_factor	1
factor_7	1

Active Node	InActive Node
crosslinked_fibrin	Ca
f5a+f10a	
f8a+f9a	
factor_10	
factor_10a	
factor_11	
factor_11a	
factor_12	
factor_12a	
factor_13	

Figure 8-2. Screenshot showing GUI, the status console, a partial view of the imported-data table, and two node selection boxes.

Chapter 9: Test Case on Public Data: *E. coli* Metabolism

OVERVIEW

For computational disease models, a prioritization of potential therapeutic targets would support clinical and computational target validation processes in pharmaceutical R&D, especially where validation resources are limited. In order to discover key molecules and modular organization in biological pathways, several graph theoretical concepts are applied to a model of *Escherichia coli* metabolism. The metabolic network of *E. coli* was chosen because it represents a large complicated system, modeling well-studied biology. This type of network consists of only enzymatic reactions, but its large size may be typical of the type of models that can be reconstructed from publicly available data. Also, as *E. coli* is a widely used biological model system, there is more information with which to validate the techniques described previously. Graph partitioning and strongly connected component identification are used to discover modular substructure in this metabolic network. The identified strongly connected components represent cyclic subpathways common to multiple well-studied metabolic pathways. Articulation points and centrality metrics are used to predict and prioritize critical molecules. Betweenness and closeness centrality are shown to be reliable metrics for prioritizing key metabolites and essential enzymes, betweenness being the most reliable. One question that consistently arises is, "How do errors in the network structure affect the quality of these topological methods?" For this system, missing information is shown to negatively affect the prioritizations for essential enzymes, but not for key metabolites.

INTRODUCTION

An organism's metabolic reaction network may have thousands of metabolites and enzymes. The complexity of interactions and dependencies between known molecules, combined with several potentially unknown molecules, frustrates rational methods for predicting the effects of perturbing the system by means of molecular intervention. It is quite difficult to accurately predict the molecules within a system which, when modulated, will cause the most significant change in system behavior. This report describes methods for objectively obtaining information about a system's modularity and critical interactions. Information about critical elements and interactions can be gathered from quantitative models with methods such as sensitivity analysis, but the time required to do the analysis may be impractical when models are large. Also, topological knowledge of the system's organization is not provided by normal application of sensitivity analysis (SA).

The majority of signaling and metabolic pathway information is semi-qualitative, roughly illustrating intermolecular relationships[58]. The size of the networks and their qualitative nature puts them outside the reach of robust SA. Hence there is utility in having a generalized qualitative SA methodology that requires only a network's topological information to indicate critical targets. Such information, when readily obtained, would aid decision making about which targets to avoid and which to focus on.

Although the majority of network information is qualitative, research is underway in quantitative whole-system modeling[64, 77, 87, 88]. SA would be ideal for finding and prioritizing targets. However, consider a system with 1000 variables with 5 states each and 50 marker variables. An analysis assessing the effects of each variable on each marker would require 250,000 simulations. Assuming a simulation requires 4 hours to complete, this sensitivity analysis would require more than a century. Still, such an analysis may be insufficient if there are important synergistic effects between variables. A pseudo-SA would be ideal for filtering a quantitative model to identify those features that are likely to be most critical, prior to quantitative-SA. In fact, pre-SA filtering is done in practice but in a subjective manner reflecting individual bias.

For a system modeled as a flow network, finding its bottlenecks can approximate SA. Network bottlenecks can be found by graph partitioning methods[68, 69]. An optimal partitioning of a graph's nodes into separate groups is assumed to be one that results in the fewest number of edges spanning different partitions, or minimally weighted edges. This set of spanning edges is called the cutset, and information flow through a network is limited by the cutset with the smallest capacity. Given the critical, integrative positioning of the cutset, many pathways between nodes in the network will include or cross the cutset. A measure of this property is betweenness centrality, which is used in Social Network Analysis[65]. Betweenness centrality can be used to prioritize nodes incident to a cutset. When an optimal partitioning is not found, modularity within the system can be approximated by the connected and strongly connected components of the network. A network's articulation points, bridges, and nodes that have high betweenness centrality scores will identify graph elements that are more likely to be in the minimal capacity cutset. In a metabolic network, where enzymes and metabolites are modeled as nodes connected by edges linking substrates to enzymes and enzymes to products, an ideal cutset and the nodes that have high centrality scores may represent key metabolites and essential enzymes.

Modules in biology have been described as groups of interacting elements, isolated physically or chemically, which perform a specific function[29]. Yi et al provides a model of bacterial chemotaxis that may classify as a fitting example of modularity[37]. This complete and parametrically robust system comprises seven proteins exhibiting integral feedback control. The system's input is the concentration of a stimulatory ligand, and the output is the level of phosphorylation for a certain protein that affects cell chemotaxis. The functions that modules provide can be integrated to achieve complicated and adaptable phenotypes. Consequently, identification of modules is expected to be useful for therapeutic target discovery.

In the following sections I describe analyses of a graph model of Escherichia coli metabolism. Modular substructure and key molecules are highlighted. Four centrality metrics were validated according to their prioritization of metabolites representing the common biosynthetic source of all cell materials, and discrimination between essential and nonessential enzymes. The effect on each metric's prioritization of removing common metabolites from the graph is also illustrated.

RESULTS

The reactions catalyzed in *E. coli* metabolism were translated into a graph for topological analysis. The number of enzymes was 575. The number of metabolites was 879: 345 inputs, 353 outputs, and 181 recycled. The number of recycled metabolites is a count of metabolites listed in the ENZYME database that are both substrates and products in a single reaction or in multiple reactions. It is important to note that several reactions in the database are conditionally reversible, but not recorded as such. For our analyses that use directed edges (SCC, and directed centrality metrics), the reaction coordinate is assumed to be as recorded in the database and is not treated as being reversible, an important point when interpreting these analyses. Additionally, several metabolites have inconsistent capitalization from reaction to reaction, but this situation was accounted for when parsing the database.

A flow diagram illustrating all phases of the analysis is shown in Figure 9-6, found in the following Methods section.

The system of metabolic reactions is modeled as a graph with 1454 nodes and

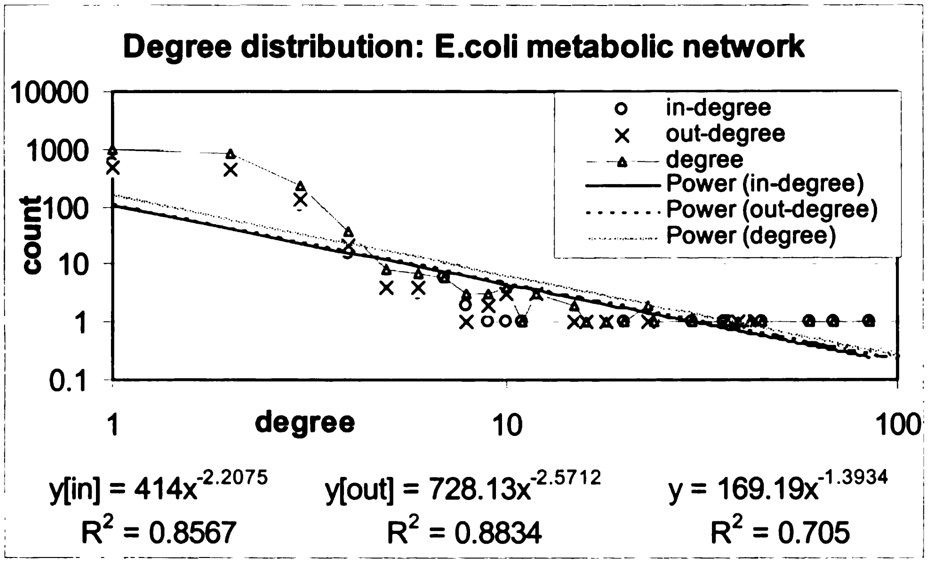


Figure 9-1. Scale free degree distribution, with fitted power functions.

2361 edges. The *degree* distribution (Figure 9-1) is scale free as previously reported—degree being the number of edges incident to a node, or the number of neighboring nodes. The number of components in the initial graph is 17, and 16 describe single enzyme reactions. While most are simple reactions: component 1 [reduced glutaredoxin, arsenate, oxidized glutaredoxin, EC 1.97.1.5, arsenite] is an arsenical pump adapter which provides resistance to arsenate toxins; component 2 [EC 2.1.1.63, protein-l-cysteine, DNA (without 6-o-methylguanine), protein s-methyl-l-cysteine, DNA (containing 6-o-methylguanine)] represents repair of alkylated DNA which provides resistance to alkylating agents in *E. coli.*

Table 9-1. Articulation points from the global graph with their centralities: Directed Betweenness Centrality (DBC), Undirected Betweenness Centrality (UBC), Directed Closeness Centrality (DCC), and Undirected Closeness Centrality (UCC). The sort is based on decreasing DBC, where larger values are preferred. 30 are shown here, but the full listing is provided in Table A-8 (see Appendix A).

Name	UBC	DBC	UCC	DCC
h(2)o	926560	329597	61195	207716
diphosphate	252937	187832	61973	207882
phosphate	233849	152231	61873	207544
atp	463531	115640	61509	209664
nadh	62298	63737	62715	208138
coa	114334	54026	62321	208866
nadph	51049	40018	62911	208458
nad(+)	116976	36014	62139	208960
pyruvate	61301	32317	62387	208674
l-glutamate	51236	30437	62307	209038
nad(p)h	22005	18938	63759	209388
alpha-d-glucose 1-phosphate	6528	17054	63755	209308
d-ribose 5-phosphate	11430	12942	63195	209020
o(2)	26133	7105	63473	211192
l-serine	6126	5210	62865	209456
3.5.1.18	12745	4063	62554	208601
glutathione	18642	4000	62883	209192
udp-glucose	16573	3812	63317	211082
2.3.1.1	3884	3481	63440	209699
4.2.1.46	7408	3076	62572	211971
10-formyltetrahydrofolate	5493	2944	63927	209438
l-homocysteine	4548	2877	63663	209478
3.2.1.31	8383	2739	62576	208601
3.1.3.27	18500	2667	62324	208605
2.7.1.16	16716	1800	62880	206567
3.1.1.61	14481	1795	62540	206614
imp	470	1347	63035	210698
d-ribulose 5-phosphate	2824	1123	63747	208479
l-ribulose 5-phosphate	5576	901	64271	206468
2.7.1.21	3210	899	62882	208562

The main component is analyzed for modularity and node centrality. This analysis is referred to as the *global* analysis in tables and figures. For each centrality prioritization, the best scores are those above a threshold calculated for each metric. The threshold for closeness centrality is 1.5 standard deviations less than the mean and 5 standard deviations greater than the mean for betweenness centrality. The best scoring molecules for each metric are listed in Table A-5 (see Appendix A). The molecules that are the graph's articulation points are shown in **Table 9-1** along with their centrality scores.

The graph's strongly connected components (SCCs) have been found. After adjusting for SCCs that contain single nodes, there are 2 SCCs; one containing 2 nodes [EC 5.4.2.1 & 2_3-diphosphoglycerate] and another containing 528 nodes. The metabolites known to be important for *E. coli* metabolism are shown in **Table** 9-2. Those with centrality scores that exceeded the significance threshold are indicated. The relevant scores for those identified are detailed in Table 9-3.

The quality of the prioritization given by each centrality metric was measured with data on essential genes from the PEC database. The database contained 4411 gene / phenotype pairs. The phenotypes were labeled as essential, nonessential, and unknown. The *Escherichia coli* K-12 MG1655 complete genome from GenBank (http://www.ncbi.nlm.nih.gov/Genbank/index.html) was parsed for gene names and EC numbers. So doing resulted in 748 gene, EC records. These two lists were joined by gene name yielding 748 records of gene, EC number, and phenotype. 575 of these enzymes matched genes in the model of *E. coli* metabolism. Of the original 748 enzyme phenotypes, 85 were essential, 631 were nonessential, and the rest were unknown. At least 11.4% of *E. coli* enzymes were found to be essential for its growth.

Table 9-2. Sixteen key metabolites for *E. coli* metabolism. A + indicates that the cooresponding meatabolite had a significant score from one or more prioritization metrics; a – indicates otherwise. An EC number indicates that the specific metabolite's score was not deemed significant, but the score of an enzyme linked to it was. Results are shown for analysis of the global and filtered graph.

Metabolites	**global**	**filtered**
Glutamate	+	+
Pyruvate	+	+
CoA	+	+
2-oxoglutarate	+	+
Glutamine	+	+
Aspartate	6.3.1.1	+
Acetyl CoA	+	+
Phosphoribosyl PP	-	-
Tetrahydrofolate	-	-
Succinate	6.2.1.5	+
3-phosphoglycerate	-	-
Serine	+	+
Oxoisovalerate[oxobutanoate]	-	-
Anthranilate	-	4.1.3.27
Chorismate	-	4.1.3.27
Valine		2.6.1.66

Table 9-3. The key metabolites that had significant indicators from the global and filtered analysis are shown below with the significant scores for each metric. Each metric's significance threshold is indicated above the metric's name. Nodes that are articulation point s (APs) are indicated. UCC, UBC, & DBC are defined in Table 9-1.

Global detail	+(yes) or –(no)	-1.5stdev	+5stdev	+5stdev
Name	AP	UCC	UBC	DBC
l-glutamate	+	62307	51236.23	30437.63
Pyruvate	+	62387	61301.99	32317.05
CoA	+	62321	114334.1	54026.96
Acetyl-CoA			32577.53	18599.11
2-oxoglutarate				12354.89
l-glutamine				16938.82
6.3.1.1				19110.76
6.2.1.5		62356	20146.84	12481.01
l-serine	+			

Filtered detail	+(yes) or -(no)	-1.5stdev	+5stdev	+5stdev
Name	AP	UCC	UBC	DBC
l-glutamate	+	3107	98179.49	11833.04
Pyruvate	+	3103	85650.74	10129.41
CoA	+	3089	92856.25	17900.24
Acetyl-CoA	+	3351	27600.73	9059.963
2-oxoglutarate	+			3742.298
l-glutamine	+		21317.14	4013.411
l-aspartate	+		23718.10	
Succinate				8607.594
l-serine	+		21803.15	
4.1.3.27	+	3307	34683.58	
2.6.1.66				7138.754

The quality of the prioritization given by each centrality metric was evaluated by how well they discriminated between essential and nonessential genes. To measure the difference between scores of essential and nonessential genes, the discrete cumulative distribution function for each phenotype was plotted (Figure 9-2) over the range of each metric's scores. These functions were generated by plotting the fraction of genes, with a specific phenotype, having scores better than score σ_t for all observed σ in the range of scores. The difference between the essential gene and the nonessential gene distribution functions was also plotted. The Kolmogorov-Smirnov (K-S) test[89, 90] was used to assess the significance of the difference between the two distributions. The direction of shift between the distributions

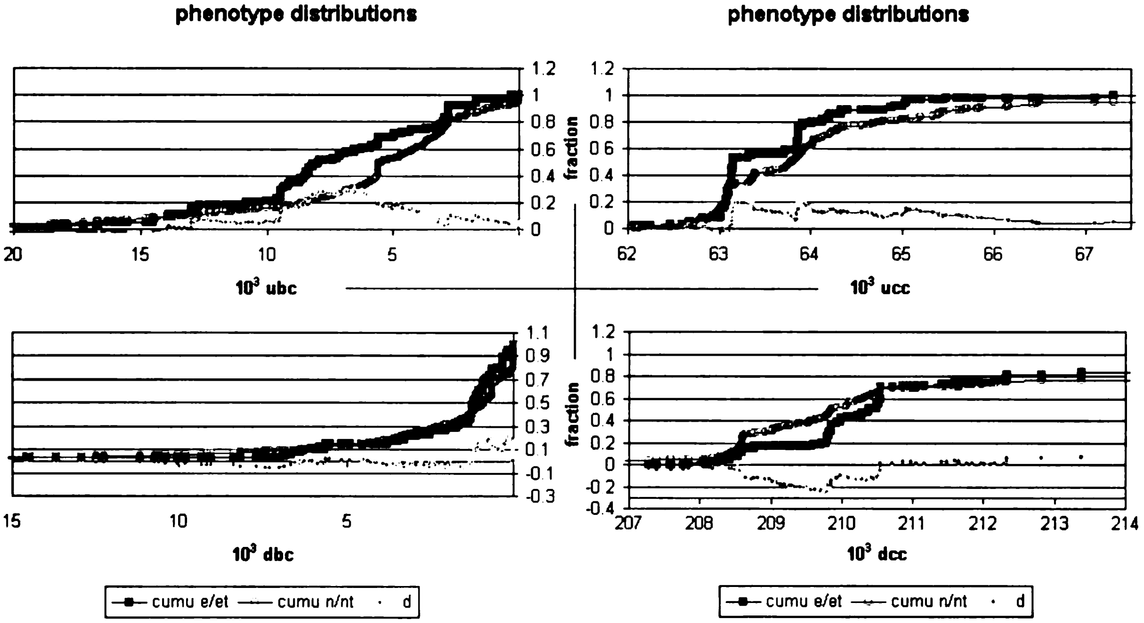


Figure 9-2. Distributions of centralities for essential and nonessential enzymes. The abscissas are oriented so that better centrality values are leftward. 'cumu e/et', 'cumu n/nt', and 'd' represent the cumulative fraction of essential and nonessential enzymes and the difference between the two distributions.

was measured by subtracting the median score of the nonessential gene distribution from the median score of the essential gene distribution, and the results are listed in **Table 9-4** along with a measure of confidence, the probability of observing a difference greater than the measured value, for each distribution difference.

Table 9-4. Distribution Statistics for Essential and Nonessential Enzymes. Values in parentheses are for the filtered graph others are for the global graph. D = maximum difference between essential and nonessential enzyme distributions. Q = probability of observing a difference greater than D. Ne = effective number of observations for the calculation of Q. S = shift of essential gene score distribution away from the nonessential gene score distribution. UCC, UBC, DCC & DBC are defined in Table 9-1.

metric	*D*	*Q*	*S*	*Ne*
ucc	0.22 (0.07)	0.01 (0.98)	-6.47 (+4.64)	57.7 (38
ubc	0.29 (0.26)	0.00 (0.00)	+7.83 (-7.18)	
dcc	0.07 (0.43)	0.93 (0.00)	+6.25 (-12.01)	
dbc	0.21 (0.44)	0.01 (0.00)	+5.75 (-1.10)	

The high degree of some common metabolites may skew centrality scores of other metabolites because of their higher likelihood to be on shortest paths through the graph. These shortest paths are used in calculating each metric. For this reason the common metabolites were removed and the graph was reanalyzed with the expectation of revealing other key metabolites. The common metabolites removed are: NAD^+, NADH, $NADP^+$, NADPH, NAD(P) $^+$, NAD(P)H, H_2O, CO_2, O_2, H_2O_2, Phosphate, AMP, Diphosphate, ATP, ADP, NH_3, oxidized Thioredoxin, reduced Thioredoxin, NH_4OH, DNA (containing 6-O-Methylguanine), DNA (without 6-O-Methylguanine), {Phosphate}(n), {Phosphate}(n-1), {Phosphate}(n+1), {Polyphosphate}(n), {Polyphosphate}(n-1), RNA 3'-terminal-phosphate, and RNA terminal- 2',3'-cyclic-phosphate. Pruning common metabolites and pendant nodes yields a graph having 799 nodes, 887 edges, and 23 components. Most components are small, i.e. two enzymes linked by a single metabolite. Three are larger and more complicated, the largest containing 604 nodes.

The largest component, containing 604 nodes and 806 edges, was subjected to the same battery of analyses as the global graph. These results are referenced to as the *filtered* results in tables and figures. The molecules with the best scores for each centrality metric are listed in Table A-6 (see Appendix A). The molecules that are the filtered graph's articulation points are listed in Table A-7 (see Appendix A) along with their centrality scores. The graph's strongly connected components have been found. Ignoring SCCs that contain single nodes, there are 6 SCCs containing: 2, 4, 4, 6, 33, and 85 nodes each.

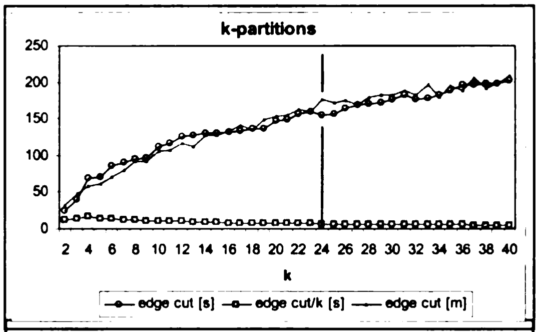


Figure 9-3. Finding an optimal partitioning. [s] denotes data from the spectral partitioner. [m] denotes data from the METIS partitioner.

The graph was partitioned into *k* subgraphs (*k* = 2-40) in order to identify any optimal partitioning. The partitioning algorithm used was based on ideal recursive octasection. An optimal value of *k* was expected to have an associated edgecut significantly lower than what would be anticipated. As evident in Figure 9-3, no single representative partitioning was indicated by observation of the trend of edge-cut vs. *k*. The balanced partitioner METIS™ was also used to search for an optimal partitioning (http://www-users.cs.umn.edu/~karypis/metis). This too failed to find a single optimal partitioning. Figure 9-3 shows the trend for both methods. The failure of these methods to find clearly defined modular organization implies that the metabolic pathways of *E. coli* are highly interconnected. This suggests that a definition of modularity other than simple isolation may be required. Partitioning 24 (k24) generated by the spectral partitioner was selected to exemplify the types of subgraphs generated, because it had the fewest edges cut compared to the METIS™ partitioning for the same value of *k*.

K24 yields subgraphs of various sizes, from 1 to 99. Subgraph 23 in Figure 9-4 is a single component that contains subpathways common to three well-studied metabolic pathways: arginine and proline metabolism; glycolysis / gluconeogenesis; and glycine, serine, and threonine metabolism. Other subgraphs, such as subgraph 5 in Figure 9-5 contain multiple components. One component contains a subpathway common to the urea cycle along with alanine and aspartate metabolism; another component contains a subpathway common to fructose & mannose metabolism, pyruvate metabolism, and glycerolipid metabolism. A final component contains a subpathway common to both galactose metabolism, and starch and sucrose metabolism.

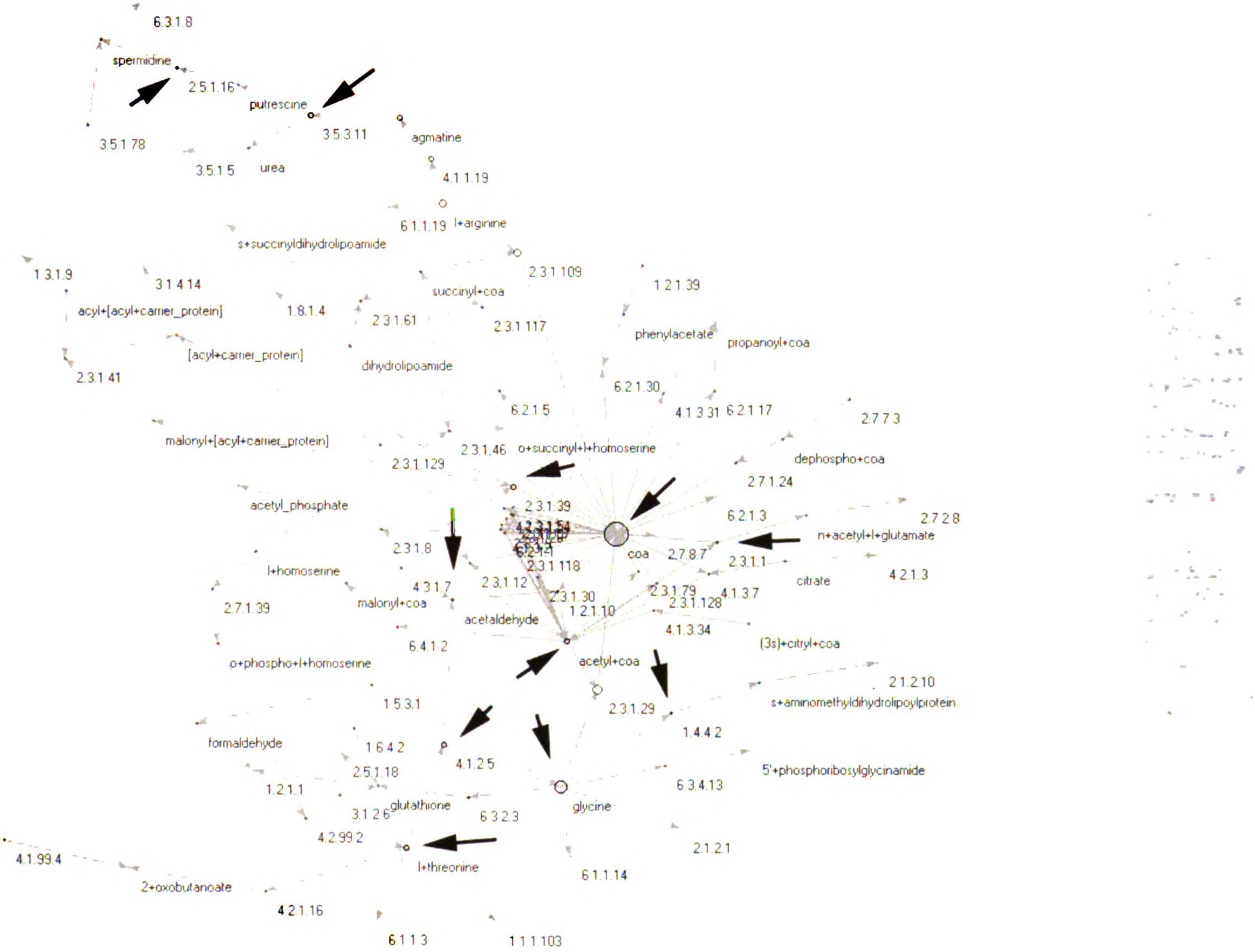


Figure 9-4. Subgraph 23 from $k = 24$, containing reactions shared by arginine and proline metabolism; glycolysis / gluconeogenesis; and glycine, serine, and threonine metabolism. Larger nodes have higher directed betweenness centrality. Arrows indicate articulation points.

To address the effect of missing information on prioritization quality, the difference and direction-of-shift between essential and nonessential gene centrality scores were tabulated for both the global system and the filtered system. As mentioned previously, the missing information is the presence of common metabolites and other high degree nodes. Table 9-4 lists the values for both global and filtered systems. The difference in undirected closeness centrality (UCC) between essential and nonessential enzymes was diminished and the prioritization direction was reversed after filtering. The corresponding differences in the three other metrics were also influenced. Undirected betweenness centrality (UBC) was generally unchanged, but direction was reversed. Directed closeness centrality (DCC) was improved and direction was reversed to reflect the correct direction. Directed betweenness centrality (DBC) was increased but direction was reversed. All metrics except DCC are negatively affected by removal of high degree nodes from the graph.

DISCUSSION

Presented is a family of methods to explore modular substructure and critical molecules in a metabolic network, utilizing solely topological information. This generalized approach seeks to explore how far one can proceed in predicting and prioritizing sensitive aspects of a molecular system in the absence of kinetic or probabilistic information, which may be unavailable or unreliable. The methodology provides a potentially powerful approach for prioritization and discovery of candidate intervention targets. The overall usefulness of the methodology is subordinate to the quality and reliability of information on the intermolecular influences within the molecular system. The usefulness of individual applications is expected to be variable.

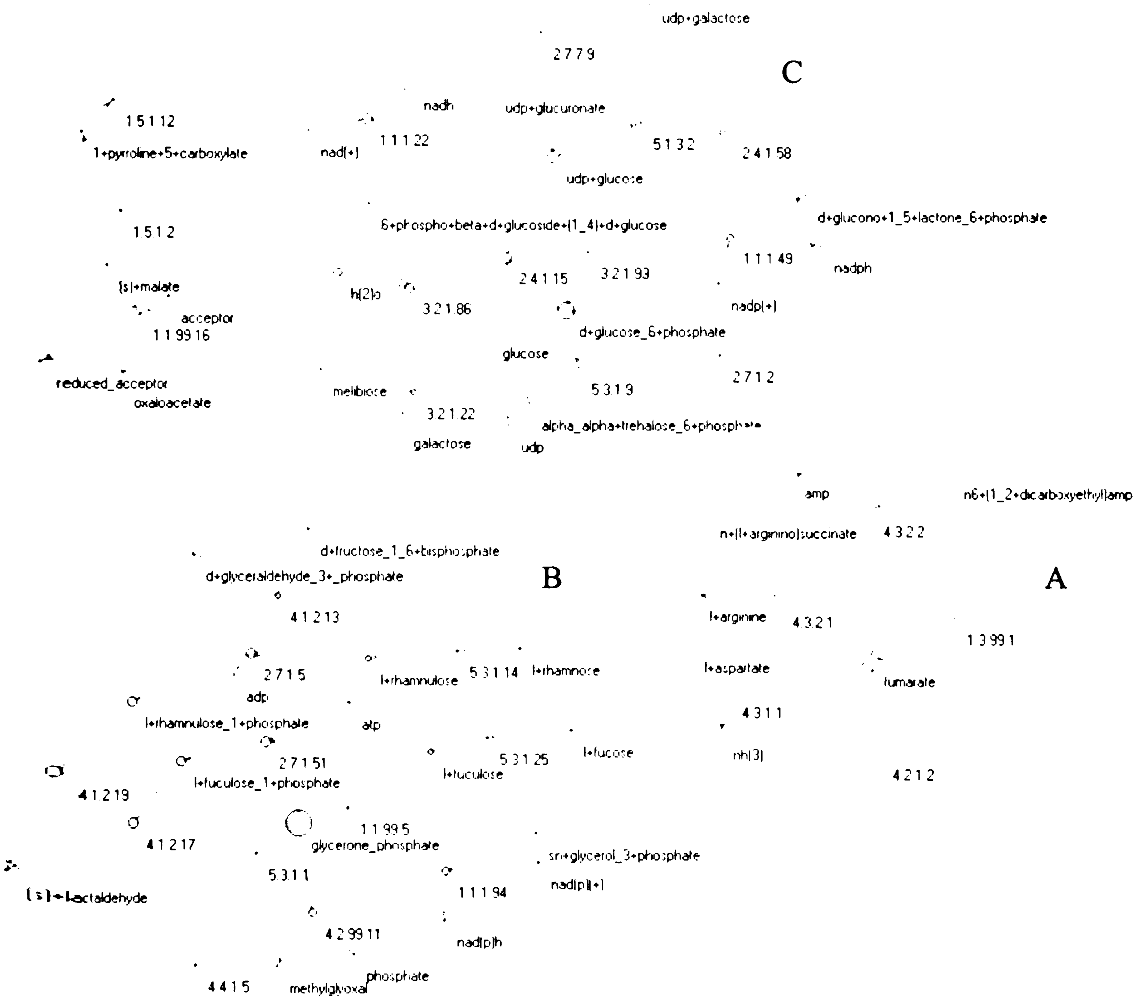


Figure 9-5. Partition 5 from k = 24. Larger nodes have higher directed betweenness centrality. Components represent subpathways common to 2 or more pathways. A) urea cycle and alanine and aspartate metabolism. B) fructose and mannose metabolism, pyruvate metabolism, and glycerolipid metabolism. C) galactose metabolism, and starch and sucrose metabolism.

The methodology provides a quick and inexpensive screen having some risk of failure. It also offers the potential for immediate success when analyzing systems that might be unapproachable objectively.

In order to determine the extent to which each metric correctly prioritizes critical molecules, I measured the difference in distributions of essential and nonessential genes. An ideal prioritization should discriminate between the essential and nonessential enzymes by producing two non-overlapping distributions of metric values. The distribution for each group was represented with discrete cumulative distribution functions. The difference between distributions was measured by median shift and application of the K-S test. Most metrics show significant differences between essential and nonessential enzymes, and correctly prioritize essential enzymes over nonessential ones. The metric providing the best discrimination is undirected betweenness centrality. It has the largest measure of D and S, with the sign of S indicating the proper direction in prioritization (from Table 9-4). Smaller values for closeness are assumed to indicate critical nodes and a negative S value indicates that, in general, essential enzymes have smaller values than nonessential enzymes. Larger values for betweenness are assumed to indicate critical nodes, and a positive S value indicates that essential enzymes have larger values than nonessential enzymes. The directed closeness centrality metric does not discriminate well between these two phenotypes. DCC is a measure of the distance from a node to all other nodes reachable by directed paths, or how far downstream a molecule is in a molecular pathway.

Articulation points seem not to be indicative of essential enzymes, as the fraction of essential enzymes that are articulation points is similar to the fraction of essential enzymes in the non-articulation point population. This may be due, in part, to the fact that for a given reaction there may be several enzymes that can catalyze it. Most articulation points are metabolites common to pathways highly prioritized by centrality metrics.

Prioritization metrics based on centrality discriminate between essential and nonessential enzymes, consistent with initial assumptions about the topological characteristics of critical molecules. All metrics are sensitive to missing information when prioritizing enzymes. Such sensitivity may imply that, in this context, enzymes are only as important as their metabolites. When important metabolites are missing from a system, the related enzymes are less significant. Looking at the reduction in undirected betweenness centrality values for essential enzymes, it is clear that in the global case, essential enzymes are very central to the system, and that after removal of common metabolites they occupy more peripheral positions.

The quality of a prioritization for essential metabolites was examined by comparing it with known key metabolites for *E. coli*. I considered a list of 16 key metabolites, which represent the common biosynthetic source for all cell materials. Most of the 16 were identified by one or more centrality metrics. For closeness and betweenness metrics, the threshold for selection was a score less than 1.5 standard deviations below the mean and greater than 5 standard deviations above the mean, respectively. Analysis of the global system resulted in nine matches to the list of sixteen. Aspartate and succinate were not selected directly, but associated enzymes were. Analysis of the system after filtering out common metabolites resulted in twelve matches to the list of 16, including aspartate and succinate. Aspartate was an articulation point in the filtered system and succinate's directed betweenness centrality score was above threshold.

Most of those selected as potentially critical molecules have qualifying directed betweenness centrality values and/or were articulation points. However, none had qualifying directed closeness centrality values. In the paper by Wagner and Fell, the metabolic network was used to generate two separate networks: one with only metabolites and another with only enzymes[85]. The metric used to predict critical molecules was similar to undirected closeness centrality, and results were reported only for the network containing metabolites. They had success using closeness centrality, but here, in a realistic, inclusive network model, closeness was not the most reliable predictor. Betweenness centrality and articulation points were most reliable, especially after filtering out the common metabolites.

The development of betweenness centrality and articulation points as predictors for critical areas in a system of interactions assumes that biological molecular systems are comprised in part of assemblages of interacting modules. In this context, each module represents a highly interconnected subsystem that is somewhat isolated from the rest of the system. Elements (whether known or unknown) linking a modular substructure to the rest of the system are essential to information flow or mass transfer throughout the system. Links between modules and the rest of the system can be thought of as an ideal cutset if the system were to be partitioned. Nodes incident to the edges of this cutset should have relatively large betweenness centrality scores. To find these nodes, articulation points and modules are located, with the latter being approximated by optimal graph partitioning, connected components, and strongly connected components.

In this analysis no optimal graph partitioning was found to identify an ideal cutset as defined above. Instead, articulation points and nodes with high betweenness centrality values approximated nodes incident to this, as yet unidentified, ideal cutset.

In validating the predictions for critical molecules, many of the metabolites already known to be critical for *E. coli* were identified and both betweenness and closeness centrality metrics were shown to discriminate between essential and nonessential enzymes. The removal of common metabolites from the system did not affect the quality of critical metabolite prioritization, but did severely affect the prioritization for essential enzymes.

As no optimal partitioning was found, the system's strongly connected components were identified and compared to known metabolic pathways. Some SCCs contained entire pathways, while others represented undocumented cycles common to multiple pathways. Such cycles may represent certain chemistry that has been repeatedly co-opted during evolution in the development of new mechanisms. Alternatively, they may represent the random accretion of cross talk between established but constantly evolving metabolic pathways.

The topological analyses presented here were effective in quickly determining metabolically important molecules and interactions. When applied to other systems, an objectively determined prioritization may be useful in generating new hypotheses. It may also aid in identification and validation of potential therapeutic intervention targets by suggesting molecules and interactions on which to focus for maximum effect, or to avoid in order to prevent unwanted toxicity. Alternatively, the prioritization could be used to select a subset of variables for thorough computational sensitivity analysis.

METHODS

A model of *E. coli* metabolism was constructed by obtaining all the enzymatic reactions in the ENZYME database that had a corresponding gene in *E. coli* (http://www.expasy.ch/enzyme/). These reactions were translated into a graph by representing each substrate, product, and enzyme as graph nodes. For each reaction, edges were created, linking substrate nodes to enzyme nodes and linking enzyme nodes to product nodes. The modularity of the *E. coli* metabolic network and the prioritization of molecules were assessed separately.

Modularity was assessed by first finding the graph's components. The largest component was analyzed for an optimal partitioning. Additionally, this component's SCCs were identified. The partitions and SCCs both approximate modularity.

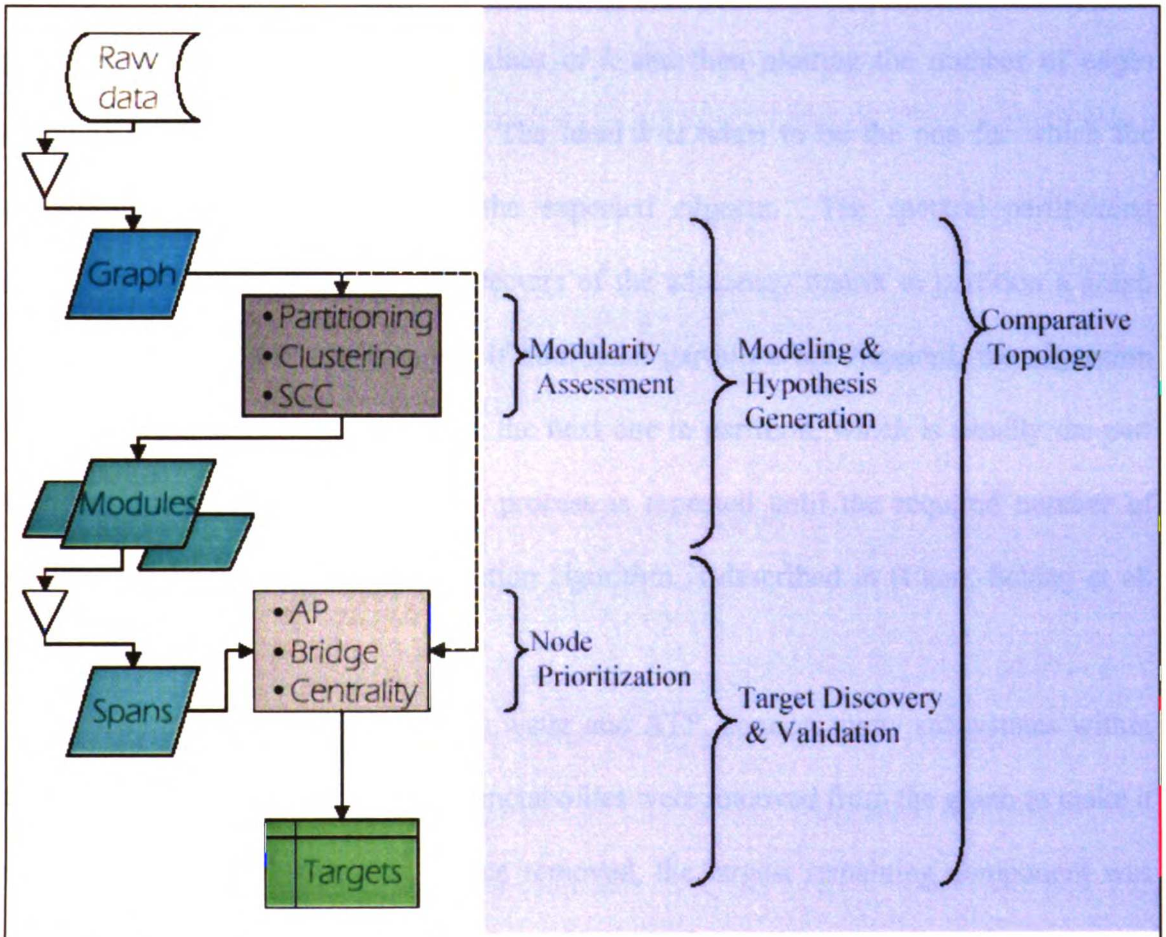


Figure 9-6. The steps involved in identifying critical objects and modularity in a network (graph) using said network's topology. Raw data on biological interaction or dependence is modeled by a graph or network. Next a variety of techniques are applied on the search for modular substructures. If found, the objects involved in linking separate modules are selected and prioritized according to several measures of criticality in a network. Above, AP stands for articulation point, and SCC stands for strongly connected component. In case modules are not found satisfactorily, all objects in the network are prioritized, and the highest priority nodes are hypothesized to be critical for information or mass flow the system being modeled. To the right: steps are labeled; the potential applicability of each phase's results are indicated; and the entire process labeled as Comparative Topology, as different systems can be objectively compared by their modularity and critical objects.

Partitions and SCCs are semi-isolated subgraphs—partitions via their minimal cutset property and SCCs by the absence of cycles between SCCs. The partitioning algorithm utilizes *ideal recursive spectral octasection* on the undirected graph, while the SCCs are defined for the directed graph, which is implied by the reaction description.

The number of network parts that optimally represent the system's modular substructure is unknown, *a priori*. The optimal number of partitions, *k*, is sought by partitioning the graph for several values of *k* and then plotting the number of edges spanning partitions (edgecut) vs. *k*. The ideal *k* is taken to be the one for which the edgecut is significantly less than the expected edgecut. The spectral partitioning algorithm uses three maximal eigenvectors of the adjacency matrix to partition a graph into at most eight parts at a time. If additional partitions are required, the algorithm chooses among the network parts for the next one to partition, which is usually the part with the maximal eigenvalue. This process is repeated until the required number of partitions is found. A spectral bisection algorithm is described in (Chan, Schlag et al. 1993)[70].

Common metabolites, such as water and ATP, connect many subsystems within the graph. These and other common metabolites were removed from the graph to make it easier to identify substructures. Once removed, the largest remaining component was reanalyzed. The pendant nodes (nodes with a single neighbor) were also removed to reduce the computational burden. Each component had its nodes prioritized with four different centrality metrics. Betweenness and closeness centrality scores were calculated for each node using directed and undirected edges. These metrics were labeled: Directed

Betweenness Centrality (DBC), Undirected Betweenness Centrality (UBC), Directed Closeness Centrality (DCC), and Undirected Closeness Centrality (UCC).

The module assignments and molecule prioritizations were validated using data from the PEC database (http://www.shigen.nig.ac.jp/ecoli/pec/), indicating essential *E. coli* genes; pathway annotations from the KEGG database (http://www.genome.ad.jp/kegg/); and literature references to key metabolites[85]. The PEC database compiles information from research reports and deletion mutant studies in order to classify *E. coli* genes as essential for growth, non-essential, or unknown. The KEGG database contains information on regulatory and metabolic pathways for several organisms. Information on key metabolites has been summarized for *E. coli* in the above-cited report. The prioritization of these key metabolites by each method was noted. The discrete cumulative distribution of essential enzymes was plotted for the measured range of each centrality metric. The distribution of nonessential enzymes was also plotted and the significance of the difference from that of the essential enzymes was tested. The significance of the difference between distributions was measured with the Kolmogorov-Smirnov (K-S) test. The K-S test measures the maximum value of the absolute difference, D, between two cumulative distribution functions, S_{N1} and S_{N2}. The significance of D was assessed by using Eq 9-1 to calculate the probability of D being greater than observed. In addition, the degree to which the quality of the prioritization is

$$P(D > observed\ ) = Q\left(\left[\sqrt{N_e} + 0.12 + 0.11 / \sqrt{N_e}\right]D\right) \qquad (9\text{-}1)$$

$$Q(\lambda) = 2\sum_{j=1}^{\infty}(-1)^{j-1}e^{-2j^2\lambda^2} \qquad (9\text{-}2)$$

$$N_e = \frac{N_1 N_2}{N_1 + N_2} \qquad (9\text{-}3)$$

affected by missing information was assessed by comparing the differences between prioritization before and after common metabolite removal. In Eq 9-3, N_e is the effective number of data points and N_1, N_2 are the number of data points for S_{N1}, S_{N2}.

Chapter 10: Conclusion

It is safe to say that when a person analyzes any system, they draw from their personal store of knowledge to evaluate elements in that system. Current models used for *in silico* drug screening are very large and complicated, partly due to the vast and growing amounts of biological and pharmacological information. Many simulation engines for *in silico* experimentation represent significant achievements and provide valuable contributions to investigators. They allow several therapeutic target intervention strategies to be computationally validated. Unfortunately, the complexity that allows such high quality models confounds attempts to use them for predicting new targets. A single person is unlikely to have intimate current knowledge of an entire system, and if set to the task of evaluating the model's various elements, an investigator would naturally start where s/he is most comfortable. This could unintentionally lead to a biased evaluation and to overlooking a system object that is a potential "diamond in the rough." Even allowing for such bias, considerable personal-objectivity and time is required for this task. In fact, a recent survey of on-market drugs of various indications stated that approximately seventy percent had similar targets. These targets were not found using the current technologies of *in silico* experimentation and bioinformatics resources, but these methods are likely to continue this trend. *In silico* tools are currently used only to further validate targets predicted by traditional methods. One reason for the small target pool is the reformulation of drugs to treat several indications. For example, Metformin™ has an indication for diabetes mellitus and obesity. An additional reason is the difficulty in navigating the complicated interdependencies of objects in many biological networks, the very same dependencies that result in robust and adaptive biological operation.

A strategy that evaluates large and highly connected biological networks and identifies potentially critical system elements is expected to be valued for many reasons; namely, the increases in research and development efficiency and the discovery of novel therapeutic targets.

As more genes are sequenced and Microarray experimentation increases, the number of targets begins to overwhelm *in vitro* validation capabilities. The number of targets evaluated per year per company is expected to rise from 40 to 200. Until *in silico* and *in vitro* validation methods are scaled up to match the number of target candidates, there is a need for additional target screening and prioritization methodologies.

The methods chosen to objectively investigate complicated models of biological systems were developed to test three hypotheses. H1.) If a model is partitioned into maximally isolated parts, the links between parts are critical to the system's operation. For example, if the model represented flow through a metabolic pathway and was best partitioned into two submodels, the capacity for mass transfer of the links would limit the flow of mass from the source to the sink. Reducing the capacity of the links would reduce the system's overall flow; as such these links would be ideal targets for the inhibition of a particular metabolic pathway. Given this hypothesis, methods were developed to partition a model into submodels. Along these lines, several researchers have hypothesized about the existence of biological modules that allow robust biological functions. The modules they describe are isolated chemically and/or physically from other biological processes. This isolation is also a property of the partitions needed to test H1; therefore, the same methods developed to test H1 can be used to identify natural modular structure. In addition to graph partitioning as a method for identifying linking nodes, articulation points and bridges were identified as they serve unique integrative roles in a graph. If these methods do find suitable partitions or modules, the number of links between them could be many. A target prioritization metric is needed for this potentially long list of targets. H2.) Given a graph consisting of nodes and edges, representing objects and their interactions, and a graph partitioning with maximally isolated partitions, the nodes with links to other partitions have high betweenness centrality. As betweenness centrality was shown to be a property of linking nodes, it was chosen as a metric to prioritize them. H3.) High betweenness centrality is a property of nodes that are critical to a system's operation. To test H2 and H3, a method was developed to calculate betweenness centrality and other centrality metrics for any node in a graph.

The methods developed to identify modularity and prioritize targets were validated by application to several models of biology. The first system was a model of blood coagulation. This model was presented as an effect diagram, indicating molecules involved in blood coagulation and their influences on each other. First, the optimal partitioning was found. Second, the molecules with links to other partitions were identified by ranking them by the fraction of neighboring nodes in the same partition. The molecules with the smallest fraction should be ideal targets according to H1. Several methods were used to optimally partition the blood coagulation system. A freely available balanced partitioner, METIS; a custom coded spectral partitioner; and strongly connected component algorithm. The targets identified by each were similar. These targets were validated by comparison to known causes of blood coagulation disorders. Partitioning identified all the molecules associated with the five primary blood coagulation disorders. Third, H2 was tested by calculating the betweenness and closeness centrality scores for each object in the network, and then checking if high centrality is a property of the previously identified linking nodes. The highest ranking objects, according to centrality, were found to have a high fraction of links to other partitions. This result indicates that centrality is a suitable prioritization metric for linking nodes, which are critical for a system's operation.

H3 required a quantitative model for which sensitivity analysis results could be compared to the prioritization metrics under development. The model chosen was the Obesity PhysioLab™. It is quite a large quantitative modeling system used by pharmaceutical companies to research obesity therapies and suggest biomarkers and patient populations for clinical trials. The procedures for finding an optimal partitioning resulted in no suitable partitioning of the system. Normally, the partitioning would be used to help identify linking nodes, which would then be prioritized by their centrality. Instead, all nodes had their centralities calculated. Those with significant centralities were selected as potential therapeutic targets for controlling obesity. A tabulation of weight control drugs and their molecular targets was compiled. Eighty percent of the known targets are included in the list of predicted targets. If the neighbors of the predicted targets were also considered, one hundred percent of the known drug targets are matched. To test H3, first two biomarkers were chosen to indicate weight loss: fat mass and rate of stored-triglyceride change. Second, the sensitivity of the biomarkers to therapies designed for several variables was calculated by simulating virtual clinical trials. By comparing centrality metrics of the variables to the calculated sensitivities, it was observed that the likelihood of a variable having a significant sensitivity increased with the value of the variable's centrality. The availability of a modified version of the obesity model also allowed the consistency of the prioritization methods to be tested.

This new version of the obesity model had many more equations and variables, leading to a much larger graph to analyze for modularity and critical targets. Regardless of the changes to the model, the great majority of the known obesity drug targets have significant centrality scores and increasing centrality is again shown to indicate the increased likelihood of observing significant biomarker sensitivity. Modularity assessments of the obesity models indicate a system organized in a noncompartmentalized fashion. Thus, nodes that have high centrality, that are articulation points, or that are incident to a bridge, were used to approximate linking nodes.

After showing that linking nodes are good therapeutic targets, that high centrality is a property of linking nodes, and that better centrality of an object implies better system sensitivity to the object; the techniques for modularity assessment and target prioritization were applied to another "real world" model of biology. A model of *E. coli* metabolism was created from a database of enzyme catalyzed reactions. Reaction coordinates were not always as indicated in the database, so analyses were done with and without respect to the indicated reaction coordinate. Inconsistent capitalization of metabolites was another issue that was dealt with in creating the graph model from the database of reactions. After prioritization of nodes according to centrality, a majority of the metabolites that serve as a source for most biosynthetic materials are found to have significant centralities. For enzymes, those essential to the growth of *E. coli* have better centrality scores than nonessential enzymes. Unfortunately, there is much overlap in the centrality distributions of these two enzyme groups, even though there is high confidence in the difference between the two distributions. A reprioritization of metabolites and enzymes after common metabolites, such as H_20, ATP, NH_2, etc, were removed from the graph resulted in an improved prioritization for metabolites, but the enzyme prioritization was negatively affected. This may be due to differences in degree distribution between these two classes of molecules and the fact that they are analyzed together in the same mixed mode graph.

An assessment of the system resulted in no clear partitioning into modules, but there were interesting observations of the makeup of the partitions and SCCs. The partitions and SCCs contained reactions that have been assigned to multiple well-studied pathways. This overlap represents potential cross talk and common processes among different biosynthetic pathways. The degree of overlap suggests that efforts to model particular pathways and predict therapeutic targets should be informed by steps attributed to other "pathways."

The steps developed for the analysis of large and complicated systems can be divided into two phases, modularity assessment and target prioritization. Suitable methods for modularity assessment would assign a system's objects into groups that have few connections with other groups and many connections among objects in the same group. In short, each group approximates the concept of a module due to its isolation from the rest of the system. The methods chosen were graph partitioning, graph clustering, and SCC identification. As the number of partitions that best describes a system is unknown *a priori*, several numbers are chosen and the one which results in a number of edges cut by the partitioning is lower than expected by observing the trend in edge-cut vs. number of partitions. This method resulted in a division of the blood coagulation pathway into the intrinsic pathway, extrinsic pathway, common pathway, and an additional segment. For other, more complicated, systems, no single number of partitions was especially appropriate. This indicates that these systems are indeed amodular, or that graph-partitioning methods that primarily minimize edge-cut are insufficient. Some partitions of *E. coli* metabolism contained single well-connected components, but others contained multiple components. Ideally, a partition should contain a single component, as the partition should contain a group of objects that interact with each other and not multiple unassociating groups. Approximating modules by the SCCs of a graph generally results in larger, single-component, groupings with lower edge density containing feedback loops and other cyclic pathways. SCCs are only defined for a graph with directional relations. So, systems like protein interaction networks cannot be analyzed in this manner, and uncertainty in the directionality of a relation affects what SCCs are found.

The target prioritization phase ranks nodes according to four metrics: directed and undirected betweenness centrality, and directed and undirected closeness centrality. Roughly, betweenness centrality is a measure of how often a node lies on a direct path of influence or communication between other nodes; closeness centrality is a measure of how close a node is to other nodes in the system. These measures require the calculation of the shortest paths between all node pairs. Said paths can be different lengths for the directed and undirected version of the same graph. In general, prioritizations based on directed betweenness centrality correctly highlighted known drug targets and best discriminated between essential and nonessential genes. Undirected closeness centrality and undirected betweenness centrality also provided useful prioritizations. Directed closeness centrality in general provided unsatisfactory prioritizations of known therapeutic targets and biomarker-sensitive variables. Although the stated hypotheses focused on betweenness centrality—which did prove to be a very useful target prioritization metric—undirected closeness centrality also provided useful prioritizations, and should also be considered in future analyses. Consideration of closeness in addition to betweenness results in minor additional computation, as both algorithms have common steps.

In coding the heuristic to implement the modularity assessment and node prioritization methods, new algorithms were created. These algorithms have general utility, even though they were developed for this specialized purpose. The centrality calculations required a shortest path algorithm, and the algorithm by Djikstra was chosen. This algorithm only retains a single shortest path, but for some node pairs there are multiple paths with the same length that is shorter than all other paths between the two nodes. Djikstra's algorithm was adapted to retain multiple shortest paths. Though a public graph partitioner was used to assess modularity a novel spectral partitioner was created and a new graph-clustering algorithm was developed. Spectral methods to partition graphs are well known and generally take the form of recursive bisection algorithms. The new algorithm recursively partitions a graph into at most eight sections at a time instead of two. A partitioning scheduler is also part of the new algorithm. This scheduler decides how many new groups need to be created at each step and selects which existing partition to divide. The eigenvalues of each group indicate the quality of partitions generated from that group. The clustering algorithm is hierarchical and agglomerative; it utilizes a combination function, expressed as several factorial functions, which initially caused to memory errors. A new algorithm was written to overcome these memory errors. It expands the factorials, simplifies, and evaluates the new expression. Each of these algorithms was written in the Python™ computer programming language. In fact, the entire heuristic was written in this language, which does not provide the fastest running code, but does result in easy to code and understand software, an ideal feature for rapid prototype development. A graphical user interface was created to automate analyses and display the results graphically. A user has two options for the visual layout of the graph: a static one, and a dynamic one. The ability to remove objects from a system and rerun the battery of analyses provides a venue for observing changes in the centrality distribution and modularity due to changes in the system.

Following this line of investigation of complicated biological systems, the next logical steps are those that increase the accessibility of the heuristic and further validate the underlying concepts. Increasing technical accessibility of the heuristic can be done in several ways. A graph-managing database that allows subgraphs to be extracted from a larger graph via logical queries and allows easy storage of analyses results would simplify the analysis process. Currently, if a graph consists of multiple components, a new input file has to be created with a single component for analysis. Automatic processing and analysis of these components would be useful. A function that enables the transfer of data in the Systems Biology Markup Language (SBML), and possibly integration with the Systems Biology Workbench (SBW) would facilitate the sharing of system models and analyses tools[91, 92]. To further validate the concepts developed here a few areas can be investigated: the topological characteristics of known limiting enzymes, the effect of edge and node weightings from kinetic or stochastic parameters, and the sensitivity of prioritization metrics to error in the system's representation.

Increasing the availability and utility of topological methods to study complicated biological networks gains importance as the size and complexity of biological models grow. The current trends in data integration and network representations of biology, and the increasing interest in systems biology herald the wider application of network-based analyses. Currently, many groups are working toward the generation of networks that model biological systems, and while these efforts are essential there are few efforts to develop methods that describe the organizational structure of a network and identify critical network features that may also be important biologically. The methods and heuristic developed here may fill this need by identifying modularity in biological systems and identifying objects in these systems which are critical to system performance. Importantly, these methods require, as input data, only a listing of the objects in the system and the relationships between them, but not the stochastic and kinetic characteristics of these objects and relationships. The simplicity of the data required makes the methods applicable to a wide variety of system models where detailed data might be unavailable or of low quality. While these methods correctly identify known & novel therapeutic targets, essential metabolites, disease biomarkers, and other critical objects in biological systems, they do not predict exact quantitative features of objects (e.g. flux control coefficients). The ranking generated is useful in hypothesis generation and prioritization of validation experiments to get at these quantitative features, but the techniques developed here are best used in conjunction with established investigative techniques that can either predict or measure the exact level to which a potential target affects a system.

References

1. Albert, R., H. Jeong, and A.L. Barabasi, *Error and attack tolerance of complicated networks.* Nature, 2000. **406**(6794): p. 378-82.
2. Karypis, G. and V. Kumar, *Multilevel algorithms for multi-constraint graph partitioning.* 1998, Department of Computer Science, University of Minnesota.
3. Biggs, N., *Algebraic graph theory.* Cambridge tracts in mathematics ; 67. 1974, London, New York: Cambridge University Press. vi, 170.
4. Seary, A. and W. Richards. *Partitioning Networks by Eigen vectors.* in *International Conference on Social Networks.* 1996.
5. Karypis, G. and V. Kumar, METIS family of Multilevel Partitioning Programs. Access (2001)
6. Doreian, P. and L. Albert., *Partitioning Political Actor Networks: Some Quantitative Tools for Analyzing Qualitative Networks.* Journal of Quantitative Anthropology, 1989. **1**: p. 279-91.
7. White, D.R., V. Batagelj, and A. Mrvar, *Analyzing Large Kinship And Marriage Networks With Pgraph and Pajek.* Social Science Computer Review, 1999. **17**(3): p. 245-74.
8. Karp, P.D. and S. Paley, *Integrated access to metabolic and genomic data.* J Comput Biol, 1996. **3**(1): p. 191-212.
9. Entelos [Menlo Park, CA]. Access (2001)
10. Levine, H.A., B.D. Sleeman, and M. Nilsen-Hamilton, *Mathematical modeling of the onset of capillary formation initiating angiogenesis.* J Math Biol, 2001. **42**(3): p. 195-238.
11. Bray, G.A. and F.L. Greenway, *Current and potential drugs for treatment of obesity.* Endocr Rev, 1999. **20**(6): p. 805-75.
12. Birmingham, K., *Experts predict bleak post-genomic era for drug R&D.* Nat Med, 2001. **7**(3): p. 262.
13. *The Fruits of Genomics: Drug Pipelines Face Indigestion Until The New Biology Ripens.* 2001, McKinsey Consulting Firm, Lehman Brothers.
14. Jenssen, T.K., et al., *A literature network of human genes for high-throughput analysis of gene expression.* Nat Genet, 2001. **28**(1): p. 21-8.
15. Hishiki, T., et al., *Developing NLP Tools for Genome Informatics: An Information Extraction Perspective. Genome Inform Ser Workshop.* Genome Inform, 1998. **9**: p. 81-90.
16. Friedman, N., et al., *Using Bayesian networks to analyze expression data.* J Comput Biol, 2000. **7**(3-4): p. 601-20.
17. Arkin, A., P.D. Shen, and J. Ross, *A test case of correlation metric construction of a reaction pathway from measurements.* Science, 1997. **277**(5330): p. 1275-1279.
18. Sasse-Dwight, S. and J.D. Gralla, *Footprinting protein-DNA complexes in vivo.* Methods Enzymol, 1991. **208**: p. 146-68.
19. Kuby, J., *Immunology.* 3rd ed. 1997, New York: W.H. Freeman. xxiv, 664.
20. Walhout, A.J. and M. Vidal, *High-throughput yeast two-hybrid assays for large-scale protein interaction mapping.* Methods, 2001. **24**(3): p. 297-306.
21. Pugh, B.F. and D.S. Gilmour, *Genome-wide analysis of protein-DNA interactions in living cells.* Genome Biol, 2001. **2**(4): p. REVIEWS1013.

22. Murphy, K. and S. Mian, *Modeling gene expression data using dynamic Bayesian networks*. 1999, University of California at Berkeley, Department of Computer Science: Berkeley, CA.

23. Zweig, G. and S. Russell, *Speech Recognition with Dynamic Bayesian Networks.* AAAI/IAAI, 1998: p. 173-180.

24. D'Haeseleer, P., S. Liang, and R. Somogyi, *Genetic network inference: from co-expression clustering to reverse engineering.* Bioinformatics, 2000. **16**(8): p. 707-26.

25. Kuffner, R., R. Zimmer, and T. Lengauer, *Pathway analysis in metabolic databases via differential metabolic display (DMD).* Bioinformatics, 2000. **16**(9): p. 825-36.

26. Mounts, W.M. and M.N. Liebman, *Qualitative modeling of normal blood coagulation and its pathological states using stochastic activity networks.* Int J Biol Macromol, 1997. **20**(4): p. 265-81.

27. Jeong, H., et al., *The large-scale organization of metabolic networks.* Nature, 2000. **407**(6804): p. 651-4.

28. Hartwell, L.H., et al., *From molecular to modular cell biology.* Nature, 1999. **402**(6761 Suppl): p. C47-52.

29. Lauffenburger, D.A., *Cell signaling pathways as control modules: complexity for simplicity?* Proc Natl Acad Sci U S A, 2000. **97**(10): p. 5031-3.

30. Rain, J.C., et al., *The protein-protein interaction map of Helicobacter pylori.* Nature, 2001. **409**(6817): p. 211-5.

31. Fromont-Racine, M., J.C. Rain, and P. Legrain, *Toward a functional analysis of the yeast genome through exhaustive two-hybrid screens.* Nat Genet, 1997. **16**(3): p. 277-82.

32. Thieffry, D. and R. Thomas, *Qualitative analysis of gene networks.* Pac Symp Biocomput, 1998: p. 77-88.

33. Fell, D., *Understanding the control of metabolism.* Frontiers in metabolism 2. 1997, London: Portland Press. xii, 301.

34. Heinisch, J., *Isolation and characterization of the two structural genes coding for phosphofructokinase in yeast.* Mol Gen Genet, 1986. **202**(1): p. 75-82.

35. Waley, S.G., *A note on the kinetics of multi-enzyme systems.* Biochem J, 1964. **91**(3): p. 514-7.

36. Meyer, T. and L. Stryer, *Molecular model for receptor-stimulated calcium spiking.* Proc Natl Acad Sci U S A, 1988. **85**(14): p. 5051-5.

37. Yi, T.M., et al., *Robust perfect adaptation in bacterial chemotaxis through integral feedback control.* Proc Natl Acad Sci U S A, 2000. **97**(9): p. 4649-53.

38. Simpson, T.W., B.D. Follstad, and G. Stephanopoulos, *Analysis of the pathway structure of metabolic networks.* J Biotechnol, 1999. **71**(1-3): p. 207-23.

39. Acerenza, L., *Design of large metabolic responses. Constraints and sensitivity analysis.* J Theor Biol, 2000. **207**(2): p. 265-82.

40. Sen, A.K., *Application of electrical analogues for control analysis of simple metabolic pathways.* Biochem J, 1990. **272**(1): p. 65-70.

41. Sen, A.K., *Metabolic control theory: a graph-theoretic approach.* Biomed Biochim Acta, 1990. **49**(8-9): p. 817-27.

42. Sen, A.K., *Topological analysis of metabolic control.* Math Biosci, 1990. **102**(2): p. 191-223.

43. King, E.L. and C. Altman, *A schematic method of deriving the rate laws for enzyme-catalysed reactions.* J Phys Chem, 1956. **60**: p. 1375-1381.

44. Henley, E.J. and R.A. Williams, *Graph theory in modern engineering; computer aided design, control, optimization, reliability analysis*. Mathematics in science and engineering, v. 98. 1973, New York: Academic Press. xvi, 303.

45. Simpson, T.W., G.E. Colon, and G. Stephanopoulos, *Two paradigms of metabolic engineering applied to amino acid biosynthesis.* Biochem Soc Trans, 1995. **23**(2): p. 381-7.

46. Schilling, C.H. and B.O. Palsson, *The underlying pathway structure of biochemical reaction networks.* Proc Natl Acad Sci U S A, 1998. **95**(8): p. 4193-8.

47. Schuster, S., D.A. Fell, and T. Dandekar, *A general definition of metabolic pathways useful for systematic organization and analysis of complicated metabolic networks.* Nat Biotechnol, 2000. **18**(3): p. 326-32.

48. Pfeiffer, T., et al., *METATOOL: for studying metabolic networks.* Bioinformatics, 1999. **15**(3): p. 251-7.

49. Coupe, V.M., et al., *Using sensitivity analysis for efficient quantification of a belief network.* Artif Intell Med, 1999. **17**(3): p. 223-47.

50. Far, B., M. Nakamichi, and M. Sampei, *Qualitative Sensitivity Analysis.* J. of Japanese Soc. For Art. Intelli, 1991. **6**(1): p. 84-95.

51. Pronet Online: Protein Interactions on the Web. Access (2001)

52. DIP Database: Database of Interacting Proteins. Access (2001)

53. GenBank Portal. Access (2001)

54. ENZYME: Enzyme nomenclature database. Access (2001)

55. Portugaly, E. and M. Linial, *Estimating the probability for a protein to have a new fold: A statistical computational model.* Proc Natl Acad Sci U S A, 2000. **97**(10): p. 5161-6.

56. GeneScape Portal Web Site. Access (2001)

57. Uetz, P., et al., *A comprehensive analysis of protein-protein interactions in Saccharomyces cerevisiae.* Nature, 2000. **403**(6770): p. 623-7.

58. Karp, P.D., et al., *Eco Cyc: encyclopedia of Escherichia coli genes and metabolism.* Nucleic Acids Res, 1999. **27**(1): p. 55-8.

59. Legrain, P. and L. Selig, *Genome-wide protein interaction maps using two-hybrid systems.* FEBS Lett, 2000. **480**(1): p. 32-6.

60. Becker, M.Y. and I. Rojas, *A graph layout algorithm for drawing metabolic pathways.* Bioinformatics, 2001. **17**(5): p. 461-7.

61. Jeong, H., et al., *Lethality and centrality in protein networks.* Nature, 2001. **411**(6833): p. 41-2.

62. Kanehisa, M. and S. Goto, *KEGG: kyoto encyclopedia of genes and genomes.* Nucleic Acids Res, 2000. **28**(1): p. 27-30.

63. Kolpakov, F.A., et al., *GeneNet: a gene network database and its automated visualization.* Bioinformatics, 1998. **14**(6): p. 529-37.

64. Hofestadt, R. and S. Thelen, *Quantitative modeling of biochemical networks.* In Silico Biol, 1998. **1**(1): p. 39-53.

65. Wasserman, S. and K. Faust, *Social network analysis: methods and applications*. Structural analysis in the social sciences ; 8. 1994, Cambridge; New York: Cambridge University Press. xxxi, 825.

66. Rigoulet, M., et al., *Redistribution of the flux-control coefficients in mitochondrial oxidative phosphorylations in the course of brain edema*. Biochim Biophys Acta, 1988. **932**(1): p. 116-23.

67. Edwards, J.S., R.U. Ibarra, and B.O. Palsson, *In silico predictions of Escherichia coli metabolic capabilities are consistent with experimental data*. Nat Biotechnol, 2001. **19**(2): p. 125-30.

68. Ahuja, R.K., T.L. Magnanti, and J.B. Orlin, *Network flows: theory, algorithms, and applications*. 1993, Englewood Cliffs, N.J.: Prentice Hall. xv, 846.

69. Kernighan, B. and S. Lin, *An efficient heuristic procedure for partitioning graphs*. Bell System Technical Journal, 1970. **29**: p. 291-307.

70. Chan, P., M. Schlag, and J. Zien. *Spectral K-Way Ratio-Cut Partitioning and Clustering*. in *30th ACM/IEEE Design Automation Conference*. 1993.

71. Bock, R. and S. Husain, *An adaptation of Holzinger's B-coefficients for the analysis of sociometric data*. Sociometry, 1950. **13**: p. 146-153.

72. Anthonisse, J., *The Rush in a Graph*. 1971, Amsterdam: Mathematische Centrum.

73. Freeman, L.C., *A set of measures of centrality based on betweenness*. Sociometry, 1977. **40**: p. 35-41.

74. Stephenson, K. and M. Zelen, *Rethinking centrality: methods and examples*. Social Networks, 1989. **11**: p. 1-37.

75. Cormen, T.H., C.E. Leiserson, and R.L. Rivest, *Introduction to algorithms*. The MIT electrical engineering and computer science series. 1990, Cambridge, MA: MIT Press. xvii, 1028.

76. Science's STKE: Signal Transduction Knowledge Environment and Connections Map. Access (2001)

77. Covert, M.W., et al., *Metabolic modeling of microbial strains in silico*. Trends Biochem Sci, 2001. **26**(3): p. 179-86.

78. Hybridgenics, PIMRider. Access (2002)

79. Anderson, A. and C.A. Hunt, *Biological Network Topology and Clues to Finding Potential Drug Targets*. 2001, Department of Bioengineering, University of California, San Francisco.

80. Online Mendelian Inheritance in Man, OMIM ™. Access (2000)

81. *Guidance for the Clinical Evaluation of Weight Control Drugs*. 1996, Food and Drug Administration: Rockville, MD.

82. *Clinical Investigation of Drugs Used in Weight Control*. 1997, European Agency for the Evaluation of Medicinal Products, Committee for Proprietary Medicinal Products (CPMP): London, UK.

83. Csete, M. and J. Doyle, *Reverse Engineering of Biological Complexity*. Science, 2002. **295**: p. 1664-69.

84. Nobel, D., *Modeling the Heart-from Genes to Cells to Whole Organs*. Science, 2002. **295**: p. 1678-82.

85. Wagner, A. and D.A. Fell, *The small world inside large metabolic networks*. Proc R Soc Lond B Biol Sci, 2001. **268**(1478): p. 1803-10.

86. Chartrand, G., *Introductory graph theory*. Unabridged and corr. ed. 1985, New York: Dover. xii, 294.

87. Tomita M, H.K., Takahashi K, Shimizu TS, Matsuzaki Y, Miyoshi F, Saito K, Tanida S, Yugi K, Venter JC, Hutchison CA 3rd, *E-CELL: software environment for whole-cell simulation.* Bioinformatics, 1999. **15**(1): p. 72-84.

88. Holtzman, S., *In silico toxicology.* Ann N Y Acad Sci, 2000. **919**: p. 68-74.

89. Stephens, M.A., Journal of the Royal Statistical Society, 1970. **32**: p. 115-122.

90. Press, W.H., et al., *Numerical Recipies in C: the art of scientific computing*. 2nd ed. 1992, New York, NY: Press Syndicate of the University of Cambridge.

91. The Systems Biology Markup Language (SBML). Access (2002)

92. Hucka, M., et al., *The ERATO Systems Biology Workbench: enabling interaction and exchange between software tools for computational biology.* Pac Symp Biocomput, 2002: p. 450-61.

APPENDIX A: Tables

Table A-5. The best scoring molecules for each metric when applied to the global graph of *E. coli* metabolism. The criteria for significant a closeness score is less than 1.5 standard deviations below the mean; and for betweenness, greater than 5 standard deviations above the mean. The betweenness centrality metrics are Directed Betweenness Centrality (DBC), and Undirected Betweenness Centrality (UBC). The closeness centrality metrics are Directed Closeness Centrality (DCC), and Undirected Closeness Centrality (UCC).

name	DBC	name	UBC	name	UCC	name	DCC
h(2)o	329597.57	h(2)o	926560.21	h(2)o	61195	5.1.3.4	206370
diphosphate	187832.89	atp	463531.85	atp	61509	2.1.1.80	206405
Phosphate	152231.1	diphosphate	252937.18	6.3.5.1	61570	2.4.2.1	206442
2.4.2.17	115916.03	phosphate	233849.67	6.3.5.5	61620	2.7.7.8	206442
Atp	115640.03	co(2)	180008.72	2.5.1.6	61626	2.4.2.4	206442
2.4.2.14	68923.041	adp	151428.15	2.5.1.17	61672	l-ribulose 5-phosphate	206468
Nadh	63737.172	nad(+)	116976.36	2.7.9.2	61718	protein l-glutamate	206513
Coa	54026.964	coa	114334.13	6.3.5.2	61718	2.7.2.3	206563
1.6.6.9	48978.955	nadp(+)	77143.998	6.3.5.3	61750	2.7.4.9	206567
nh(3)	48202.119	nadh	62298.221	2.7.9.3	61784	2.7.1.16	206567
Nadph	40018.254	s-adenosyl-l-methionine	61519.275	3.6.3.3	61796	2.7.1.17	206567
nad(+)	36014.329	pyruvate	61301.993	3.6.3.16	61796	2.1.2.9	206594
4.1.1.31	35719.814	6.3.5.1	55150.653	3.6.3.2	61796	3.5.4.19	206610
Pyruvate	32317.047	amp	54150.309	3.6.3.5	61796	3.1.5.1	206614
4.1.2.15	30539.266	l-glutamate	51236.229	3.6.3.4	61796	3.6.1.41	206614
4.1.2.16	30539.266	nh(3)	51094.799	3.6.3.14	61796	3.1.4.16	206614
l-glutamate	30437.633	nadph	51049.506	3.6.3.12	61796	3.2.1.22	206614
nadp(+)	30039.785	3.2.2.9	49335.901	phosphate	61873	3.1.1.61	206614
co(2)	29844.154	s-adenosyl-l-homocysteine	46152.602	adp	61903	3.1.1.5	206614
6.3.5.1	24518.267	2.7.1.23	42552.395	3.6.1.1	61960	3.1.1.32	206614
6.3.1.1	19110.765	4.1.1.50	40928.698	diphosphate	61973	3.1.1.45	206614
nad(p)h	18938.492	6.3.5.5	36968.21	co(2)	62045	3.5.1.11	206614
4.2.1.52	18795.241	2.5.1.6	35007.294	2.4.2.14	62098	3.5.2.5	206614
acetyl-coa	18599.108	1.8.1.2	32909.376	4.1.1.31	62134	3.5.2.6	206614
4.2.1.70	17639.911	acetyl-coa	32577.535	nad(+)	62139	2.7.1.19	207286
alpha-d-glucose 1-phosphate	17054.914	1.4.1.4	28639.908	3.5.4.25	62158	2.7.8.20	207401
l-glutamine	16938.821	6.3.5.2	28098.306	3.6.1.23	62168	phosphate	207544
6.3.1.2	16119.848	1.6.6.4	27851.673	3.1.7.2	62174	3.1.4.14	207587
h(2)o(2)	15848.814	6.3.3.3	27731.659	3.6.1.31	62174	3.5.4.10	207591
4.4.1.16	15010.137	2.5.1.17	27263.225	3.6.1.41	62180	3.5.2.3	207609
reduced acceptor	14665.137	2.7.9.2	27257.469	amp	62279		

1.4.1.4	14578.233	1.2.1.16	26893.984	1.2.2.2	62290
2.3.1.8	14473.344	o(2)	26133.707	6.3.3.3	62298
1.2.1.16	13673.622	6.3.5.3	24737.354	3.1.3.11	62300
d-ribose 5-phosphate	12942.387	1.1.1.22	23610.129	6.3.4.14	62306
6.3.4.2	12909.198	nad(p)(+)	22005.178	l-glutamate	62307
6.2.1.5	12481.008	nad(p)h	22005.178	1.4.1.4	62310
1.6.6.4	12477.233	3.6.3.16	20506.59	4.2.99.2	62314
2-oxoglutarate	12354.885	3.6.3.5	20506.59	4.1.2.15	62316
3.5.4.2	12243.556	3.6.3.4	20506.59	4.1.2.16	62320
l-alanine	12019.521	3.6.3.3	20506.59	coa	62321
adenine	11603.581	3.6.3.14	20506.59	3.1.3.27	62324
2.4.2.22	11183.284	3.6.3.12	20506.59	3.2.2.4	62326
1.6.6.8	10744.875	3.6.3.2	20506.59	3.6.1.13	62326
6.3.3.3	10602.256	6.2.1.5	20146.84	3.1.3.10	62326
acetate	10530.654			3.1.3.15	62328
6.2.1.1	10399.924			3.1.3.48	62328
1.5.3.1	10187.299			3.6.1.7	62328
1.11.1.6	10040.144			3.1.3.1	62328
6.3.5.2	10037.869			3.1.3.2	62328
5-phospho-alpha-d-ribose 1-diphosphate	9997.4471			2.9.1.1	62330
2.4.1.7	9931.5849			3.1.3.16	62332
1.6.1.2	9828.3354			3.1.3.26	62332
1.6.1.1	9828.3354			3.1.3.18	62332
3.2.2.9	9467.0887			3.6.1.11	62332
1.6.99.3	9315.6614			3.6.1.40	62332
nad(p)(+)	9248.8166			3.1.3.12	62332
1.6.4.5	9192.139			3.1.3.5	62332
3.5.1.1	9072.9784			3.1.3.3	62332
d-fructose 6-phosphate	8857.1424			1.5.1.12	62334
				3.5.3.19	62340
				3.5.1.5	62348
				nh(3)	62355
				6.2.1.5	62356
				6.3.1.2	62368
				6.3.4.2	62374
				1.1.1.22	62380
				2.7.2.2	62380
				4.2.1.51	62382
				4.1.1.48	62386
				pyruvate	62387
				4.2.1.1	62388

Table A-6. The best scoring molecules for each metric when applied to the filtered graph of E. coli metabolism. The criteria for significant a closeness score is less than 1.5 standard deviations below the mean; and for betweenness, greater than 5 standard deviations above the mean. The betweenness centrality metrics are Directed Betweenness Centrality (DBC), and Undirected Betweenness Centrality (UBC). The closeness centrality metrics are Directed Closeness Centrality (DCC), and Undirected Closeness Centrality (UCC).

Name	DBC	name	UBC	name	UCC	name	DCC
Coa	17900.238	l-glutamate	98179.487	coa	3089	5.1.3.1	472068
l-glutamate	11833.041	coa	92856.245	pyruvate	3103	d-ribulose 5-phosphate	473051
reduced acceptor	10405.129	pyruvate	85650.739	l-glutamate	3107	1.1.2.3	473072
4.4.1.16	10281.129	5-phospho-alpha-d-ribose 1-diphosphate	48354.627	2.3.1.1	3225	5.3.1.9	473075
Pyruvate	10129.405	2.3.1.1	43001.599	4.1.3.27	3307	ferrocytochrome c	473085
6.2.1.5	9730.6076	2.4.2.14	40889.652	acetyl-coa	3351	1.11.1.5	473098
succinyl-coa	9606.6076	d-fructose 6-phosphate	39323.667	2.3.1.54	3361	ferricytochrome c	473111
acetyl-coa	9059.9629	4.1.3.27	34683.582	oxaloacetate	3419	2.4.1.15	474043
Succinate	8607.5936	d-glyceraldehyde 3-phosphate	30951.634	2.6.1.1	3451	alpha_alpha-trehalose 6-phosphate	474055
l-alanine	8606.9109	2.6.1.16	29061.235	4.1.3.7	3485	d-glucose 6-phosphate	474059
2.3.1.46	8378.2688	acetyl-coa	27600.73	4.1.2.21	3487	d-glucose	474065
Acceptor	8104.848	oxaloacetate	26934.213	4.1.2.14	3487	3.2.1.93	474067
4.2.99.9	8095.7273	sn-glycerol 3-phosphate	25913.25	4.1.3.31	3505	2.7.1.2	474075
o-succinyl-l-homoserine	7958.0606	phosphoenolpyruvate	24743.324			uracil	476035
1.3.99.1	7707.4662	s-adenosyl-l-methionine	24675.791			glutathione	477042
2.6.1.66	7138.7544	2.3.1.54	24001.527			spermidine	477047
2.3.1.47	7128.9109	l-aspartate	23718.102			(s)-lactate	477053
2.6.1.16	5693.073	2.4.2.9	23419.118			5.3.1.6	478029
d-fructose 6-phosphate	5562.9669	2.3.1.47	23144.908			2.7.6.1	478029
1.7.99.5	4515.3298	glycine	22222.014			2.4.1.44	478040
2.3.1.1	4292.909	2.2.1.1	22207.435			1.2.1.22	478041

5_10-methylenetetrahydrofolate	4254.4405	l-serine	21803.152
l-glutamine	4013.4108	l-glutamine	21317.137
2-oxoglutarate	3742.2985	glycerone phosphate	21184.829
		d-ribose 5-phosphate	20172.047
		3.1.4.46	19924
		2.1.1.10	19630.69
		2.3.1.15	19594.014
		acceptor	19359.528
		reduced acceptor	19359.528

Table A-7. Articulation points for the filtered graph, listed with their centrality scores. The sort is based on decreasing UBC, where larger values are preferred.

UCC	UBC	DCC	DBC	name
3107	98179.487456	407304	11833.040888	l-glutamate
3089	92856.245297	241212	17900.238296	coa
3103	85650.738565	241398	10129.405197	pyruvate
3225	43001.598503	241438	4292.909036	2.3.1.1
3519	40889.652409	407383	1290.35269	2.4.2.14
3845	39323.666623	407470	5562.9669	d-fructose 6-phosphate
3307	34683.581704	407525	1881.774426	4.1.3.27
3351	27600.729885	241500	9059.962916	acetyl-coa
3419	26934.213408	241526	1346.767216	oxaloacetate
4623	24675.791095	489000	0	s-adenosyl-l-methionine
3893	23718.102207	407750	1028.577778	l-aspartate
3521	23144.907834	241862	7128.910864	2.3.1.47
3903	22222.014386	482022	608.5	glycine
3725	21803.152001	243730	1680.032468	l-serine
3515	21317.13746	407454	4013.410848	l-glutamine
4169	21184.828673	238933	253	glycerone phosphate
4315	19630.689984	240334	1088.363636	2.1.1.10
4485	18056.67644	239364	1260	2.6.1.62
3949	17816.116781	242082	1997	l-homocysteine
3937	17403.342881	243126	473.780195	1.1.99.5
4027	17085.529071	241112	1500	8-amino-7-oxononanoate
3673	16899.147225	241662	2735.520531	acetate
3587	16278.378599	242832	2447.496104	4.4.1.8
5825	16119.661234	486004	469.25	alpha-d-glucose 1-phosphate
3711	15753.845642	241620	8606.910864	l-alanine
4737	13126.085656	226604	798	s-adenosyl-l-homocysteine
4055	11186.532205	243364	4254.440476	5_10-methylenetetrahydrofolate
3911	9960.269914	407450	3742.298482	2-oxoglutarate
5435	9661.616567	473051	16	d-ribulose 5-phosphate
4703	9526.830385	222142	271	adenine
3673	9086.217171	242202	1589	2.3.1.39

UCC	UBC	DCC	DBC	name
3675	7532.875037	407517	1168.10607	6.3.5.2
4893	7353.34307	225871	534	3.2.2.9
4863	7263.435393	485010	426	l-threonine
6415	7164	485008	16	2.7.7.24
6025	7164	486006	6	1.1.1.44
8537	7164	489000	0	d-mannonate
3673	6802.590933	473072	2038	1.1.2.3
5575	6221.414187	480025	18	putrescine
3625	6181.379918	407525	282.200885	2.6.1.11
5667	6101.463417	483015	1030.25	udp-glucose
7007	5980	484013	15	dtdp-glucose
9129	5980	488001	1	4.2.1.8
5077	5980	238617	1012	7_8-diaminononanoate
4303	5980	240870	250	5.1.1.1
4459	5643.003923	481030	8	6.3.2.3
5671	4792	237871	762	6.3.3.3
5117	4792	489000	0	2-phospho-d-glycerate
7601	4792	483019	12	4.2.1.46
4803	4583.171575	398760	180	gdp
4781	4047.965201	240610	252	2.6.1.13
3697	3767.68497	485010	256	3.3.2.1
4239	3714.030186	243134	749.980195	1.4.99.1
5713	3604	487002	3	5.4.2.1
8197	3600	482026	7	dtdp-4-dehydro-6-deoxy-d-glucose
6267	3600	237126	510	dethiobiotin
6171	3600	479035	10	2.5.1.16
5441	3600	487003	4	l-histidinol
5289	3509.841775	397840	92	2.7.7.22
3911	3196.058029	239691	251	glyoxylate
4683	2912.824492	237020	254	6.3.2.9
4483	2857.267436	406833	83	4.1.1.11
4747	2775.938149	242384	496	a 2-oxo acid
3693	2552.752592	407533	137.10607	6.3.5.3
4277	2404.181818	243134	255.980195	1.1.99.1
4267	2404	489000	0	3.5.1.18
6039	2404	486006	3	1.1.1.23
6865	2404	236382	256	2.8.1.6
4501	2404	481030	8	1.4.4.2
3653	2368.230817	241458	1553.186334	4.1.3.2
4301	2138.428994	240870	604	6.3.2.8
6423	1798	485008	4	2.7.7.27
4777	1714.251953	244284	1584.366342	10-formyltetrahydrofolate
4243	1529.138073	406600	315	gmp
4595	1379.959704	240121	356	udp-n-acetylmuramoyl-l-alanine
8545	1204	489000	0	(s)-dihydroorotate
4737	649.237926	405684	235	2.7.4.8
4599	354.965105	488001	1	1.1.1.3

Table A-8. Articulation points from the global graph with their centralities. The sort is based on decreasing DBC, where larger values are preferred.

UCC	UBC	DCC	DBC	Name
61195	926560	207716	329597	h(2)o
61973	252937	207882	187832	diphosphate
61873	233849	207544	152231	phosphate
61509	463531	209664	115640	atp
62715	62298	208138	63737	nadh
62321	114334	208866	54026	coa
62911	51049	208458	40018	nadph
62139	116976	208960	36014	nad(+)
62387	61301	208674	32317	pyruvate
62307	51236	209038	30437	l-glutamate
63759	22005	209388	18938	nad(p)h
63755	6528	209308	17054	alpha-d-glucose 1-phosphate
63195	11430	209020	12942	d-ribose 5-phosphate
63473	26133	211192	7105	o(2)
62865	6126	209456	5210	l-serine
62554	12745	208601	4063	3.5.1.18
62883	18642	209192	4000	glutathione
63317	16573	211082	3812	udp-glucose
63440	3884	209699	3481	2.3.1.1
62572	7408	211971	3076	4.2.1.46
63927	5493	209438	2944	10-formyltetrahydrofolate
63663	4548	209478	2877	l-homocysteine
62576	8383	208601	2739	3.2.1.31
62324	18500	208605	2667	3.1.3.27
62880	16716	206567	1800	2.7.1.16
62540	14481	206614	1795	3.1.1.61
63035	470	210698	1347	imp
63747	2824	208479	1123	d-ribulose 5-phosphate
64271	5576	206468	901	l-ribulose 5-phosphate
62882	3210	208562	899	2.7.1.21
63833	6439	206513	898	protein l-glutamate
63965	2298	208498	897	d-glucuronate
63711	11136	208502	897	phosphatidylglycerol
63945	5576	208498	897	ll-2_6-diaminoheptanedioate
63959	2582	211868	897	dtdp-4-dehydro-6-deoxy-d-glucose
63695	4157	212542	761	5_10-methylenetetrahydrofolate
64031	74	209596	274	n-acetyl-l-glutamate
64065	13214	1097005	53	udp-n-acetyl-d-glucosamine
64637	5787	1100000	0	isopentenyl diphosphate
1451003	0	1100000	0	4-deoxy-l-threo-5-hexosulose uronate
1449007	0	1100000	0	[protein]-l-cysteine
1451003	0	1100000	0	3-cis-dodecenoyl-coa
1451003	0	1100000	0	alpha-d-glucose
1449007	0	1100000	0	(5-l-glutamyl)-peptide
1451003	0	1100000	0	cyclobutadipyrimidine (in

UCC	UBC	DCC	DBC	Name
				dna)
1451003	0	1100000	0	5-carboxymethyl-2-hydroxymuconate

UCC	UBC	DCC	DBC	Name
				dna)
1451003	0	1100000	0	5-carboxymethyl-2-hydroxymuconate

APPENDIX B: Website References

http://dip.doe-mbi.ucla.edu/, DIP Database: Database of Interacting Proteins

http://www.expasy.ch/enzyme/, ENZYME: Enzyme nomenclature database

http://www.ncbi.nlm.nih.gov/Genbank/index.html, GenBank Portal

http://www.shigen.nig.ac.jp/ecoli/pec/, Profiling of E. coli Chromosome (PEC) database

http://www-users.cs.umn.edu/~karypis/metis/, METIS family of Multilevel Partitioning Programs

http://www.genome.ad.jp/kegg/, KEGG: Kyoto encyclopedia of genes and genomes

APPENDIX C: User Manual

Introduction

Abraham Anderson
abeand@socrates.berkeley.edu

This tool set was created to analyze the topology of networks indicating critical points within the network and highlighting cohesive subsystems. It was first applied to biological networks, where the aim was to indicate and prioritize potential drug targets for validation. There are several analyses available in this tool set plus a graphical user interface (GUI) that graphically displays the analyses' results. The GUI allows interaction with the displayed system and the functionality to modify the system for reanalysis. With this tool set it's easy to find articulation points, bridges, strongly connected components, determine modularity, and calculate centrality scores.

Look to the developer's manual for details on selected functions. The developer's manual points out adjustable parameters to customize your analysis, and also areas for improvement.

This work was done with the Hunt Lab at the University of California at San Francisco as part of the Ph.D. dissertation of Abraham A. Anderson, a student in the joint Bioengineering Graduate Group of U.C.S.F. & U.C. Berkeley. All rights reserved. © Abraham Anderson, 2002. This work was partly supported by grant # LSI-10041 from the California Life Science Informatics Technology Program, and by grant # r25-gm56847 from the NIH National Institute of General Medical Sciences.

Requirements

X-files

Step_one.py
Step_two.py
Ac2met.py
Ac2bnt.py
cc.py
scc.py
get_centrality.py
get_centralityU.py
x_SUMMARY.py
peval.py
ccluster.py
ks_part3.py
Grapher_v09.py

Python™ files

Python2.0 [http://www.python.org]
Numeric Python package [http://sourceforge.net/projects/numpy]
TCL Plug-in version 2.0 [http://www.python.org/topics/Tkinter/download.html]

METIS™ files

METIS version 4.0 (compiled) [http://www-users.cs.umn.edu/~karypis/metis/metis/download.html]

Setting up the system for initial use.

Install the *Python™files* and ensure that its directory is in the system path. Create a directory "metis-4.0-compiled" and place the *METIS™files* inside. Create a new directory for the *X-files* and place them inside. This will be the working directory for all analyses. I suggest creating a new directory for each analysis and move the analyses output files there when they are created. This will avoid confusion when running multiple analyses.

Setting up your input files for analysis.

The inputs to the tool set are an edge file and a name file. These two files contain the structure of and the labeling for the system, which is abstracted to a graph for topological analysis. A graph is a set of nodes and edges (linking nodes.) The edge file lists all the graph's edges (DO NOT name this file edges.txt or dedges.txt). Each line in this file describes an edge with two node ids and an edge id, all separated by a comma. The name file lists all the graphs nodes. Each line describes a node with a node id and a node name separated by a comma—a pair of double quotation marks encloses the node name. How you generate these inputs is up to you, but I generally use a database tool to organize the system's nodes and edges. A query was written to assemble the output data and said query's results were saved to the two appropriately formatted file. Once you have these input files, the analyses may proceed.

Console Functions

Automated Analysis Tool

This tool automatically runs several analyses and organizes the results into a single file. Each analysis also prints out their respective results and files for use in other functions. To run the tool from your system's console type: `python step_one.py`. One may also *double-click* the tool's icon if you are using a windowing operating system. Initially, the user is requested to provide the name of the node or name file and the name of the edge file. After some computation, the user is asked to decide whether or not to partition the graph. If the graph is to be partitioned, the user will be asked for the smallest and largest number of partitions to partition the graph into. The graph will be partitioned for values within this input range.

There are several data files produced by the automated analysis tool. They are described as follows.

- `Graph.metis` is the graph reformatted for input to METIS™.
- `cc.txt` associates a node id with a connected component of the graph. Each connected component represents a set of nodes that a connected to each other by undirected paths.
- `Scc.txt` associates a node id with a strongly connected component of the graph. Each strongly connected component (SCC) represents a set of nodes connected to each other by directed paths. This file also contains additional information about the graph. This data, in order of appearance, consists of the graph's articulation points, bridges, SCC assignments, SCC spanning edges (spans). Node ids, and not node labels are used in this file.
- `D_centrality.txt` and `U_centrality.txt` are files that contain centrality calculations for the directed and undirected versions of the graph. If the system is not conceptualized as a directed graph, ignore all directed graph calculations. Each file contains the number of graph diameters. A diameter is a shortest path between two nodes. The number of nodes, and node ids associated with their betweenness and closeness centralities.
- `Summary.txt` is a summary of the preceding files. The fields are: node ID, CC, SCC, Art {1: node is an articulation point, 0: otherwise}, Bridge {-1: node is incident to a bridge's tail, 1: ... bridge's head}, Span {-1: node is incident to a span's tail, 1: ... span's head}, UCC (undirected closeness centrality), UBC (undirected betweenness centrality), DCC (directed closeness centrality), DBC (directed betweenness centrality), `bgcs.d (bridge group centrality score for directed betweenness), sgcs.d (span group centrality score for directed betweenness), bgcs.u (bridge group centrality score for undirected betweenness), and sgcs.u (span group centrality score for undirected betweenness).`
- The METIS™ output is saved for each partitioning, and the file is named `graph.metis.part.#`. The partition assignments are also summarized and saved in `x_partitions.txt`. `partition_eval.txt` contains a column for each partitioning in the range specified earlier, a column for node degree, and a column for the node's id. In the partition-specific columns is a listing of the

fraction of a node's neighbors in the same partition as the node. The last row of the file contains the number of nodes separated by the partitioning. In other words, twice the number of edges cut by the partitioning.

Individual Tools

Each of the tools automated by the `step_one.py` and `step_two.py` can be run individually, but they are somewhat interdependent so they should be run in the order specified in the automation programs. Here are some excerpts from the automation code (each function appears after the word 'import'):
Step One:

- **`#copy them (input files) to xnodes.txt and xedges.txt`**
- `print "Converting from Access to Metis > graph.metis, node_map.txt, edges.txt, dedges.txt"`
- `import` **`ac2met`**
- `print "Converting from Access to BioNet > graph.html"`
- `import` **`ac2bnt`**
- `print "Looking for connected components > CC.txt"`
- `import` **`cc`**
- `print "Looking for strongly connected components, articulation points, and bridges. > SCC.txt"`
- `import` **`scc`**
- `print "Calculating Centralities > D_centrality.txt, U_centrality.txt"`
- `import` **`get_centrality`**
- `import` **`get_centralityU`**
- `import` **`x_SUMMARY`**

The graph can be partitioned with METIS™ after running `ac2met.py`. This is accomplished by running `step_two.py`.

There are other tools not included in the automation. They should be run after the automated tools are run once. These include a spectral graph partitioner (**`ks_part3.py`**) and a graph-clustering tool (**`ccluster.py`**). Both tools seek to identify modularity in the system. `ks_part3.py` reads the xedges.txt file (a copy of your edges file) and asks the user to input the range of partitionings required. The results are summarized into x_partitions.txt and partition_eval.txt as before with the automated METIS™ partitioner. These should be renamed to distinguish them from those files produced by the automated partitioning tool. `ccluster.py` requires no user input. It produces an output file called `cClust.txt`. This file lists node ids and their cluster assignments. If the cluster assignment = -1, then this node could not be assigned to a cluster. The clustering tool is designed to find the optimal number of clusters, so there is only one cluster assignment, and no user input.

If you are running the automated analysis tool and it crashes due to memory error your work can still be resumed at the crash point. Usually a memory error occurs when the number of nodes passes 1400 and the number of edges passes 2300. The error usually happens between subprograms, so you can start up where it left off by figuring out which tools have not been run yet and manually running them. For example, if the last files that was produced was U_centrality.txt, then x_SUMMARY.py has not been run. Type python x_SUMMARY.py at the command

Examples

Finding articulation points and assessing modularity in a system

The first step is to prepare your abstraction of the system in graph form. Assuming your system is a protein interaction network, start by compiling a list of graph nodes. Assume each protein is a unique node. The node list should contain each protein's name and a unique id. Next, compile a list of graph edges. An edge will represent a mutual interaction between two proteins. In the edge list, record the ids of the two proteins interacting and a unique id for each interaction. In this example the interactions are mutual and represented by undirected edges, so the order of the node ids is not important. For a directed edge, place the id of the node incident to the tail of the edge first.

Run the step_one.py program and input the file names of the node and edge files when asked. Decline the request to perform the partitioning. Running step_one.py will produce many results but the file that contains the list of articulation points is scc.txt. Alternatively, this information is summarized in the summary.txt file, where a '1' designates articulation points. The articulation points are important because if they are removed, the system will break up into separated subsystems.

To determine the modularity of the system, run the ks_part3.py program. *I* will assume that the system has between 6 and 12 separable modules. When asked input 2 as the minimum number and 12 as the maximum number of partitions. After the program runs, look to the partition_eval.txt file for results. The optimal partitioning is assumed to be the partitioning with a significantly lower number of edges cut than other partitionings when considering the trend in edge-cut as number of partitions grow. To do this, first calculate the edge-cut for each partition by dividing the last number in each column by 2. Next plot the edge-cuts for all the partitionings, and look for a significant negative deviation from the trend. When the optimal number of partitions is determined look to the x_partitions.txt file. The node id assignments to partitions for the n^{th} partitioning in the range of calculated partitions are in the n^{th} column of this file. The node is the same partition can be grouped for other analysis. These partitions are intended to represent modules because of their relative isolation from the system.

GUI Interface

Even though the various analyses can be done independently of the GUI and interpretation of results done in a spreadsheet program or database, the GUI was designed to tie everything together and provide an additional view of the cold, hard numbers. Some things are picked up on faster with a visual display versus a numerical one. For instance, looking at the displayed graph one can immediately see errors in the graph, such as missing edges or nodes. Other things such as completely isolated sub graphs can also be seen and one can get an indication of how modular the system is prior to quantitative modularity assessment. Finally, the presentation of analyses results in a concise manner and facilitating analysis refinement are the main motivations in designing this GUI.

Displaying a graph

- Open the GUI by running `grapher_v09.py`.
- Choose File > Open.
- In the Open File window, type the full file names with extensions for the node and edge files.
- Click the Open button.

If the files you typed cannot be found in the running directory, error messages appear in the system console window and the Open File window does not close. Once the graph is opened properly, it is displayed with random positioning of the nodes, and all node labels ( ) are turned on.

Moving a node:

- Left-click over the node and drag to reposition.

Magnifying a region:

- Choose Scene > Fisheye.
- Hold down the right-cursor button, and drag the cursor to make the lens appear.
- Right-drag it over the graph to distort the region.

  The nodes will move away from each other as if a fisheye lens was magnifying the region, but the node sizes will remain constant, and edge positions will not be updated.
- To turn the lens off, choose Scene > Fisheye.

Choosing a layout for the graph

There are three main types of graph layout available: Kinetic, undirected Multidimensional Scaling, and directed Multidimensional Scaling. After each you may wish to resize the main window and center the graph in the window.

Centering the graph:

- Choose Scene > Center

Layout based on Multidimensional Scaling of the undirected graph:

- Choose Scene > MDS layoutU

The console window will show the status of the operation.
The MDS layouts cannot be used until the analysis tool has been run. The MDS tool uses a distance matrix calculated during the analysis.

Layout based on Multidimensional Scaling of the directed graph:

- Choose Scene > MDS layoutD

See undirected MDS info.

Layout based on Kinetics:

- Choose Scene > Relax

The nodes will dynamically adjust their positions relative to each other. Nodes have a universal repulsive force between themselves and an attractive force along the direction of the edge. When these forces equilibrate, the layout is assumed optimal. While the system is being relaxed the background is green. During relaxation the nodes can still be selected and moved with the cursor.

Adjusting Kinetic layout via its parameters:

- Choose Scene > Kinetics.
- In the Kinetics… window, adjust the parameter you wish.
- Click Update button.

Increasing [node] *Repulsion* and *Nat*[ural] *length* [of the edge] will spread out the layout. Increasing the *Spring const* value will contract the layout. The number in square brackets beside the field label is the current state of the field's parameter in the relaxation process.

Selecting Nodes

Nodes may be selected in three ways: by dragging a box around them with the cursor, selecting then from a list, or by adjusting the activated/deactivated lists.

Region selection [Box]:

- Choose Select > Box.
- Left click in the window and drag the cursor so that the selected region covers the node(s) you want to select.

List selection:

- Choose Select > List.

A List Select window appears, listing the names of all nodes and their status {1; label is visible, 0; otherwise}.

- In the List Select window, choose the nodes you want to select with the cursor.
- Click Select.

Activating or deactivating a node:

This window is used to indicate which nodes are to be included in a new graph that will be saved or reanalyzed.

- Choose Select > Activate?

- In the Activation / Deactivation window, select the node(s) you wish to reassign.
- Click the appropriate direction {<; activate, >; deactivate} button.
- When you are finished, click the Set button.

Labels

To remove all labels:

- Choose Select > Label > Hide all.

To show all labels:

- Choose Select > Label > Show all.

To show just the selected labels:

- Choose Select > Label > Show selected.

To remove only the selected labels:

- Choose Select > Label > Hide selected.

Analysis Tools

Once a graph is displayed, the analyses can begin. All the analyses are run at once. In fact the process is almost identical to Step_one.py (see, above, **Console Functions > Automated Analysis Tool**). Partitioning is offered as an option but results are not displayed in the GUI. Clustering can be initiated from the GUI, and the results are displayed by coloring nodes according to their cluster assignment.

To run automated analysis tools:

- Choose Analysis > Run all.

The progress is indicated in the system console, and when it is complete all results are available for visualization.

- In the Load Data window, confirm the node and edge files' names and click the Open button.

Once this is complete the node's sizes are automatically adjusted to indicate the directed betweenness centrality.

To cluster a graph:

- Choose Analysis > Cluster

Displaying analysis results

Node Properties

To view articulation Points:

- Select Scene > DataViz > Show Articulation Points

Articulation points are colored red and all other nodes gray.

To view directed Betweenness centrality:

- Select Scene > DataViz > D Betweenness

Bigger nodes have higher betweenness scores, which means they are more *central* in the network. This type of centrality indicates the potential of a node to act as a key gatekeeper for interaction between different parts of the network.
This score is qualified by the word 'directed'. This means that only directed paths were used in its calculation. If your graph is undirected, please ignore this data.

To view undirected Betweenness centrality:

- Select Scene > DataViz > U Betweenness

This score ignores the directionality of the graph's edges. See **directed Betweenness centrality** section above for more general information.

To view directed Closeness centrality:

- Select Scene > DataViz > D Closeness

Bigger nodes have smaller closeness centrality scores, which means they are more central to the network. This type of centrality indicates the potential of a node to influence other nodes.

To view undirected Closeness centrality:

- Select Scene > DataViz > D Closeness

This score ignores the directionality of the graph's edges. See **directed Closeness centrality** section above for more general information.

To reset node sizes to a uniform size:

- Select Scene > DataViz > None

Group Properties (Modularity assessment)

To show the graph's Bridges and Spans:

- Select Scene > DataViz > Show Bridges | Spans.

Bridges will be colored pink, spans colored , and if an edge is both it will be colored orange. A span is an edge linking two SCCs.

To show the graph's Connected Components:

- Select Scene > DataViz > Show Components.

To show the graph's Strongly Connected Components:

- Select Scene > DataViz > Show Strongly Connected Components.

This is only relevant if you have a directed graph.

Examples

Which nodes are critical and if they are disabled are there novel critical nodes or does the system break apart?

First, create your input files. Second, open grapher_v09.py. Display your graph by choosing File > Open and type in the input files' names. Click Open. Next, start the analysis by choosing Analysis > Run all. Monitor the analyses progress in the console window, and when you are asked to run the partitioner, enter No. When this is complete and the Load Data window appears, confirm the proper file names and click Open.

The graph is a jumble, so perform a layout operation by choosing Scene > MDS layoutU. Monitor this function's progress in the console. When its finished the graph is automatically adjusted for better viewing. There are probably too many node labels on the screen. To get rid of them choose Select > Label > Hide all.

Let's assume you want to investigate the betweenness centrality prioritization for the undirected version of your graph. To display this data select Scene > DataViz > U Betweenness. The critical nodes, according to this metric, are the bigger ones. Turn on the label for the biggest by choosing Select > Box. Left click in the window and drag the cursor so that the selected region covers the biggest node. Now you have selected a node, now to turn its label on choose Select > Label > Show selected. To more thoroughly analyze the results, look in the running directory for the summary.txt file. It contains all the results. See the console functions instruction above.

Display the graph's strongly connected components by selecting Scene > DataViz > Show Strongly Connected Components and notice how many there are with a significant amount of nodes. You may want to rearrange the node positions by left clicking on them and dragging the cursor.

This node might be very critical in your system, and a good way to destabilize it. Let's perform an experiment to see what happens to the graph and the centrality scores when it's deleted from the system. First choose Select > Activate? In the Activation / Deactivation window, select the node(s) you wish to deactivate. Click the > button. When you are finished, click the Set button. Now the graph must be saved prior to reanalysis. Choose File > Save. Now the graph without the node you deactivated along with it's incident edges is saved in the running directory as the original name appended to a version number. If you node file was nodes.txt, the saved graph will be named nodes_<<a number greater than 10>>.txt (example: nodes_12.txt).

Display the saved graph by opening it with File > Open. Run the full analysis, except for partitioning. Use the undirected MDS layout, and display the undirected betweenness centrality values. Visually check to see how the scores change. Display the strongly connected components and notice how many significant ones are left.

Index

activate, 141
articulation points, 141

betweenness, 135, 141, 142, 143
bgcs.d, 135
bgcs.u, 135
box, 140
bridge, 142

cc.txt, 135
cClust.txt, 136
ccluster.py, 136
centrality
  directed betweenness centrality, 135, 141
  directed closeness centrality, 135
  undirected betweenness centrality, 135
  undirected closeness centrality, 135
closeness, 135, 142
components, 142, 143

D_centrality.txt, 135, 136
deactivate, 141
directed
  Multidimensional Scaling, 139

edge file, 134, 135
edge-cut, 138

Fisheye, 139

Graph.metis, 135
graph.metis.part, 135
grapher_v09.py, 139, 143
GUI, 139

Kinetic, 139
ks_part3.py, 134, 136, 138

label, 135, 139, 140, 141, 143
layout, 139, 140, 143

MDS, 139, 140, 143
METIS, 134, 135
modularity, 133, 136, 138, 139

name file, 134, 135

partition, 135, 136, 138
partition_eval.txt, 135
Python, 134

Relax, 140

Scc.txt, 135
sgcs.d, 135
sgcs.u, 135
span, 135, 142
spectral, 136
step_one.py, 135, 136, 138
step_two.py, 136
Summary.txt, 135
  fields, 135

U_centrality.txt, 135, 136
undirected
  Multidimensional Scaling, 139

x_partitions.txt, 135, 136, 138

Appendix D: Developer Manual

The document that follows describes the design and implementation of a few selected functions. Not all are described here, but the code is laced with comments from the programmer. These comments provide an insight into what each operation does. The plentiful in-line commentary precludes the description of every function and operation in the following text. I suggest one start by reading the Python™ code in order of logical execution. Use this document for supplementary design insight, and suggested areas of improvement.

Contact: *Abraham Anderson abeand@socrates.berkeley.edu*

This work was done with the Hunt Lab at the University of California at San Francisco as part of the Ph.D. dissertation of Abraham A. Anderson, a student in the joint Bioengineering Graduate Group of U.C.S.F. & U.C. Berkeley. All rights reserved. © Abraham Anderson, 2002. This work was partly supported by grant # LSI-10041 from the California Life Science Informatics Technology Program, and by grant # r25-gm56847 from the NIH National Institute of General Medical Sciences.

Codebase

Most of the code is written in Python™. This language was selected because of its ease-of-use and portability across different operating systems. It does have the detraction of being relatively slow compared to other programming languages. This may be due to the fact that it is an *interpreted* language (based on C++). As it is an offshoot of C++, it should be easy to extend and embed in a combination with C++ programs. This will come in handy when functions are rewritten in C++ for speed.

The GUI was also written in Python™ and Tk™. Tk™ has been very easy to develop a GUI in and is compatible with the Python™ code. Tk™ is also platform neutral. At this point there is no compelling reason to rewrite the GUI in another language. If pushed, I would suggest a Java™ applet GUI, but Tk™ applets is increasing in use (plug-in required, so, extra install burden for user.)

Analysis Tools

CC.py

This tool identifies all components of a graph. The graph is read from the node_map.txt file. This dictionary [and others] is created by ac2met.py. It is unpickled and a reverse dictionary is created which maps new(internal) node ids to old(external) ids. The dictionary mapping a node to its neighbors is unpickled from edges.txt. To assign nodes to connected components a list of nodes is generated from the node dictionary, a DFS is initiated for a node and as a node is seen it is removed from the list and given the component id of the root node. When no more nodes are reachable by the DFS algorithm, and the list is not empty, this process is repeated after incrementing the current component id.

SCC.py

This tool identifies edges as bridges and/or spans, identifies nodes as articulation points, and assigns nodes to strongly connected components. The pseudo-code for these tasks is shown below in three segments. Each operation has some steps in common with the others, so they are interleaved in the actual coding. The spans are calculated at the end by checking if each edge's incident nodes are in different SCCs.

Box 1. Finding SCCs Given G(V,E)

```
1. DFS(G); keep last seen time f(v) for each node v.
2. get Gt =G transposed.
3. DFS(Gt), in DFS search each v in order of decreasing f(v).
4. output each tree found in step 3 as a new SCC.
```

Box 2. Finding Bridges Given G(V,E)

```
compute low(v) for each v.
for each v do:
 for each child u of v do:
  if low(u) = pre(u) then:
   (v,u) is a bridge.
```

---Computing low(v)---------------------

```
explore(v):
 low(v) ← pre(v) # pre=time first seen v
 for each e(v,w):
  if w not visited yet:
    explore(w)
    low(v)←min(low(v),low(w))
  else:  low(v) ← min(low(v),pre(w))
```

Box 3. Finding Articulation Points

```
Given G(V,E)
  compute low(v) for $v \subset V$.
  for each v do:
    if v is the root & has >1 child:
      v is an articulation point.
    else: for each child u of v do:
      if low(u)≥pre(v):
        v is an articulation point.
```

Centrality Calculations

Closeness and Betweenness centrality are calculated simultaneously. Both metrics rely on the calculation of the graph's diameters or shortest paths between node pairs. Dijkstra's shortest-path algorithm is modified to handle situations where there is more than one shortest path between two nodes. As each diameter is found, a count in incremented for each node in the diameter. This count is used to calculate the betweenness score for each node after all diameters are found. The process previously described, results in a matrix of minimal internode distances. This distance matrix was used to calculate the closeness centrality score. The distance matrix is pickled for later use by other tools.

METIS™ Interface

Look to the METIS™ manual for thorough details on its use. A library of METIS™ functions is available for importing into C++ code. It can potentially be embedded in Python™. Also available, are several precompiled tools. It is one of these precompiled tools (pmetis.exe) that is used to partition input graphs. A command is formulated in memory by step_two.py and a system call is issued to run the command.

Spectral Graph Partitioning

As opposed to the *balanced* partition determined by METIS™, a spectral graph partitioner was developed to generate natural or *unbalanced* partitions. This process starts by unpickling the edge-mapping dictionary. The eigenspace for a special adjacency matrix is calculated. The programmer has a choice in the type of matrix used: unsigned adjacency matrix, degree matrix, Laplacian matrix, and Disconnection matrix. Currently the Disconnection matrix is used in calculating the eigenspace, but it is quite similar to the Laplacian matrix, another good choice. Once the eigenspace is calculated the 3 *maximal* (in general this means closest to 0) eigenvalues are selected along with their respective eigenvectors. In order to bisect a graph, the eigenvector for the maximal eigenvalue is selected and each node is associated with a scalar from this vector. The nodes are then assigned to two groups according to the sign of their associated scalar. To create an arbitrary number of partitions, this method uses recursive sectioning. Since the three eigenvectors are used, eight sections can be created per iteration. A partitioning scheduler was developed to decide whether 8, 4, 3, or 2 new partitions are needed at each step in the process of attaining the desired number of partitions. Sometimes, there are several sections and not all of them need to be partitioned. Some sections will partition better than others. The quality of a partitioning is taken to be the number of edges cut—smaller is better. The best section to partition is the one with the best maximal eigenvalue. Briefly, the entire method is as follows: decide how many partition(s) needed this round, sort sections in order of maximal eigenvalues, chose the best sections to partition and partition, repeat till desired number of sections is achieved.

Graph Clustering

This tool aims to group graph nodes into clusters that are maximally isolated from the rest of the graph, and are maximally connected to each other. This is achieved by a hierarchical clustering algorithm, where clusters are assessed as they grow. If a cluster is found to have the properties previously stated, then its growth is halted and no longer considered in the rest of the clustering process. The programmer has implementation options in the clustering function, and the cluster assessment function.

The closest two clusters are merged at each phase to form a new cluster. This process repeats until there is one cluster. Closeness can be determined in three ways, each of which is implemented as options in the tool. The three choices are minimal internode distance, maximal internode distance, and average internode distance. Minimal internode distance is the default choice. Regardless of closeness choice, this step is the slowest in the clustering program. It should be optimized, by blocking or rewritten in C++ and embedded into the Python™ code.

Each newly formed cluster is assessed for its cohesiveness or modularity. There are three choices for determining cohesivity: density ratio, and two versions of the intracluster edge observation likelihood. The density ratio is ratio of the cluster density and the overall graph density. A lower threshold of 7 is suggested for a suitably cohesive cluster if the density ratio is used. The other option for assessing cohesivity calculates the probability of observing at least the number of edges within a cluster. L = # Total Edges, g= # Total Nodes, gs= #Cluster Nodes

$$\sum_{k=q}^{\min(1+L,1+gs(gs-1)/2)} \binom{L}{k} \binom{g(g-1)/2-L}{gs(gs-1)/2-k} \div \binom{g(g-1)/2}{gs(gs-1)/2}$$

The threshold for this measure is at most 0.00025. This value seems to be graph dependent. In order to calculate this probability, the combinatorics had to be rewritten, hence the two versions. Initially the division of factorials caused an overflow error. This division was algebraically simplified and rewritten in a new combinatorial function, fixing the overflow problem. The factorial F(a) is calculated with Sterling's formula for a > 10. The error in Sterling's for a=10 is 0.8%. For a <=10 the factorial is calculated by recursive multiplication.

The GUI

The graphical user interface (GUI) will accept an edge file and node file as inputs in order to display the graph and node labels. Initially, the graph's nodes are randomly positioned. The Kinetic layout, which will be described later, is the only automated layout option available prior to graph analysis. This is because the other layout option, Multidimensional Scaling (MDS) uses data structures pickled during the automated graph analysis.

The clustering algorithm described previously is written into the GUI, but it does not yet have its own parameter console from which a user may choose clustering options. The clustering currently proceeds with the default parameters and colors the nodes according to cluster assignment.

Layout

MDS

This layout uses the eigenspace of the graph's distance matrix (unpickled from [U,D]_dist.txt, a file created during the centrality calculations) to assign 2D Cartesian coordinates to the nodes. Two sets of coordinates are calculated and stored the first time an MDS layout is requested. This is to avoid unnecessary computation in the event of multiple layout toggling.

The algorithm begins by unpickling the directed and undirected distance matrices. These unpickled matrices use the original node ids. Sometimes the range of node ids is not complete; there can be skipped integer ids. To deal with this contingency a new range of complete, ungapped, ids is created for the nodes. This id range is used to create the new 2D distance matrices. The eigenspace is calculated and the three largest eigenvalues are found. Two of the three are selected to define 2 coordinate axes. Their eigenvectors are used to assign coordinates to the nodes. For example, let *a* be a node and *d* be its distance vector (the distances to all other nodes from *a*). Let *e* be an eigenvector representing the x-axis. *a*'s x-coordinate is the dot product, $e \cdot d$. Once this is done the coordinates are scaled to fill a 600x600 pixel space. In future a console could be made to allow a user to select eigenvalues for the principal axes from a sorted list. Currently eigenvalues 2 and 3 are used, because once upon a time eigenvalue 1 caused an extremely unpleasant coordinate distribution.

Kinetic

This layout models the graph as a mass and spring system. The nodes are modeled as masses with inter-node repulsion, and the edges are modeled as springs with spring constant and natural length. First the contribution of the node repulsion is calculated from the distance between nodes and a scaling factor. This contribution is stored in a temporary array. Then the effect of the spring's deformation from its natural length and the spring constant is calculated and combined with the temporary array values to generate the new coordinates. Calculating the internode effects is the most expensive step. Compartmentalizing the simulation space to reduce the number of neighbor calculations, or rewriting the current algorithm in a C++ extension could improve it. After several iterations, the node coordinates reach an equilibrium at which the simulation can be stopped. This equilibrium state is expected to reduce edge overlap, bring together neighboring nodes, and reduce node crowding.

Data Visualization

After analysis is completed, the analysis results are imported and available for display. Values are stored in global arrays, and all grouping values are consolidated into two global data structures, *groups & sgroups*. In order to keep track of which graph is associated with the analysis results, a file is kept with the name of said graph. The file is found in the current working directory and is called *xstatus.txt*. The UserManual provides detailed explanation of the various data views.

Aside from the graphical visualization of data, there are rudimentary tabular visualizations. The table classes can be reused for advanced tabular displays. Class *UniListBox* demonstrates how to quickly display a list with multiple coordinated fields. Class *TableDialog* demonstrates how to display multiple columns and rows of coordinated data. This class uses a label grid instead of internal listing capabilities, thus its slowness to display. Class *DoubleListBox* demonstrates how to shuffle data between two separate lists and perform an action based on the final ordering.

Appendix E: Computer Executable Code

```
:::ac2bnt.py:::

#Author: Abraham Anderson
#© 2002 The Regents of the University of California

file = open("c:/my documents/bin/xedges.txt",'r')
file1 = open("c:/my documents/bin/xnodes.txt",'r')
import string
name = {} #this holds the title of each node
for line in file1.readlines():
      line = line.rstrip()
      id = line.split(',')[0]
      na = line.split(',')[1][1:-1]
      na = string.replace(na,' ','_')
      na = string.replace(na,'-','+')
      na = string.replace(na,'/','\\')
      name[id] = na
file1.close()

edge = []
for line in file.readlines():
      edge.append(line.rstrip())
file.close()
nedges = len(edge)
directed = {} #this dictionary preserves the directed edges
i=0
for e in edge:
    d = e.split(',')

    a = d[0]
    b = d[1]
    if a in directed.keys():
        prior = directed[a]
        prior.append(b)
        directed[a] = prior
    else:
        directed[a] = [b]


nnodes = len(name)
print "%d nodes & %d edges" % (nnodes,nedges)

nodes = []
for e in directed.keys():
    nodes.append(e)
nodes.sort()

outstring = ""
end = ""
for e in nodes:
      for t in directed[e]:
            outstring = string.join([outstring,name[e]+'-
'+name[t]+','],'')
      outstring = string.join([outstring,end],'')
outstring = outstring[0:-1]

file2 = open("c:/my documents/bin/graph.html",'w')
#print outstring
head = "<HTML><HEAD><TITLE>Template graph viewer from
BioNet</TITLE>\n<META content=\"text/html; charset=unicode\" http-
equiv=Content-Type>\n<META content=\"MSHTML 5.00.2614.3500\"
name=GENERATOR></HEAD>\n<BODY>\n<table><tr><td width=400
valign=\"top\">\n<pre>\n</pre><APPLET code=Graph.class height=600
width=800 archive=\"hsql.jar\">\n   <PARAM NAME=\"center\"
VALUE=\"\">\n     <PARAM NAME=\"edges\" VALUE=\"%s\">" % outstring
file2.write(head)
file2.write("\n      alt=\"Your browser understands the &ltAPPLET&gt
tag but isn't running the \n      applet, for some reason.\"      Your
browser is completely ignoring the \n      &ltAPPLET&gt tag!
</APPLET>\n<hr><b>Holding down \'Shift\' while selecting a node fixes
it to the screen. Control makes the selected node
invisible.</b><hr></td><td valign=\"top\" width=400><font size=-2>
\n<br>\n</font></td></tr></table></BODY></HTML>")
file2.close()
```

:::ac2met.py:::

```
#Author: Abraham Anderson
#© 2002 The Regents of the University of California

file = open("c:/my documents/bin/xedges.txt",'r')
file2 = open("c:/my documents/bin/graph.metis",'w')
file3 = open("c:/my documents/bin/node_map.txt",'w')
file4 = open("c:/my documents/bin/edges.txt",'w')
file5 = open("c:/my documents/bin/dedges.txt",'w')
import pickle

edge = []
for line in file.readlines():
      edge.append(line.rstrip())
file.close()
nedges = 0
direct = {}   #this one maps both parents and children (undirected
edges)
directed = {} #this dictionary preserves the directed edges
nmap = {}
i=0
for e in edge:
    d = e.split(',')

    if d[0] in nmap.keys():
      a = nmap[d[0]]
    else:
      i += 1
      nmap[d[0]] = i
      a = i
    if d[1] in nmap.keys():
      b = nmap[d[1]]
    else:
      i += 1
      nmap[d[1]] = i
      b = i

    if a in direct.keys():
      if b not in direct[a] and b != a:  #no repeats and no loops
            prior = direct[a]
            prior.append(b)
            nedges+=1
            prior.sort()
            direct[a] = prior
            directed[a] = prior
    elif b != a:
        direct[a] = [b]
        directed[a] = [b]
        nedges+=1

pickle.dump(directed,file5)
file5.close()
#del directed,file5
nnodes = len(nmap)
pickle.dump(nmap,file3)
file3.close()
#del nmap,file3


# this step makes the edges undirected
for bb in direct.keys():
      for cc in direct[bb]:
            if cc in direct.keys() and cc != bb:
                  if bb not in direct[cc]:
                        prior2 = direct[cc]
                        prior2.append(bb)
                        nedges+=1
                        prior2.sort()
                        direct[cc] = prior2
            elif cc != bb:
                  direct[cc] = [bb]
                  nedges+=1



print nedges, nedges/2
print "%d nodes & %d edges" % (nnodes,nedges/2)
outstring = "%d %d\n" % (nnodes,nedges/2)
file2.write(outstring)

import string
nodes = []
for e in direct.keys():
    #nodes.append(string.atoi(e))
    #now nodes are integers: "a,b = i"
    nodes.append(e)
nodes.sort()
import fpformat
for e in nodes:
    #print string.join(direct[fpformat.fix(e,0)])
    #outstring = "%s\n" % string.join(direct[fpformat.fix(e,0)])
      outstring = ""
      end = "\n"
      for t in direct[e]:
            gg = fpformat.fix(t,0)
            outstring = string.join([outstring,gg])
      outstring = string.join([outstring,end])
      #print outstring
      file2.write(outstring)
file.close()
file2.close()

#print nmap
pickle.dump(direct,file4)
file4.close()
```

:::cc.py:::

```
#Author: Abraham Anderson
#© 2002 The Regents of the University of California

#Finds Connected Components.
import pickle
import string
import fpformat

#dictionary[old]=>new
file3 = open("c:/my documents/bin/node_map.txt",'r')
nmap = pickle.load(file3)

nmap2={} #new > old
for y in nmap.keys():
    nmap2[nmap[y]] = y
file3.close()

#dictionary[from]=>to
file4 = open("c:/my documents/bin/edges.txt",'r')
direct = pickle.load(file4)
file4.close()

source = 1

nodes = nmap2.keys()
T=[]+ nodes
c = 0
cc = {}

def seek(a,c):
    cc[a]=c
    T.remove(a)
    if a in direct.keys():
        for d in direct[a]:
            if d in T:
                seek(d,c)
    return
```

```
while len(T)>0:
    source = T[-1]
    seek(source,c)
    c+=1

print "Components: ",c
#put in nice list
ccomps = []
for a in range(0,c):
    temp = []
    for b in cc.keys():
        if cc[b] == a:
            temp.append(nmap2[b])
    ccomps.append(temp)
#print ccomps
import fpformat
output = ""
for a in range(0,c):
    text =
'['+fpformat.fix(a,0)+']'+':'+fpformat.fix(len(ccomps[a]),0)+", "
    output = string.join([output,text])
print "Lengths: ",output

ofile = open("c:/my documents/bin/CC.txt","w")
for a in cc.keys():
    ofile.write("%s\t%d\n" % (nmap2[a],cc[a]))
ofile.close()
print cc
```

:::ccluster.py:::

```
#Author: Abraham Anderson
#© 2002 The Regents of the University of California

#Ccluster.py

#read in undir distance matrix (U_dist.txt),
import pickle
file2 = open("c:/my documents/bin/U_dist.txt",'r')
udist = pickle.load(file2)
file2.close()
#... directed (dedges.txt) and undirected (edges.txt) graphs
 #dictionary[from]=>to
file4 = open("c:/my documents/bin/dedges.txt",'r')
child = pickle.load(file4)
file4.close()
 #dictionary[from]=>to
file4 = open("c:/my documents/bin/edges.txt",'r')
neighbor = pickle.load(file4)
file4.close()
 #dictionary[old]=>new
file3 = open("c:/my documents/bin/node_map.txt",'r')
nmap = pickle.load(file3)
nmap2={} #new > old
for y in nmap.keys():
    nmap2[nmap[y]] = y
file3.close()
```

```
def Fact(a):
    import math
    if a == 0:
        return(1)
    if a < 10: #error for Stirling at 10 is 0.8%
        c=1
        while a > 0:
            c = c*a
            a = a-1
        return(c)
    answer =
round(math.pow((2*2.718281825/(a+1)),0.5)*1/math.exp(a+1)*pow(a+1.0,a+1
.0))
    return(answer)
def Comb(a,b):
    c = Fact(a)/Fact(b)/Fact(a-b)
    c_str = "%f" % c
    if c_str == '-1.#IND00':
        c = 0.9999999999
    return(c)
def sComb(a,b):
    mx = max(b,(a-b))
    mn = min(b,(a-b))
    num = 1.0000000 #recursive mult
    for i in range(mx+1,a+1):
        num *= i
    c = num/Fact(mn)
    #print a,b,c
    c_str = "%f" % c
    if c_str == '-1.#IND00':
        c = 0.9999999999
    return(c)

L = 0  #total number of edges in dir graph
for i in child.keys():
    L = L + len(child[i])
g = len(neighbor.keys()) #number of nodes in undir graph

def Cohesiveness(A):
    #this doesn't work because of the overflow errors
    #pass in dictionary for directed and undirected graph with A
    global neighbor, child, L, g
    #A = list of nodes like, [a,b,c,d]
    q = 0 #number of edges within dir cluster A
    for i in A:
        if i in child.keys():
            for j in child[i]:
                if j in A:
                    q = q + 1
    gs = len(A) #number of nodes within undir cluster A
    pf = 0
    for k in range(q,min(L,1+gs*(gs-1)/2)):
        pf = pf + Comb(L,k)*Comb(g*(g-1)/2-L,gs*(gs-1)/2-k)/Comb(g*(g-
1)/2,gs*(gs-1)/2)
        #print pf, Comb(L,k)
    return(pf)
```

```
def Cohesiveness2(A):
    #ratio of cluster density to graph density
    #density = 2L/g(g-1)
    global L, g

    density1 = 2.0*L/g/(g-1)

    q = 0 #number of edges within dir cluster A
    for i in A:
        if i in child.keys():
            for j in child[i]:
                if j in A:
                    q = q + 1
    gs = len(A) #number of nodes within undir cluster A

    density2 = 2.0*q/gs/(gs-1)

    return(density2/density1)

def Cohesiveness3(A):
    #b4 this didn't work because of the overflow errors
    #but now I am trying to simplify the factorial math involved in
calculating lCk
    #pass in dictionary for directed and undirected graph with A
    global neighbor, child, L, g
    #A = list of nodes like, [a,b,c,d]
    q = 0 #number of edges within dir cluster A
    for i in A:
        if i in child.keys():
            for j in child[i]:
                if j in A:
                    q = q + 1
    gs = len(A) #number of nodes within undir cluster A
    pf = 0
    for k in range(q,min(1+L,1+gs*(gs-1)/2)):
        pf = pf + sComb(L,k)*sComb(g*(g-1)/2-L,gs*(gs-1)/2-
k)/sComb(g*(g-1)/2,gs*(gs-1)/2)
    return(pf)

def MinDist(A,a): #0:min,1:max,2:avg
    global udist
    avgc = [-1,-1,500000000]
    maxc = [-1,-1,500000000]
    min = [-1,-1,500000000]
    for i in A:
        for j in A:
            avg = 0
            max = [-1,-1,0]
            if j != i:
                for k in i:
                    for l in j:
                        d = udist[k-1][l]
                        if d > max[2]:
                            max = [A.index(i),A.index(j),d]
                        if d < min[2]:
                            min = [A.index(i),A.index(j),d]
                        avg += d
                if max[2]<maxc[2]:
                    maxc = max
                avg = avg/len(i)/len(j)
                if avg < avgc[2]:
                    avgc = [A.index(i),A.index(j),avg]
    if a == 0:
        return(min[0],min[1])
    if a == 1:
        return(maxc[0],maxc[1])
    if a == 2:
        return(avgc[0],avgc[1])




#main
R = []
ta = []
for i in neighbor.keys():
    ta.append([i])
R.append(ta)
ta = []

S = []
t = 0


while len(R[t]) > 1:
    t += 1
    Cq = []
    [i,j] = MinDist(R[t-1],0)
    Cq = R[t-1][i] + R[t-1][j]
    T = [] + R[t-1]
    T.remove(R[t-1][i])
    T.remove(R[t-1][j])
    C = Cohesiveness3(Cq)
    print C
    if (C <= 0.00025) & (len(Cq) > 1): #with cohesiveness2 use
thresshold of 7; w/3 use?
        S.append(Cq)
        R.append(T)
    else:
        T.append(Cq)
        R.append(T)


#print R[-1],S
file = open("c:/my documents/bin/cClust.txt",'w')
file.write("ID,cluster(-1:r; n:s)\n")
for o in R[-1]:
    for p in o:
        file.write(nmap2[p]+",-1\n")
q = 0
for o in S:
    for p in o:
        outstring = ",%d\n" % q
        file.write(nmap2[p]+outstring)
    q+=1
file.close()
print "... done.\n"
```

:::get_centrality.py:::

```
#Author: Abraham Anderson
#© 2002 The Regents of the University of California

#this program gets ths centralities for a directed graph!
# and pickles a distance matrix

import pickle
import string
import fpformat

#dictionary[old]=>new
file3 = open("c:/my documents/bin/node_map.txt",'r')
nmap = pickle.load(file3)

nmap2={} #new > old
for y in nmap.keys():
    nmap2[nmap[y]] = y
file3.close()

#dictionary[from]=>to
file4 = open("c:/my documents/bin/dedges.txt",'r')
direct = pickle.load(file4)
file4.close()

#pre-init
Vt = direct.keys()
V = []
T = []
for a in Vt:
    V.append(a)
V.sort()
N = len(V)
INF = 1000 #should be infinity


#calc min path for dest
def seek(a, b, p={}, path = []):
    path = [nmap2[a]]+path
    if a == b:
        return [path]
    if not p.has_key(a):
        return []
    paths = []
    for c in p[a]:
        #if c not in path:
            newpaths = seek(c,b,p,path)
            for newpath in newpaths:
                if newpath not in paths:
                    paths.append(newpath)
    return paths


def mDijk():
    #Dijkstra's algorithm (modified to handle multiple equal length
paths)
    #init
    Distances = []
    Dfreq = {}
    Cnes = {}
    for vert in nmap.keys():
        Dfreq[vert] = 0
        Cnes[vert] = 0
    diameters = 0

    for source in V:
        T=[]
        T=T+V #if node not in T then its marked
        d = {}
        for i in V:
            d[i] = INF
        d[source] = 0
        d[0] = INF

        p = {}
        j = 0

        #main
        while len(T) != 0:
            min = 0
            for a in T:
                if d[a] < d[min]:
                    min = a
            if min == 0:
                break
            T.remove(min)
            for a in direct[min]:
                if a in T:
                    if d[a] == d[min]+1:
                        tp = p[a]
                        p[a] = tp + [min]
                    elif d[a] > d[min]+1:
                        d[a] =d[min]+1
                        p[a]= [min]

        #count each node's appearances in graph diameters
        for dd in p.keys():
            s = seek(dd,source,p)
            for pth in s:
                diameters+=1
                pth = pth[1:-1]
                for o in pth:
                    if not Dfreq.has_key(o):
                        Dfreq[o]=1./len(s)
                    else:
                        Dfreq[o]+=1./len(s)
        #calculate closeness centrality: sum of distances to other
nodes
        del d[0]
        Distances.append(d)
        for dd in d.keys():
            Cnes[nmap2[dd]]+=d[dd]
    return diameters,Dfreq,Cnes,Distances
#end of mDijk()

ofile = open("c:/my documents/bin/D_centrality.txt","w")
import time
start = time.time()
#for x in range(0,50):
r = mDijk()
print time.time()-start,'s'
print "\ndiameters\n",r[0]
ofile.write("\ndiameters\n%d\n" % r[0])
print "N\n",N,"\nVertex betweenness Closeness"
ofile.write("N\n%d\nVertex betweenness Closeness\n" % N)
for dkey in r[1].keys():
    #print dkey , r[1][dkey], r[2][dkey]
    ofile.write("%s %f %f\n" % (dkey , r[1][dkey], r[2][dkey]))
ofile.close()
#saving dist library
#print r[3]
dfile = open("c:/my documents/bin/D_dist.txt","w")
pickle.dump(r[3],dfile)
dfile.close()

#diameter calc
diam_dist = []
for a in r[3]:
    b = diam_dist + a.values()
    diam_dist = b
diam_dist.sort()
if 1000 in diam_dist:
    c = diam_dist.index(1000)
else:
    c = 0
print "Graph diameter is %d/%d\n" % (diam_dist[c-1],diam_dist[-1])
```

:::get_centralityU.py:::

```
#Author: Abraham Anderson
#© 2002 The Regents of the University of California

#this program calculates centrality on the undirected graph!

import pickle
import string
import fpformat

#dictionary[old]=>new
file3 = open("c:/my documents/bin/node_map.txt",'r')
nmap = pickle.load(file3)

nmap2={} #new > old
```

```
for y in nmap.keys():
    nmap2[nmap[y]] = y
file3.close()

#dictionary[from]=>to
file4 = open("c:/my documents/bin/edges.txt",'r')
direct = pickle.load(file4)
file4.close()

#pre-init
Vt = direct.keys()
V = []
T = []
for a in Vt:
    V.append(a)
V.sort()
N = len(V)
INF = 1000 #should be infinity

#calc min path for dest
def seek(a, b, p={}, path = []):
    path = [nmap2[a]]+path
    if a == b:
        return [path]
    if not p.has_key(a):
        return []
    paths = []
    for c in p[a]:
        #if c not in path:
            newpaths = seek(c,b,p,path)
            for newpath in newpaths:
                if newpath not in paths:
                    paths.append(newpath)
    return paths


def mDijk():
    #Dijkstra's algorithm (modified to handle multiple equal length
paths)
    #init
    Distances = []
    Dfreq = {}
    Cnes = {}
    for vert in nmap.keys():
        Dfreq[vert] = 0
        Cnes[vert] = 0
    diameters = 0

    for source in V:
        T=[]
        T=T+V #if node not in T then its marked
        d = {}
        for i in V:
            d[i] = INF
        d[source] = 0
        d[0] = INF
        p = {}
        j = 0

        #main
        while len(T) != 0:
            min = 0
            for a in T:
                if d[a] < d[min]:
                    min = a
            if min == 0:
                break
            T.remove(min)
            for a in direct[min]:
                if a in T:
                    if d[a] == d[min]+1:
                        tp = p[a]
                        p[a] = tp + [min]
                    elif d[a] > d[min]+1:
                        d[a] =d[min]+1
                        p[a]= [min]

        #count each node's appearances in graph diameters
        for dd in p.keys():
            s = seek(dd,source,p)
            for pth in s:
                diameters+=1
                pth = pth[1:-1]
                for o in pth:
                    if not Dfreq.has_key(o):
                        Dfreq[o]=1./len(s)
                    else:
                        Dfreq[o]+=1./len(s)

        #calculate closeness centrality: sum of distances to other
nodes
        del d[0]
        Distances.append(d)
        for dd in d.keys():
            Cnes[nmap2[dd]]+=d[dd]
    return diameters,Dfreq,Cnes,Distances
#end of mDijk()

ofile = open("c:/my documents/bin/U_centrality.txt","w")
import time
start = time.time()
#for x in range(0,50):
r = mDijk()
print time.time()-start,'s'
print "\ndiameters\n",r[0]
ofile.write("\ndiameters\n%d\n" % r[0])
print "N\n",N,"\nVertex betweenness Closeness"
ofile.write("N\n%d\nVertex betweenness Closeness\n" % N)
for dkey in r[1].keys():
    #print dkey , r[1][dkey], r[2][dkey]
    ofile.write("%s %f %f\n" % (dkey , r[1][dkey], r[2][dkey]))
ofile.close()
#saving dist library
```

```
#print r[3]
dfile = open("c:/my documents/bin/U_dist.txt","w")
pickle.dump(r[3],dfile)
dfile.close()

#diameter calc
diam_dist = []
for a in r[3]:
    b = diam_dist + a.values()
    diam_dist = b
diam_dist.sort()
if 1000 in diam_dist:
    c = diam_dist.index(1000)
else:
    c = 0
print "Graph diameter is %d/%d\n" % (diam_dist[c-1],diam_dist[-1])
```

:::grapher_v09.py:::

```
#Author: Abraham Anderson
#© 2002 The Regents of the University of California

from Tkinter import *
from math import *

class UniListbox:  #with customizable info fields...
    def __init__(self,parent):
        top = self.top = Toplevel(parent)
        top.title("List Select")


        #action buttons
        frame3 = Frame(top)
        ba = Button(frame3,text="Select.", command=self.action)
        ba.pack(side=RIGHT)
        frame3.pack(side=BOTTOM, fill=X, expand=0)

        #list box
        frame = Frame(top)
        frame2 = Frame(top)
        lname = Label(frame, text="Node")
        lname.pack(anchor=NW)

        scrollbar = Scrollbar(frame2, orient=VERTICAL)

        self.listbox = Listbox(frame, selectmode=EXTENDED,
exportselection=0, yscrollcommand=scrollbar.set)

        scrollbar.config(command=self.listbox.yview)
        #scrollbar.pack(side=RIGHT, fill=Y)
        self.listbox.pack(side=LEFT, fill=BOTH, expand=1)

        frame.pack(side=LEFT, fill=BOTH, expand=1)

        #field1
        lname = Label(frame2, text="Label")
        lname.pack(anchor=NW)
        #scrollbar = Scrollbar(frame2, orient=VERTICAL)

        self.listbox2 = Listbox(frame2, width=1, selectmode=NONE,
exportselection=0, yscrollcommand=scrollbar.set)

        #scrollbar.config(command=self.listbox2.yview)
        scrollbar.pack(side=RIGHT, fill=Y)
        self.listbox2.pack(side=LEFT, fill=BOTH, expand=1)

        frame2.pack(side=LEFT, fill=BOTH, expand=1)

        #populate list
        global name, text_ids
        templist = name.keys()
        templist.sort()
        for a in templist:
            self.listbox.insert(END,name[a])
            if a in text_ids.keys():
                self.listbox2.insert(END,'1')
            else:
                self.listbox2.insert(END,'0')

        self.result = None

    def action(self):
        items = self.listbox.curselection()
        try:
            items = map(int, items) #integer index
        except ValueError: pass
        #print items
        items2=[]
        for a in items:
            items2.append(self.listbox.get(a)) #strings
        #print items2

        #place new labels
        global node_ids, text_ids, name
        for i in name.keys():
            if name[i] in items2:
                if i in text_ids.keys():
                    canvas.delete(text_ids[i]) #toggling labels
                    continue
                else:
                    pass
                r = canvas.coords(node_ids[i])
                r1 = r[0]+(r[2]-r[0])/2
                r2 = r[1]+(r[3]-r[1])/2
                text =
canvas.create_text(r1+10,r2+10,text=name[i],fill="yellow", anchor=NW)
                text_ids[i]=text

        self.top.destroy()

#end class

class DoubleListbox:
    def __init__(self,parent):
        top = self.top = Toplevel(parent)
        top.title("Activation / Deactivation")


        #action buttons
        frame3 = Frame(top)
        la = Label(frame3,text="action items here.")
        la.pack()
        bset = Button(frame3,text="Set",command=self.set_action)
        bset.pack() #look into inactivating button while no changes
have been made
        frame3.pack(side=BOTTOM, fill=X, expand=0)

        #list box
        frame = Frame(top)
        lname = Label(frame, text="Active Node")
        lname.pack(anchor=NW)

        scrollbar = Scrollbar(frame, orient=VERTICAL)

        self.listbox = Listbox(frame, selectmode=EXTENDED,
exportselection=0, yscrollcommand=scrollbar.set)

        scrollbar.config(command=self.listbox.yview)
        scrollbar.pack(side=RIGHT, fill=Y)
        self.listbox.pack(side=LEFT, fill=BOTH, expand=1)

        frame.pack(side=LEFT, fill=BOTH, expand=1)

        #action buttons
        frame1 = Frame(top)
        boff = Button(frame1, text=">", command=self.off_action)
        bon = Button(frame1, text="<", command=self.on_action)
        boff.pack(anchor=CENTER)
        bon.pack(anchor=CENTER)
        frame1.pack(side=LEFT, expand=1)

        #second list
        frame2 = Frame(top)
        lname = Label(frame2, text="InActive Node")
        lname.pack(anchor=NW)

        scrollbar = Scrollbar(frame2, orient=VERTICAL)

        self.listbox2 = Listbox(frame2, selectmode=EXTENDED,
exportselection=0, yscrollcommand=scrollbar.set)

        scrollbar.config(command=self.listbox2.yview)
        scrollbar.pack(side=RIGHT, fill=Y)
        self.listbox2.pack(side=LEFT, fill=BOTH, expand=1)

        frame2.pack(side=LEFT, fill=BOTH, expand=1)

        #self.listbox.insert(END, "a list entry")
        #for a in range(0,200):
        #    for item in ["one", "two", "three", "four"]:
```

```
        #        self.listbox.insert(END, item)
        #populate lists based on active list
        global active
        for a in active.keys():
            if active[a] == 1:
                self.listbox.insert(END,a)
            else:
                self.listbox2.insert(END,a)
        #sorting both lists
        list1 = self.listbox.get(0,END)
        list2 = self.listbox2.get(0,END)
        list11 = []
        for a in list1:
            list11.append(a)
        list22 = []
        for a in list2:
            list22.append(a)
        list11.sort()
        list22.sort()
        self.listbox.delete(0,END)
        self.listbox2.delete(0,END)
        for a in list11:
            self.listbox.insert(END,a)
        for a in list22:
            self.listbox2.insert(END,a)


        self.result = None


    def off_action(self):
        items = self.listbox.curselection()
        try:
            items = map(int, items) #integer index
        except ValueError: pass
        #print items
        items2=[]
        for a in items:
            items2.append(self.listbox.get(a)) #strings
        #print items2
        for a in items2:
            self.listbox2.insert(END,a)
        i=0
        for a in items:
            self.listbox.delete(a-i)
            i+=1
        #sorting both lists
        list1 = self.listbox.get(0,END)
        list2 = self.listbox2.get(0,END)
        list11 = []
        for a in list1:
            list11.append(a)
        list22 = []
        for a in list2:
            list22.append(a)
        list11.sort()
        list22.sort()
        self.listbox.delete(0,END)
        self.listbox2.delete(0,END)
        for a in list11:
            self.listbox.insert(END,a)
        for a in list22:
            self.listbox2.insert(END,a)


    def on_action(self):
        items = self.listbox2.curselection()
        try:
            items = map(int, items)
        except ValueError: pass
        #print items
        items2=[]
        for a in items:
            items2.append(self.listbox2.get(a))
        #print items2
        for a in items2:
            self.listbox.insert(END,a)
        i=0
        for a in items:
            self.listbox2.delete(a-i)
            i+=1
        #sorting both lists
        list1 = self.listbox.get(0,END)
        list2 = self.listbox2.get(0,END)
        list11 = []
        for a in list1:
            list11.append(a)
        list22 = []
        for a in list2:
            list22.append(a)
        list11.sort()
        list22.sort()
        self.listbox.delete(0,END)
        self.listbox2.delete(0,END)
        for a in list11:
            self.listbox.insert(END,a)
        for a in list22:
            self.listbox2.insert(END,a)

    def set_action(self):
        global active
        list = self.listbox.get(0,END)
        list1 = []
        for a in list:
            list1.append(a)
        list = self.listbox2.get(0,END)
        list2 = []
        for a in list:
            list2.append(a)

        for a in list1:
            active[a] = 1
        for a in list2:
            active[a] = 0
        #end class

class OpenFileDialog:
    def __init__(self,parent):
        top = self.top = Toplevel(parent)
        top.title("Open File...")

        Label(top, text="Input filename").grid(row=0)
        Label(top, text="Nodes").grid(row=1, column=0)
        Label(top, text="Edges").grid(row=2, column=0)
        self.e = Entry(top)
        self.e.grid(row=1,column=1)
        self.f = Entry(top)
        self.f.grid(row=2,column=1)

        b = Button(top, text="Open", command=self.report)
        b.grid(row=2, column=2, padx=3)

        self.result = None

    def report(self):
        global datafile
        datafile = (self.e.get(),self.f.get())
        self.top.destroy()


class GetNames_Edges(OpenFileDialog):
    def checkStatus(self):  # checks if the current file has been
analysed yet
        import string
        global done_comp
        n = ""
        e = ""
        try:
            donef = open('c:/my documents/bin/xstatus.txt','r')
            line = donef.readline()
            line.rstrip()
            [n,e] = string.split(line,';')
            donef.close()
            global datafile
            if n in datafile:
                if e in datafile:
                    #prior analysis complete
                    check = 1
                    done_comp = 1
                    pass
                else: check = 0
            else: check = 0
        except:
            check = 0

        return(check)


    def initActiveLib(self):
        global active, name
        for a in name.keys():
            active[name[a]] = 1

    def importSCC(self):
        global groups
        #read in SCC grouping
        file4 = open('c:/my documents/bin/SCC.txt','r')
        import string
        while string.count(file4.readline(),'SCC') != 1:
            pass
        while 1:
            line = file4.readline()
            if string.count(line,'Spann') != 1:
                pass
            else:
                break
            line = line.rstrip()
            [id,sc] = string.split(line,' ')
            groups[id] = [sc,'#aaaaaa']
        file4.close()

    def importAP(self):
        global groups
        #read in SCC grouping
        file4 = open('c:/my documents/bin/SCC.txt','r')
        import string
        while string.count(file4.readline(),'artic') != 1:
            pass
        while 1:
            line = file4.readline()
            if string.count(line,'bridge') != 1:
                pass
            else:
                break
            line = line.rstrip()
            [id] = string.split(line,'\t')
            groups[id] = [groups[id][0],'red']
        file4.close()

    def importCC(self):
        global groups
        #read in CC grouping
        file4 = open('c:/my documents/bin/CC.txt','r')
        import string
        lines = file4.readlines()
        for line in lines:
            line = line.rstrip()
            [id,cc] = string.split(line,'\t')
            if id in groups.keys():
                groups[id] = [groups[id][0],groups[id][1],cc]
            else:
                groups[id] = [-1,'#545454',cc]
        file4.close()

    def importSigEdg(self):
        global sgroups, bgroups, edges, edge_ids
        #read in Bridge grouping
        file4 = open('c:/my documents/bin/SCC.txt','r')
        import string
        while string.count(file4.readline(),'bridge') != 1:
            pass
        while 1:
            line = file4.readline()
            if string.count(line,'SCC') != 1:
                pass
            else:
                break
            line = line.rstrip()
            [fr,to] = string.split(line,' - ')
            #find edge id that goes with this edge
            i = edges[fr].index(to)
            bgroups.append(edge_ids[fr][i])

        #read in Span grouping
        while string.count(file4.readline(),'Spann') != 1:
            pass
        for line in file4.readlines():
            line = line.rstrip()
            [fr,to] = string.split(line,' ')
            #find edge id that goes with this edge
            i = edges[fr].index(to)
            sgroups.append(edge_ids[fr][i])
        file4.close()


    def importC(self):
        global centrality,u_centrality
        #read in centralities
        file3 = open('c:/my documents/bin/D_centrality.txt');
        import string
        while string.count(file3.readline(),'Vertex') != 1:
            pass
        for line in file3.readlines():
            line = line.rstrip()
            [id,btw,clo] = string.split(line,' ')
            centrality[id] = [btw,clo]
            if string.atof(btw) > centrality['max'][0]:
                centrality['max'][0] = string.atof(btw)
            if string.atof(clo) > centrality['max'][1]:
                centrality['max'][1] = string.atof(clo)
        file3.close()

        file3 = open('c:/my documents/bin/U_centrality.txt');
        while string.count(file3.readline(),'Vertex') != 1:
            pass
        for line in file3.readlines():
            line = line.rstrip()
            [id,btw,clo] = string.split(line,' ')
            u_centrality[id] = [btw,clo]
            if string.atof(btw) > u_centrality['max'][0]:
              u_centrality['max'][0] = string.atof(btw)
            if string.atof(clo) > u_centrality['max'][1]:
                u_centrality['max'][1] = string.atof(clo)
        file3.close()


    def importF(self):
        global datafile, name
        name={}
        #read in nodes and ids
        file1 = open("c:/my documents/bin/"+datafile[0],'r')
        import string
        for line in file1.readlines():
                line = line.rstrip()
                id = line.split(',')[0]
                na = line.split(',')[1][1:-1]
                na = string.replace(na,' ','_')
                na = string.replace(na,'-','+')
                na = string.replace(na,'/','\\')
                name[id] = na
        file1.close()

    def importG(self):
        global datafile, edges
        edges={}
        file = open("c:/my documents/bin/"+datafile[1],'r')
        edge = []
        for line in file.readlines():
                edge.append(line.rstrip())
        file.close()
        nedges = len(edge)
        directed = {} #this dictionary preserves the directed edges
        i=0
        for e in edge:
            d = e.split(',')

            a = d[0]
            b = d[1]
            if a in directed.keys():
                prior = directed[a]
                prior.append(b)
                directed[a] = prior
            else:
                directed[a] = [b]
        edges=directed


    def report(self):
        canvas_clear_all()
        global datafile
        datafile = (self.e.get(),self.f.get())
        self.importF()
        self.importG()
        s = self.checkStatus()
        if s == 1: #analysis has been done for loaded graph
            self.importC()
            self.importSCC()
            self.importAP()
            self.importCC()
        elif s == 0: #analysis has not been done for loaded graph
```

```
            pass
        draw()
        if s == 1:
            self.importSigEdg()
        self.initActiveLib()
        self.top.destroy()

class Load_Data(GetNames_Edges):
    def __init__(self,parent):
        top = self.top = Toplevel(parent)
        top.title("Load Data")

        Label(top, text="Input filename").grid(row=0)
        Label(top, text="Nodes").grid(row=1, column=0)
        Label(top, text="Edges").grid(row=2, column=0)
        global datafile
        self.e = Entry(top)
        self.e.insert(0,datafile[0])
        self.e.grid(row=1,column=1)
        self.f = Entry(top)
        self.f.insert(0,datafile[1])
        self.f.grid(row=2,column=1)

        b = Button(top, text="Open", command=self.report)
        b.grid(row=2, column=2, padx=3)

        self.result = None

    def report(self):
        self.importC()
        self.importSCC()
        self.importAP()
        self.importCC()
        self.importSigEdg()
        self.top.destroy()


class GetNames(OpenFileDialog):
    def importF(self):
        global datafile, name
        #read in nodes and ids
        file1 = open("c:/my documents/bin/"+datafile[0],'r')
        import string
        for line in file1.readlines():
                line = line.rstrip()
                id = line.split(',')[0]
                na = line.split(',')[1][1:-1]
                na = string.replace(na,' ','_')
                na = string.replace(na,'-','+')
                na = string.replace(na,'/','\\')
                name[id] = na
        file1.close()
        #redraw table list

    def report(self):
        global datafile
        datafile = (self.e.get(),self.f.get())
        self.importF()
        self.top.destroy()

class KinDialog:
    def __init__(self,parent):
        global k,h,l
        top = self.top = Toplevel(parent)
        top.title("Kinetics...")

        import fpformat
        k2 = k+0
        h2 = h+0
        l2 = l+0
        Label(top,
text="Repulsion["+fpformat.fix(k2,5)+"]").grid(row=0,column=0)
        Label(top, text="Spring
const["+fpformat.fix(h2,5)+"]").grid(row=1,column=0)
        Label(top, text="Nat
length["+fpformat.fix(l2,5)+"]").grid(row=2,column=0)
        self.e = Entry(top)
        self.e.insert(0,fpformat.fix(k2,5))
        self.e.grid(row=0,column=1)
        self.f = Entry(top)
        self.f.insert(0,fpformat.fix(h2,5))
        self.f.grid(row=1,column=1)
        self.g = Entry(top)
        self.g.insert(0,fpformat.fix(l2,5))
        self.g.grid(row=2,column=1)
        Button(top,
text="Update",command=self.action).grid(row=3,column=1)

    def action(self):
        global h, k , l
        import string
        h=string.atof(self.f.get())
        k=string.atof(self.e.get())+0.00001
        l=string.atof(self.g.get())
        self.top.destroy()


class TableDialog:
    def __init__(self,parent):
        global name, centrality, u_centrality
        top = self.top = Toplevel(parent)
        top.title("Table Dialog...")

        menu1 = Menu(top)
        #menu1.add_command(label="+ Node(s)",command=openf2)
        #menu1.add_command(label="- Node(s)")

        top.config(menu=menu1)

        frame = Frame(top)

        Label(frame, text="Name",
relief=GROOVE,width=20).grid(row=0,column=0,sticky=SW)
```

```
        Label(frame, text="ID",
relief=GROOVE).grid(row=0,column=1,sticky=SW)
        Label(frame, text="D_Btwnes",
relief=GROOVE).grid(row=0,column=2,sticky=SW)
        Label(frame, text="D_Clsnes",
relief=GROOVE).grid(row=0,column=3,sticky=SW)
        Label(frame, text="U_Btwnes",
relief=GROOVE).grid(row=0,column=4,sticky=SW)
        Label(frame, text="U_Clsnes",
relief=GROOVE).grid(row=0,column=5,sticky=SW)

        i = 0
        items = name.keys()
        for item in items:
            i+=1
            Label(frame, text=name[item] ,width=40 , relief=RIDGE,
anchor=W).grid(row=i,column=0)
            Label(frame, text=item, anchor=W).grid(row=i,column=1)
            Label(frame, text=centrality[item][0],
anchor=W).grid(row=i,column=2)
            Label(frame, text=centrality[item][1],
anchor=W).grid(row=i,column=3)
            Label(frame, text=u_centrality[item][0],
anchor=W).grid(row=i,column=4)
            Label(frame, text=u_centrality[item][1],
anchor=W).grid(row=i,column=5)

        frame.pack(anchor=NW)

    def addNode(self):
        global datafile, name
        openf2()

#main program
#globals
datafile = ("None","None")
tag1=0
gotNode = 0
snode = None
selected = [[],[]]
active = {}
node_ids = {}
text_ids = {}
edge_ids = {}
edges = {}
nmap = {}
name = {} #this holds the title of each node
centrality = {'max': [0,0]} #this holds the centralities: d-btw, d-
scloseness
u_centrality = {'max': [0,0]} #this holds the centralities: u-btw, u-
closeness
max_r_size = 20 #this is the maximum size for a node's visual diameter
min_r_size = 2 #this is the minimum size for a node's visual diameter
k = 10   #node repulsion
h = 0.5   #spring constant
l = 40  #natural length
d = 1   # fake viscosity
```

```
rel = 0 #should it be relaxing?
coords_d =[]  #coordinates from Multidimentional Scaling on a directed
graph
coords_u = [] #coordinates from Multidimentional Scaling on an
undirected graph
done_MDS = 0
done_comp = 0
done_clust = 0
lens = 0
lens_off = 1
temp_coords = {}
groups = {} #contains all the node groupings [node_id:scc,ap,cc]
cgroups = {} #contains cluster assignments
[node_id:hCluster,partitions(later)]
bgroups = [] #contains edge ids that are bridges
sgroups = [] #contains edge ids that are spans


def kinetics():
    KinDialog(root)

def freport():
    global datafile, node_ids
    print "Filename = ",datafile
    #print node_ids

def openDoubleListDialog():
    d = DoubleListbox(root)

def openUniListDialog():
    u = UniListbox(root)

def openTable():
    t=TableDialog(root)

def openf():
    #need system halt here
    #old & basic: d = OpenFileDialog(root)
    GetNames_Edges(root)


def openf2():
    #need system halt here
    dd = GetNames(root)

def fitwin(event):
    global tag1
    if tag1 == 0:
        tag1 = event.num
    if event.num < tag1:
        canvas.config(width=root.winfo_pixels(root.winfo_width()),
height=root.winfo_pixels(root.winfo_height()))

def release1(event):
    global gotNode, snode
    if gotNode == 1:
        gotNode = 0
        snode = None
        placeEdges()

def selectNode(event):
    global gotNode, node_ids, snode, text_ids
    xo = canvas.canvasx(event.x)
    yo = canvas.canvasy(event.y)

    if gotNode == 0:
        try:
            i = canvas.find_closest(xo,yo)[0]
            for a in node_ids.keys():
                if i == node_ids[a]:
                    snode = a
                    gotNode = 1
                    break
        except:
            pass
        if gotNode:
            loc = canvas.coords(node_ids[snode])
            r = (loc[2] - loc[0]) / 2
            canvas.coords(node_ids[snode],xo - r,yo - r,xo + r,yo + r)
            try:
                canvas.coords(text_ids[snode],xo + 5,yo + 5)
            except:
                pass
            #placeEdges()
    elif gotNode == 1:
        r = (canvas.coords(node_ids[snode])[2] -
canvas.coords(node_ids[snode])[0])/2
        canvas.coords(node_ids[snode],xo - r,yo - r,xo + r,yo + r)
        try:
            canvas.coords(text_ids[snode],xo + 10,yo + 10)
        except:
            pass
        #placeEdges()

def selectNode2(event):
    global gotNode, node_ids, snode, text_ids
    xo = canvas.canvasx(event.x)
    yo = canvas.canvasy(event.y)

    if gotNode == 0:
        for i in node_ids.keys():
            loc = canvas.coords(node_ids[i])
            x = (loc[2]+loc[0])/2
            y = (loc[3]+loc[1])/2
            if hypot((x-xo),(y-yo)) < 5:
                gotNode = 1
                snode = i
                r = (loc[2] - loc[0]) / 2
                canvas.coords(node_ids[snode],xo - r,yo - r,xo + r,yo +
r)
                try:
                    canvas.coords(text_ids[snode],xo + 5,yo + 5)
                except:
                    pass
                placeEdges()
    elif gotNode == 1:
        r = (canvas.coords(node_ids[snode])[2] -
canvas.coords(node_ids[snode])[0])/2
        canvas.coords(node_ids[snode],xo - r,yo - r,xo + r,yo + r)
        try:
            canvas.coords(text_ids[snode],xo + 10,yo + 10)
        except:
            pass
        placeEdges()


def placeNodes():
    global node_ids, edge_ids, edges, k, h, l, d
    temp = {}
    #calc new positions and store in temp array
    for i in node_ids.keys():
      xo = 0
      yo = 0
      dx = 0
      dy = 0
        loc = canvas.coords(node_ids[i])
        x = (loc[2]+loc[0])/2
        y = (loc[3]+loc[1])/2
        #node repulsion
        for j in node_ids.keys():
            if j != i:
                loc2 = canvas.coords(node_ids[j])
                x2 = (loc2[2]+loc2[0])/2
                y2 = (loc2[3]+loc2[1])/2
                dxx = (x-x2)+0.000001
                dx += k*dxx/pow(abs(dxx),3)
                dyy = (y-y2)+0.000001
                dy += k*dyy/pow(abs(dyy),3)
        #sum of forces
        #xo = x+min(k*dx/pow(abs(dx),3),300)
        #yo = y+min(k*dy/pow(abs(dy),3),300)
        if abs(dx) < 40:
            xo = x + dx
        else:
            xo = x + 40*dx/abs(dx)
        if abs(dy) < 40:
            yo = y + dy
        else:
            yo = y + 40*dy/abs(dy)
        temp[i] = [xo,yo,(loc[2]-loc[0])/2]

    #edge repulsion
    for j in edges.keys():
        for yy in edges[j]:
            dly = 0
            dlx = 0
            loc = canvas.coords(node_ids[j])
            x1 = (loc[2]+loc[0])/2
            y1 = (loc[3]+loc[1])/2
            loc2 = canvas.coords(node_ids[yy])
            x2 = (loc2[2]+loc2[0])/2
            y2 = (loc2[3]+loc2[1])/2
            dl = h*(1-hypot((x1-x2),(y1-y2)))
            dl = dl * d / (d + abs(dl))  #fake viscosity adjustment
            angle = atan2((y2-y1),(x2-x1))
            dlx += dl*cos(angle)
            dly += dl*sin(angle)
            temp[j][0] -= dlx/2
            temp[j][1] -= dly/2
            temp[yy][0] += dlx/2
            temp[yy][1] += dly/2

    #update new positions from temp array
    for i in node_ids.keys():
        canvas.coords(node_ids[i],temp[i][0] - temp[i][2],temp[i][1] -
temp[i][2],temp[i][0] + temp[i][2],temp[i][1] + temp[i][2])
        try:
            canvas.coords(text_ids[i],temp[i][0] + 10,temp[i][1] + 10)
        except:
            pass


def placeEdges():
    global edges, node_ids, edge_ids
    for i in edge_ids.keys():
        kk = 0
        for jj in edge_ids[i]:
            yy = edges[i][kk]
            loc = canvas.coords(node_ids[i])
            x = (loc[2]+loc[0])/2
            y = (loc[3]+loc[1])/2
            loc2 = canvas.coords(node_ids[yy])
            x2 = (loc2[2]+loc2[0])/2
            y2 = (loc2[3]+loc2[1])/2
            canvas.coords(jj,x,y,x2,y2)
            kk +=1

def relax():
    global rel, done_MDS
    done_MDS = 0
    it = 0
    if rel:
        rel = 0
        canvas.config(background="black")
    else:
        rel = 1
        canvas.config(background="green")

    while rel:#it < 2:
      it += 1
        placeNodes()
        #adjust edges to new node locations
        placeEdges()
        canvas.update()

def center():
    global node_ids
    #get window center
    wx = root.winfo_pixels(root.winfo_width()) / 2
    wy = root.winfo_pixels(root.winfo_height()) / 2
    wx=canvas.canvasx(wx)
    wy=canvas.canvasy(wy)
    #get node center
    nx = 0
    ny = 0
    num = 0
    for i in node_ids.keys():
        num += 1
        loc = canvas.coords(node_ids[i])
        nx += (loc[2]+loc[0])/2
        ny += (loc[3]+loc[1])/2
    nx = nx/num
    ny = ny/num
    #pan canvas to s.t. node ctr eq. win ctr (wx-nx)
    for n in canvas.find_all():
        canvas.move(n,wx-nx,wy-ny)

def draw():
    global name, edges, node_ids, edge_ids, text_ids, max_r_size,
min_r_size, centrality
    #refresh for new data
    node_ids = {}
    edge_ids = {}
    text_ids = {}
    import string
    for i in name.keys():
        r = 100*random.random()
        r2 = 100*random.random()
        try:
            r3 =
max_r_size*string.atof(centrality[i] [0])/centrality['max'][0]
            if r3 < min_r_size:
                r3 = min_r_size
            r3 = r3/2
            node = canvas.create_oval(70+r-r3,70+r2-
r3,70+r+r3,70+r2+r3, outline="white")
        except:
            node = canvas.create_oval(70+r-5,70+r2-5,70+r+5,70+r2+5,
outline="white")
        text =
canvas.create_text(70+r+10,70+r2+10,text=name[i],fill="yellow",
anchor=NW)
        node_ids[i]=node
        text_ids[i]=text
    for i in edges.keys():
        for yy in edges[i]:
            loc = canvas.coords(node_ids[i])
            x = (loc[2]+loc[0])/2
            y = (loc[3]+loc[1])/2
            loc2 = canvas.coords(node_ids[yy])
            x2 = (loc2[2]+loc2[0])/2
            y2 = (loc2[3]+loc2[1])/2

            edge =
canvas.create_line(x,y,x2,y2,fill="#444444",arrow=LAST)
```

```
            canvas.lower(edge)
            if i in edge_ids.keys():
                prior = edge_ids[i]
                prior.append(edge)
                edge_ids[i] = prior
            else:
                edge_ids[i] = [edge]

def dviz_normal():
    global node_ids
    import string
    for i in node_ids.keys():
        loc = canvas.coords(node_ids[i])
        x = (loc[2]+loc[0])/2
        y = (loc[3]+loc[1])/2
        r = 5
        canvas.coords(node_ids[i],x - r,y - r,x + r,y + r)

def dviz_ap():
    global node_ids, groups
    colors = {}
    for i in node_ids.keys():
        try:
            canvas.itemconfigure(node_ids[i], outline=groups[i][1])
        except:
            pass

def dviz_edges():
    global bgroups, sgroups
    for i in bgroups:
        try:
            canvas.itemconfigure(i, fill='#ff0099')
        except:
            pass
    for i in sgroups:
        if i in bgroups:
            try:
                canvas.itemconfigure(i,fill='orange')
            except:
                pass
        else:
            try:
                canvas.itemconfigure(i, fill='yellow')
            except:
                pass

def dviz_cc():
    global node_ids, groups
    import string
    colors = {}
    for i in node_ids.keys():
        #as a new group id is read, assign a new random color and store
in the colors dictionary
        try:
            if groups[i][2] in colors.keys():
                pass
            else:
                tk_rgb = "#%02x%02x%02x" % (255*random.random(),
255*random.random(), 255*random.random())
                colors[groups[i][2]]= tk_rgb
            canvas.itemconfigure(node_ids[i],
outline=colors[groups[i][2]])
        except:
            pass

def dviz_scc():
    global node_ids, groups
    import string
    colors = {}
    for i in node_ids.keys():
        #as a new group id is read, assign a new random color and store
in the colors dictionary
        try:
            if groups[i][0] in colors.keys():
                pass
            else:
                tk_rgb = "#%02x%02x%02x" % (255*random.random(),
255*random.random(), 255*random.random())
                colors[groups[i][0]]= tk_rgb
            canvas.itemconfigure(node_ids[i],
outline=colors[groups[i][0]])
        except:
            pass

def dviz_d_bet():
    global centrality, max_r_size, min_r_size, node_ids
    import string
    for i in node_ids.keys():
        loc = canvas.coords(node_ids[i])
        x = (loc[2]+loc[0])/2
        y = (loc[3]+loc[1])/2
        r3 =
max_r_size*string.atof(centrality[i][0])/centrality['max'][0]
        if r3 < min_r_size:
            r3 = min_r_size
        r3 = r3/2
        canvas.coords(node_ids[i],x - r3,y - r3,x + r3,y + r3)
def dviz_d_clo():
    global centrality, max_r_size, min_r_size, node_ids
    import string
    for i in node_ids.keys():
        loc = canvas.coords(node_ids[i])
        x = (loc[2]+loc[0])/2
        y = (loc[3]+loc[1])/2
        r3 = max_r_size*(1-
(string.atof(centrality[i][1])/centrality['max'][1]))
        if r3 < min_r_size:
            r3 = min_r_size
        r3 = r3/2
        canvas.coords(node_ids[i],x - r3,y - r3,x + r3,y + r3)
def dviz_u_bet():
    global u_centrality, max_r_size, min_r_size, node_ids
    import string
    for i in node_ids.keys():
        loc = canvas.coords(node_ids[i])
        x = (loc[2]+loc[0])/2
        y = (loc[3]+loc[1])/2
        r3 =
max_r_size*string.atof(u_centrality[i][0])/u_centrality['max'][0]
        if r3 < min_r_size:
            r3 = min_r_size
        r3 = r3/2
        canvas.coords(node_ids[i],x - r3,y - r3,x + r3,y + r3)
def dviz_u_clo():
    global u_centrality, max_r_size, min_r_size, node_ids
    import string
    for i in node_ids.keys():
        loc = canvas.coords(node_ids[i])
        x = (loc[2]+loc[0])/2
        y = (loc[3]+loc[1])/2
        r3 = max_r_size*(1-
(string.atof(u_centrality[i][1])/u_centrality['max'][1]))
        if r3 < min_r_size:
            r3 = min_r_size
        r3 = r3/2
        canvas.coords(node_ids[i],x - r3,y - r3,x + r3,y + r3)

def Scale(width, coords):
    global node_ids
    maxx = 0
    minx = 50000000
    maxy = 0
    miny = 50000000
    new_coords = []
    for a in coords:
        if a[0] > maxx:
            maxx = a[0]
        if a[0] < minx:
            minx = a[0]
        if a[1] > maxy:
            maxy = a[1]
        if a[1] < miny:
            miny = a[1]
    x = maxx-minx
    y = maxy-miny

    for a in coords:
        new_coords.append([a[0]*width/x,a[1]*width/y])
    global nmap
    for i in node_ids.keys():
        b = nmap[i] -1
        if b > len(new_coords)-1:
            print b
            pass
        else:
            xo = new_coords[b][0]
            if xo < 0:
                xo = abs(xo)*(-1.0)
            else:
                xo = abs(xo)
            yo = new_coords[b][1]
            if yo < 0:
                yo = abs(yo)*(-1.0)
            else:
                yo = abs(yo)
            r = (canvas.coords(node_ids[i])[2] -
canvas.coords(node_ids[i])[0])/2
            canvas.coords(node_ids[i],xo - r,yo - r,xo + r,yo + r)
            try:
                canvas.coords(text_ids[i],xo + 10,yo + 10)
            except:
                pass

    placeEdges()
    center()



def MDS():
    #Multi dimentional scaling tool
    from Matrix import *
    from LinearAlgebra import *
    #read in D & U distance matrix
    import pickle
    print "reading in filez"
    file1 = open("c:/my documents/bin/D_dist.txt",'r')
    ddist = pickle.load(file1)
    file2 = open("c:/my documents/bin/U_dist.txt",'r')
    udist = pickle.load(file2)
    file1.close()
    file2.close()
    print "done read in"
    #directed graph
    DM = []
    a = ddist[0].keys()
    a.sort()
    s = a[-1]
    for b in range(0,s):
        DM.append([])
    diff = 0
    for b in range(0,s):
        if b+1 in a:
            for c in range(0,s):
                if c+1 in a:
                    DM[b].append(ddist[b-diff][c+1])
                else:
                    DM[b].append(2000)
        else:
            diff+=1
            for c in range(0,s):
                if b == c:
                    DM[b].append(0)
                else:
                    DM[b].append(2000)
    DM =Matrix(DM)
    print "made DM"
    #undirected graph
    UM = []
    a = udist[0].keys()
    a.sort()
    s = a[-1]
    for b in range(0,s):
        UM.append([])
    diff = 0
    for b in range(0,s):
        if b+1 in a:
            for c in range(0,s):
                if c+1 in a:
                    UM[b].append(udist[b-diff][c+1])
                else:
                    UM[b].append(2000)
        else:
            diff+=1
            for c in range(0,s):
                if b == c:
                    UM[b].append(0)
                else:
                    UM[b].append(2000)
    UM =Matrix(UM)
    print "made UM"

    #get D & U MDS coordinates
     #get eigen system
    es_d = eigenvectors(DM)
    es_u = eigenvectors(UM)
     #Choose 2 largest eigenvalues
    #directed graph
    first = 3
    second = 4
    third = 5
    temp = []
    for a in range(0,len(es_d[0])):
        if abs(es_d[0][a]) > abs(es_d[0][third]):
            if abs(es_d[0][a]) > abs(es_d[0][second]):
                if abs(es_d[0][a]) > abs(es_d[0][first]):
                    first = a
                else:
                    second = a
            else:
                third = a
    pc1_d = es_d[1][first]
    pc2_d = es_d[1][second]
    pc3_d = es_d[1][third]

     #Get coords from projection of each node's distance vector onto
each principal axis
    coords_d = []
    for a in range(0,len(DM)):
        if a+1 in ddist[0].keys():
            try:
                x = (pc1_d*DM[a]).real
                y = (pc2_d*DM[a]).real
            except:
                x = pc1_d*DM[a]
                y = pc2_d*DM[a]
            coords_d.append([x,y])
    print "got d coords"
    #undirected graph
    first = 3
    second = 4
    third = 5
    temp = []
    for a in range(0,len(es_u[0])):
        if abs(es_u[0][a]) > abs(es_u[0][third]): # this line
previously had es_d instead of es_u (this may have caused as out of
range error)
            if abs(es_u[0][a]) > abs(es_u[0][second]):
                if abs(es_u[0][a]) > abs(es_u[0][first]):
                    first = a
                else:
                    second = a
            else:
                third = a
    pc1_u = es_u[1][first]
    pc2_u = es_u[1][second]
    pc3_u = es_u[1][third]

     #Get coords from projection of each node's distance vector onto
each principal axis
    coords_u = []
    for a in range(0,len(UM)):
        if a+1 in udist[0].keys():
            try:
                x = (pc1_u*UM[a]).real
                y = (pc2_u*UM[a]).real
            except:
                x = pc1_u*UM[a]
                y = pc2_u*UM[a]
            coords_u.append([x,y])
    print "got u coords"
    #quick read in of node id mapping for use in scaling function
    #dictionary[old]=>new
    import pickle
    file3 = open("c:/my documents/bin/node_map.txt",'r')
    global nmap
    nmap = pickle.load(file3)
    file3.close()
    #done mapping read in.
    return coords_d,coords_u
    #end MDS

def layout_u():
    global coords_d, coords_u,done_MDS,done_comp
    #if done_MDS != 1:
    if done_comp == 0:
        return
    if len(coords_d) > 0:
        #do layout
        Scale(600,coords_u)
        done_MDS = 1
    else:
        coords_d, coords_u = MDS()
        #do layout
        Scale(600,coords_u)
        done_MDS = 1

def layout_d():
    global coords_d, coords_u,done_MDS,done_comp
    #if done_MDS != 1:
    if done_comp == 0:
        return
    if len(coords_d) > 0:
        #do layout
        Scale(600,coords_d)
        done_MDS = 1
    else:
        coords_d, coords_u = MDS()
        #do layout
        Scale(600,coords_d)
        done_MDS = 1

def canvas_clear_all():
    global text_ids, node_ids, edge_ids
    for a in text_ids.keys():
        canvas.delete(text_ids[a])
        del text_ids[a]
    for a in node_ids.keys():
        canvas.delete(node_ids[a])
        del node_ids[a]
    for a in edge_ids.keys():
        canvas.delete(edge_ids[a])
        del edge_ids[a]
    for j in canvas.find_all():
        canvas.delete(j)


def label_clear_all():
    global text_ids
    for a in text_ids.keys():
        canvas.delete(text_ids[a])
        del text_ids[a]

def label_show_all():
    global node_ids, text_ids, name
    for i in name.keys():
        r = canvas.coords(node_ids[i])
        r1 = r[0]+(r[2]-r[0])/2
        r2 = r[1]+(r[3]-r[1])/2
        text =
canvas.create_text(r1+10,r2+10,text=name[i],fill="yellow", anchor=NW)
        text_ids[i]=text

def label_show():
    global node_ids, text_ids, name, selected
    for i in name.keys():
        if node_ids[i] in selected[1]:
            r = canvas.coords(node_ids[i])
            r1 = r[0]+(r[2]-r[0])/2
            r2 = r[1]+(r[3]-r[1])/2
            try: # to avoid duplicate labels
                canvas.delete(text_ids[i])
            except:
                pass
            text =
canvas.create_text(r1+10,r2+10,text=name[i],fill="yellow", anchor=NW)
            text_ids[i]=text

def label_hide():
    global node_ids, text_ids, name, selected
    for i in name.keys():
        if node_ids[i] in selected[1]:
            r = canvas.coords(node_ids[i])
            r1 = r[0]+(r[2]-r[0])/2
            r2 = r[1]+(r[3]-r[1])/2
            try: # to avoid duplicate labels
                canvas.delete(text_ids[i])
                del text_ids[i]
            except:
                pass


def selectNode_box(event):
    global gotNode, selected
    xo = canvas.canvasx(event.x)
    yo = canvas.canvasy(event.y)
    if gotNode == 0:
        selected[0].append(xo)
        selected[0].append(yo)
        gotNode = 1
        item =
canvas.create_rectangle(xo,yo,xo+1,yo+1,outline="blue",fill="blue",stip
ple="gray12")
        selected[0].append(item)
    elif gotNode == 1:

canvas.coords(selected[0][2],selected[0][0],selected[0][1],xo,yo)

def release2b(event):
    global gotNode, selected
    xo = canvas.canvasx(event.x)
    yo = canvas.canvasy(event.y)

    #get new point and add all nodes inside box to selected list.
    if gotNode == 1:
        gotNode = 0
        canvas.bind("<B1-Motion>",selectNode)
        canvas.bind("<ButtonRelease-1>",release1)
        canvas.delete(selected[0][2])
        selected[1] =
canvas.find_overlapping(selected[0][0],selected[0][1],xo,yo)
        #later parsed for nodeids to act on (ie. add label)
        selected[0] = []

def select_box():
    canvas.bind("<B1-Motion>",selectNode_box)
    canvas.bind("<ButtonRelease-1>",release2b)
```

```
def draw_lens(event):
    global lens, lens_off, selected, name, node_ids, text_ids,
temp_coords
    xo = canvas.canvasx(event.x)
    yo = canvas.canvasy(event.y)
    R = 100.0   #lens(sphere) radius
    H = 25.0    #depth of sphere center beneath plane (originally 50.0)
    if lens_off:
        lens = canvas.create_oval(xo-R,yo+R,xo+R,yo-R,outline="blue")
        lens_off = 0
    else:
        canvas.coords(lens,xo-R,yo+R,xo+R,yo-R)
        #select nodes inside lens
        selected[1] = canvas.find_enclosed(xo-R,yo+R,xo+R,yo-R)
        #distort them
        for i in name.keys():
            if node_ids[i] in selected[1]:
                if i in temp_coords.keys():
                    r = temp_coords[i]
                else:
                    r = canvas.coords(node_ids[i])
                    temp_coords[i] = r  #save old coordinates

                ox = r[0]+(r[2]-r[0])/2
                oy = r[3]+(r[1]-r[3])/2
                ax = 0.0001-(xo-ox)
                ay = 0.0001+oy-yo
                if hypot(ax,ay) > R:  #make sure node is within lens
                    continue
                r = temp_coords[i]
                c = min(R,sqrt(pow(ax,2)+pow(ay,2)+pow(H,2)))
                d = R-c
                e = d*hypot(ax,ay)/(c+0.0001)
                p = e+hypot(ax,ay)
                theta = atan(abs(ay/ax))
                px = p*cos(theta)*ax/abs(ax) + xo
                py = p*sin(theta)*ay/abs(ay) + yo

                #px and py are new coordinates for node
                rr = (canvas.coords(node_ids[i])[2] -
canvas.coords(node_ids[i])[0])/2
                canvas.coords(node_ids[i],px - rr,py - rr,px + rr,py +
rr)
                try:
                    canvas.coords(text_ids[i],px + 10,py + 10)
                except:
                    pass



def kill_lens():
    global lens, lens_off, temp_coords
    canvas.unbind("<B3-Motion>")
    canvas.delete(lens)
    lens_off = 1
    temp_coords = {}
```

```
def fisheye_lens():
    global lens_off
    if lens_off:
        canvas.bind("<B3-Motion>",draw_lens)
    else:
        kill_lens()

def runall():
    global datafile,done_comp
    import os
    cmd = "python c:\\mydocu~1\\bin\\step_one.py %s %s" %
(datafile[0],datafile[1])
    os.system(cmd)
    done_comp = 1
    #load results
    Load_Data(root)

def ccluster():
    #later put in console for different clustering options.
    global cgroups, done_comp, done_clust
    if done_comp==0:
        print "Analysis must be done prior to clustering.  \nPlease
select 'Run all.'\n"
        return
    if done_clust==0:
        done_clust=1
        #Ccluster.py

        #read in undir distance matrix (U_dist.txt),
        import pickle
        file2 = open("c:/my documents/bin/U_dist.txt",'r')
        udist = pickle.load(file2)
        file2.close()
        #... directed (dedges.txt) and undirected (edges.txt) graphs
         #dictionary[from]=>to
        file4 = open("c:/my documents/bin/dedges.txt",'r')
        child = pickle.load(file4)
        file4.close()
         #dictionary[from]=>to
        file4 = open("c:/my documents/bin/edges.txt",'r')
        neighbor = pickle.load(file4)
        file4.close()
         #dictionary[old]=>new
        file3 = open("c:/my documents/bin/node_map.txt",'r')
        nmap = pickle.load(file3)
        nmap2={} #new > old
        for y in nmap.keys():
            nmap2[nmap[y]] = y
        file3.close()

        global Fact
        def Fact(a):
            import math
            if a == 0:
                return(1)
            if a < 10: #error for Stirling at 10 is 0.8%
```

```
                c=1
                while a > 0:
                    c = c*a
                    a = a-1
                return(c)
            answer =
round(math.pow((2*2.718281825/(a+1)),0.5)*1/math.exp(a+1)*pow(a+1.0,a+1
.0))
            return(answer)

        global Comb
        def Comb(a,b):
            c = Fact(a)/Fact(b)/Fact(a-b)
            c_str = "%f" % c
            if c_str == '-1.#IND00':
                c = 0.9999999999
            return(c)

        global sComb
        def sComb(a,b):
            mx = max(b,(a-b))
            mn = min(b,(a-b))
            num = 1.0000000 #recursive mult
            for i in range(mx+1,a+1):
                num *= i
            c = num/Fact(mn)
            #print a,b,c
            c_str = "%f" % c
            if c_str == '-1.#IND00':
                c = 0.9999999999
            return(c)

        L = 0  #total number of edges in dir graph
        for i in child.keys():
            L = L + len(child[i])
        g = len(neighbor.keys()) #number of nodes in undir graph

        def Cohesiveness(A,neighbor, child, L, g):
            #this doesn't work because of the overflow errors
            #pass in dictionary for directed and undirected graph with
A
            #global neighbor, child, L, g
            #A = list of nodes like, [a,b,c,d]
            q = 0 #number of edges within dir cluster A
            for i in A:
                if i in child.keys():
                    for j in child[i]:
                        if j in A:
                            q = q + 1
            gs = len(A) #number of nodes within undir cluster A
            pf = 0
            for k in range(q,min(L,1+gs*(gs-1)/2)):
                pf = pf + Comb(L,k)*Comb(g*(g-1)/2-L,gs*(gs-1)/2-
k)/Comb(g*(g-1)/2,gs*(gs-1)/2)
                #print pf, Comb(L,k)
            return(pf)
        def Cohesiveness2(A,L,g):
            #ratio of cluster density to graph density
            #density = 2L/g(g-1)
            #global L, g

            density1 = 2.0*L/g/(g-1)

            q = 0 #number of edges within dir cluster A
            for i in A:
                if i in child.keys():
                    for j in child[i]:
                        if j in A:
                            q = q + 1
            gs = len(A) #number of nodes within undir cluster A

            density2 = 2.0*q/gs/(gs-1)

            return(density2/density1)

        def Cohesiveness3(A,neighbor, child, L, g):
            #b4 this didn't work because of the overflow errors
            #but now I am trying to simplify the factorial math
involved in calculating lCk
            #pass in dictionary for directed and undirected graph with
A
            #global neighbor, child, L, g
            #A = list of nodes like, [a,b,c,d]
            q = 0 #number of edges within dir cluster A
            for i in A:
                if i in child.keys():
                    for j in child[i]:
                        if j in A:
                            q = q + 1
            gs = len(A) #number of nodes within undir cluster A
            pf = 0
            for k in range(q,min(L,1+gs*(gs-1)/2)):
                pf = pf + sComb(L,k)*sComb(g*(g-1)/2-L,gs*(gs-1)/2-
k)/sComb(g*(g-1)/2,gs*(gs-1)/2)
                #print pf, sComb(L,k)
            return(pf)

        def MinDist(A,a,udist): #0:min,1:max,2:avg
#            global udist
            avgc = [-1,-1,500000000]
            maxc = [-1,-1,500000000]
            min = [-1,-1,500000000]
            for i in A:
                for j in A:
                    avg = 0
                    max = [-1,-1,0]
                    if j != i:
                        for k in i:
                            for l in j:
                                d = udist[k-1][l]
                                if d > max[2]:
                                    max = [A.index(i),A.index(j),d]
                                if d < min[2]:
                                    min = [A.index(i),A.index(j),d]
                                avg += d
                        if max[2]<maxc[2]:
                            maxc = max
                        avg = avg/len(i)/len(j)
                        if avg < avgc[2]:
                            avgc = [A.index(i),A.index(j),avg]
            if a == 0:
                return(min[0],min[1])
            if a == 1:
                return(maxc[0],maxc[1])
            if a == 2:
                return(avgc[0],avgc[1])




        #main
        R = []
        ta = []
        for i in neighbor.keys():
            ta.append([i])
        R.append(ta)
        ta = []

        S = []
        t = 0

        while len(R[t]) > 1:
            t += 1
            Cq = []
            [i,j] = MinDist(R[t-1],0,udist)
            Cq = R[t-1][i] + R[t-1][j]
            T = [] + R[t-1]
            T.remove(R[t-1][i])
            T.remove(R[t-1][j])
            C = Cohesiveness3(Cq,neighbor, child, L, g)
            if (C <= 0.025) & (len(Cq) > 1): #with cohesiveness2 use
thresshold of 7; w/3 use?
                S.append(Cq)
                R.append(T)
            else:
                T.append(Cq)
                R.append(T)


        #print R[-1],S
        file = open("c:/my documents/bin/cClust.txt",'w')
        file.write("ID,cluster(-1:r; n:s)\n")
        for o in R[-1]:
            for p in o:
                file.write(nmap2[p]+",-1\n")
                id = nmap2[p]
                if id in cgroups.keys():
                    cgroups[id] = [cgroups[id][0]]
                else:
                    cgroups[id] = [-1]

        q = 0
        for o in S:
            for p in o:
                outstring = ",%d\n" % q
                file.write(nmap2[p]+outstring)
                id = nmap2[p]
                if id in cgroups.keys():
                    cgroups[id] = [cgroups[id][0]]
                else:
                    cgroups[id] = [q]
            q+=1
        file.close()
        print "... done, clustering.\n"

    #display the clusters by node color.
    global node_ids
    import string
    colors = {}
    for i in node_ids.keys():
        #as a new group id is read, assign a new random color and store
in the colors dictionary
        try:
            if cgroups[i][0] in colors.keys():
                pass
            else:
                tk_rgb = "#%02x%02x%02x" % (255*random.random(),
255*random.random(), 255*random.random())
                colors[cgroups[i][0]]= tk_rgb
            canvas.itemconfigure(node_ids[i],
outline=colors[cgroups[i][0]])
        except:
            pass

def export_graph():
    global edge_ids, name, edges, active, datafile
    import string

    if 1 in active.values():
        #check for a flag (_'#)in the name, if its there look for its
id and get its increment
        #create a new file with the flag and new id.
        #Important!!! Add funtion to check for prexisting output file
with the same name

        if ':' in datafile[0]:
            [rootname1,tmp] = string.split(datafile[0],'_')
            [id1,suf1] = string.split(tmp,'.')
        else:
            [rootname1,suf1] = string.split(datafile[0],'.')
            id1 = '10'
        id2 = "%d" % (string.atoi(id1)+1)
        newname1 = rootname1+"_"+id2+"."+suf1
        #print newname1
        nameout = open("c:/my documents/bin/"+newname1,'w')
        if ':' in datafile[1]:
            [rootname1,tmp] = string.split(datafile[1],'_')
            [id1,suf1] = string.split(tmp,'.')
        else:
            [rootname1,suf1] = string.split(datafile[1],'.')
            id1 = '10'
        id2 = "%d" % (string.atoi(id1)+1)
        newname2 = rootname1+"_"+id2+"."+suf1
        #print newname2
        edgeout = open("c:/my documents/bin/"+newname2,'w')
        pass
    else:
        return
    for a in name.keys():
        if active[name[a]] == 1:
            #print "%s,\"%s\"\n" % (a,name[a])
            nameout.write(a+",\""+name[a]+"\"\n")
    for a in edges.keys():
        if active[name[a]] == 1:
            for b in edges[a]:
                if active[name[b]] == 1:
                    i = edges[a].index(b)
                    #print "%s,%s,%d\n" %(a,b,edge_ids[a][i])
                    outstring = "%s,%s,%d\n" %(a,b,edge_ids[a][i])
                    edgeout.write(outstring)
    nameout.close()
    edgeout.close()


root = Tk()

menubar = Menu(root)
filemenu = Menu(menubar, tearoff=0)
filemenu.add_command(label="Open", command=openf)
filemenu.add_command(label="Save (saves active graph)",
command=export_graph)
filemenu.add_command(label="Info", command=freport)
filemenu.add_command(label="Table", command=openTable)
filemenu.add_separator()
filemenu.add_command(label="Exit", command=root.quit)

analysismenu = Menu(menubar,tearoff=0)
analysismenu.add_command(label="Run all", command=runall)
analysismenu.add_command(label="Cluster", command=ccluster)
analysismenu.add_command(label="centrality")
analysismenu.add_command(label="topology")

scenemenu = Menu(menubar, tearoff=0)
dvizmenu = Menu(scenemenu,tearoff=0)
labelmenu = Menu(scenemenu,tearoff=0)
scenemenu.add_cascade(label="DataViz",menu=dvizmenu)
scenemenu.add_separator()
scenemenu.add_command(label="Relax", command=relax)
scenemenu.add_command(label="Center", command=center)
scenemenu.add_command(label="Kinetics", command=kinetics)
scenemenu.add_separator()
scenemenu.add_command(label="MDS layout U", command=layout_u)
scenemenu.add_command(label="MDS layout D", command=layout_d)
scenemenu.add_separator()
scenemenu.add_command(label="Fisheye", command=fisheye_lens)


selectmenu = Menu(menubar, tearoff=0)
selectmenu.add_cascade(label="Label",menu=labelmenu)
selectmenu.add_command(label="Box", command=select_box)
selectmenu.add_command(label="Activate?", command=openDoubleListDialog)
selectmenu.add_command(label="List", command=openUniListDialog)

labelmenu.add_command(label="Hide all", command=label_clear_all)
labelmenu.add_command(label="Show all", command=label_show_all)
labelmenu.add_command(label="Show selected", command=label_show)
labelmenu.add_command(label="Hide selected", command=label_hide)


dvizmenu.add_command(label="None", command=dviz_normal)
dvizmenu.add_command(label="D Betweenness", command=dviz_d_bet)
dvizmenu.add_command(label="D Closeness", command=dviz_d_clo)
dvizmenu.add_command(label="U Betweenness", command=dviz_u_bet)
dvizmenu.add_command(label="U Closeness", command=dviz_u_clo)
dvizmenu.add_separator()
dvizmenu.add_command(label="Strongly Connected Components",
command=dviz_scc)
dvizmenu.add_command(label="Show Components", command=dviz_cc)
dvizmenu.add_command(label="Show Articulation Points", command=dviz_ap)
dvizmenu.add_command(label="Show Bridges | Spans", command=dviz_edges)

menubar.add_cascade(label="File", menu=filemenu)
menubar.add_cascade(label="Analysis",menu=analysismenu)
menubar.add_cascade(label="Scene",menu=scenemenu)
menubar.add_cascade(label="Select",menu=selectmenu)
# display the menu
root.config(menu=menubar)

yscrollbar = Scrollbar(root)
yscrollbar.pack(side=RIGHT, fill=Y)
xscrollbar = Scrollbar(root, orient=HORIZONTAL)
xscrollbar.pack(side=BOTTOM,fill=X)

canvas = Canvas(root, background="black",width=500, height=500,
yscrollcommand=yscrollbar.set, xscrollcommand=xscrollbar.set)
canvas.bind("<Configure>",fitwin)
canvas.bind("<B1-Motion>",selectNode)
canvas.bind("<ButtonRelease-1>",release1)
canvas.pack(side=LEFT,anchor=NW)

yscrollbar.config(command=canvas.yview)
xscrollbar.config(command=canvas.xview)

import random
draw()

root.geometry("200x300+0+0")
root.mainloop()
```

:::ks_part3.py:::

```
#Author: Abraham Anderson
#© 2002 The Regents of the University of California

#begin init
import string
import pickle

#dictionary[old]=>new
file3 = open("c:/my documents/bin/node_map.txt",'r')
nmap = pickle.load(file3)
nodes_r = nmap

nmap2={}
for y in nmap.keys():
    nmap2[nmap[y]] = y

nodes = nmap2
file3.close()


file1 = open("c:/my documents/bin/xedges.txt",'r')
lines = file1.readlines()
for a in lines:
    b = a.strip()
    c = lines.index(a)
    lines.remove(a)
    lines.insert(c,b)


from Matrix import *
from LinearAlgebra import *

l = max(nodes_r.values())
A = []
for a in range(0,l):
    t = []
    for b in range(0,l):
            t.append(0)
    A.append(t)

for a in lines:
    [i,j,k] = string.split(a,',')
    i = nodes_r[i]
    j = nodes_r[j]
    i -= 1
    j -= 1
    A[i][j] = 1
    A[j][i] = 1


Deg = {}
for a in A:
      Deg[A.index(a)] = a.count(1)

D = []
for a in range(0,l):
    t = []
    for b in range(0,l):
            t.append(0)
    D.append(t)


for a in Deg.keys():
    D[a][a] = Deg[a]

e = len(lines)
LL=[]
for a in lines:
    [i,j,k] = string.split(a,',')
    i = nodes_r[i]
    j = nodes_r[j]
    i-=1
    j-=1
    t = []
    for b in range(0,l):
            t.append(0)
    t[i]= -1
    t[j]= 1
    LL.append(t)

LL = Matrix(LL)
from Numeric import transpose
LLi = transpose(LL)


A = Matrix(A) #unsigned adjacency matrix
D = Matrix(D) #degree matrix
L = A-D       #Laplacian [1]. use -1 as max_ev
B = D-A       #Disconnection matrix -> Standard spectrum [2]. use 1 as
max_ev
LLj = LLi*LL  #Laplacian from [2]. This equals B but is presented as
'different?'
M = B   #spectrum of choice
maximal_ev = 1 #index for sorted ev list
#end init

#selecting the cluster that gives a minimal cut
def BestV(T,h):
    l = []
    for a in T:
        l.append(len(a[2]))
        #v1 l.append(a[0][h])
    #v1 return(T[l.index(min(l))])
    return(T[l.index(max(l))]) #the biggest cluster

#add new cluster(s) w/ calculated eigen space to another cluster list
def UpdateT(t):
    global M,T
    for a in t:
        l = len(a[2])
        if l <= 3:
            T.append(a)
            continue
        mm = [] #temporary matrix
        for c in range(0,l):
            m = []
            for b in range(0,l):
                m.append(0)
            mm.append(m)
        for c in a[2]:
            for b in a[2]:
                mm[a[2].index(c)][a[2].index(b)] = M[c-1][b-1]
        mm=Matrix(mm)

        from LinearAlgebra import *
        es = eigenvectors(mm)
        evs = []
        for aa in es[0]:
            evs.append(aa)
        evs.sort()
        v = []
        for aa in es[0]:
            v.append(aa)
        #set of 3 'maximal' eigenvalues
        meval =
[es[0][v.index(evs[maximal_ev])],es[0][v.index(evs[maximal_ev+1])],es[0
][v.index(evs[maximal_ev+2])]]
        #set of 3 'maximal' eigenvectors
        mevec = []
        mevec.append(es[1][v.index(evs[maximal_ev])])
        mevec.append(es[1][v.index(evs[maximal_ev+1])])
        mevec.append(es[1][v.index(evs[maximal_ev+2])])

        v = meval #list of 3 smallest e_values
        e = mevec #list of 3 associated e_vectors
        T.append([v,e,a[2]])


#core partitioner
def Bisect(t,k):
    lt = []
    rt = []
    ev = [[],[],[]]
    ev2 = [[],[],[]]
    for a in t[2]:
        if t[1][k][t[2].index(a)] < 0:
            lt.append(a)
            ev[0].append(t[1][0][t[2].index(a)])
            ev[1].append(t[1][1][t[2].index(a)])
            ev[2].append(t[1][2][t[2].index(a)])
        else:
            ev2[0].append(t[1][0][t[2].index(a)])
            ev2[1].append(t[1][1][t[2].index(a)])
            ev2[2].append(t[1][2][t[2].index(a)])
            rt.append(a)
    lt = [t[0],ev,lt]
    rt = [t[0],ev2,rt]
    return([lt,rt])
```

```
#Partitioning scheduler
def Divide(k,i):
    global T
    for a in range(0,i):
        t = BestV(T,0)
        if k==2:
            s = Bisect(t,0)
            T.remove(t)
            UpdateT(s)
        if k==3:
            s = Bisect(t,0)
            T.remove(t)
            t = BestV(s,1)
            s.remove(t)
            UpdateT(s)
            s = Bisect(t,1)
            UpdateT(s)
        if k==4:
            s = Bisect(t,0)
            T.remove(t)
            t = BestV(s,1)
            s.remove(t)
            ss = Bisect(t,1)
            for h in ss:
                s.append(h)
            t = BestV(s,1)
            s.remove(t)
            ss = Bisect(t,2)
            for h in ss:
                s.append(h)
            UpdateT(s)
        if k==8:
            s = Bisect(t,0)
            T.remove(t)
            for b in s:
                ss = Bisect(b,1)
                for c in ss:
                    sss = Bisect(c,2)
                    UpdateT(sss)
    return(T)

#Main function
def KPart(k,T):
    #k = number of partitons
    #N = list of all nodes
    #T = list of partitions(as sublists)
    i = k/8
    if i == 0:
        pass
    else:
        T = Divide(8,i)
    i = (k-len(T)+1)/4
    if i == 0:
        pass
    else:
        T = Divide(4,i)
    i = (k-len(T)+1)/3
    if i == 0:
        pass
    else:
        T = Divide(3,i)
    i = (k-len(T)+1)/2
    if i == 0:
        return(T)
    else:
        T = Divide(2,i)
        return(T)

#main
print "Spectral partitioning of X into k peices.\n"
print "Smallest number of partitions:"
while 1:
    c = raw_input("\t")
    break
print "Largest number of partitions:"
while 1:
    p = raw_input("\t")
    break

pgroups = {} #list of partition assignments like [1,2,1,5,...,k]

for a in range(string.atoi(c),string.atoi(p)+1):
    #repeat this...
    print "Begin p%d ..." % a
    T=[]
    UpdateT([[[],[],nodes.keys()]])
    T = KPart(a,T)

    p =0
    for g in T:
        #print g[2]
        for u in g[2]:
            if u in pgroups.keys():
                set = pgroups[u]
                set.append(p)
                pgroups[u] = set
            else:
                pgroups[u] = [p]
        p += 1
    print "Finish p%d." % a
    #stop repeated section.
print "Assembling x_partitions.txt ..."
mydir = "c:\\mydocu~1\\bin\\"
outfile = open(string.join([mydir,'x_partitions.txt'],''),'w')
idz = pgroups.keys()
idz1 = []
for a in idz:
    idz1.append(a)
idz1.sort()
for a in idz1:
    idz2 = a
    outline = ""
    for z in pgroups[idz2]:
        zz = "%d" % z
        outline = string.join([outline,zz],'\t')
    outline = string.join([outline,"\n"],'')
    outline = outline[1:]
    outfile.write(outline)
outfile.close()
print "Done with partitioner."
print "Looking for nodes on the edge of the partitions, and writing to
partition_eval.txt.\n"
import peval
print "Results are ready for analysis."
```

:::peval.py:::

```
#Author: Abraham Anderson
#© 2002 The Regents of the University of California

import pickle
import string

#dictionary[new]=>old
file3 = open("c:/my documents/bin/node_map.txt",'r')
nmap = pickle.load(file3)

nmap2={}
for y in nmap.keys():
    nmap2[nmap[y]] = y

file3.close()
#dictionary[from]=>to
file4 = open("c:/my documents/bin/edges.txt",'r')
direct = pickle.load(file4)

file4.close()

file = open("c:/my documents/bin/x_partitions.txt",'r')
a=string.split(string.strip(file.readline()),"\t")
b = len(a)
i=1
d = []
d.append(a)
for a in file.readlines():
    i+=1
    aa=string.split(string.strip(a),"\t")
    d.append(aa)

result = []
for e in range(0,len(d)+1):
    result.append([])
for g in range(0,b):
    part = {}
    frac = {}
    tout = 0
    for e in range(0,len(d)):
        part[e+1] = d[e][g]
    #for each node count edges in and edges out of partition
    for e in range(0,len(d)):
        iin = out =0
        for f in direct[e+1]:
            if string.atoi(part[e+1]) == string.atoi(part[f]):
                iin +=1
            else:
                out +=1
        frac[e+1] = 100*iin/(iin+out+0.0)
        result[e].append(frac[e+1])
        tout += out
    #return sum of nodes out for this partiotioning
    result[e+1].append(tout)
#return the undirected degree for each node
for e in range(0,len(d)):
    result[e].append(len(direct[e+1]))
#return mapping to old id
for e in range(0,len(d)):
    if e+1 in nmap2.keys():
        result[e].append(nmap2[e+1])

file.close()
import fpformat
file2 = open("c:/my documents/bin/partition_eval.txt","w")
end = "\n"
outstring = ""
for t in result:
    for u in t:
        gg = fpformat.fix(u,2)
        outstring = string.join([outstring,gg])
    outstring = string.join([outstring,end])
#print outstring
file2.write(outstring)
file2.close()


#return nodes ranked by percent nodes out
```

:::scc.py:::

```
#Author: Abraham Anderson
#© 2002 The Regents of the University of California

#Finds Strongly Connected Components. And Articulation points. And
bridges.
import pickle
import string
import fpformat

#dictionary[old]=>new
file3 = open("c:/my documents/bin/node_map.txt",'r')
nmap = pickle.load(file3)

nmap2={} #new > old
for y in nmap.keys():
    nmap2[nmap[y]] = y
file3.close()
```

```
#dictionary[from]=>to
file4 = open("c:/my documents/bin/dedges.txt",'r')
direct = pickle.load(file4)
file4.close()

source = 1
#direct={1:[2],2:[3,4],3:[1],4:[5],5:[6],6:[7],7:[5]}
#nmap2={1:1,2:2,3:3,4:4,5:5,6:6,7:7}

#Topo sort
#init
Vt = direct.keys()
#Vt = nmap2.keys()
V = []
for a in Vt:
    V.append(a)
V.sort()
N = len(V)
T=[]+V #if node not in T then its marked
d = {} #distance or time first seen
INF = 1000 #should be infinity
for i in V:
    d[i] = INF
d[source] = 0
d[0] = INF

p = {} #time last seen
dd = {}
pp = {}
t = 0
low = {} #highwater mark for finding articulation point.

#topo sort w/ dfs
def seek(a,t):
    d[a]=t
    low[a]=t
    T.remove(a)
    for c in direct[a] or c not in V:
        if c in T:
            t+=1
            t = seek(c,t)
            if low[c]>=d[a]:             #get art v.2
                if a == source:
                    if len(direct[a])>1: Art.append(nmap2[a])
                else:
                    Art.append(nmap2[a])
            else:                        #
                low[a]=min(low[a],low[c])
        else:
            if c in d.keys():
                low[a]=min(low[a],d[c])
        if c not in V: #this is to deal with nodes that don't have
children
            t+=1
            d[c]=t
            low[c]=t
            t+=1
            p[a]=t
    t+=1
    p[a]=t
    return t

def rseek(a,t,sc):
    scc[a]=sc
    dd[a]=t
    T.remove(a)
    if a in direct2.keys():
        for c in direct2[a]:
            if c in T:
                t+=1
                t = rseek(c,t,sc)
    t+=1
    pp[a]=t
    return t

T2 = []+T
T3 = []+T
Art = []

#topo sort phase
while(len(T)!=0):
    for n in T:
        t = 1+ seek(n,t)

#get the articulation points.
#for v in T2:
#    if d[v] == 0:
#        if len(direct[v]) > 1:
#            #root check
#            Art.append(nmap2[v])
#    else:
#        for u in direct[v]:
#            if low[u] >= d[v]:
#                Art.append(nmap2[v])
ofile = open("c:/my documents/bin/SCC.txt","w")
#print "\narticulation point(s): ",Art #not so good, but OK.
ofile.write("\narticulation point(s): \n")
for u ın Art:
    ofile.write("%s\n" % u)

#get bridges.
Bridge = []
while(len(T3)!=0):
    for v in T3:
        T3.remove(v)
        for u in direct[v]:
            if low[u]==d[u] and low[u]!=low[v]:
                Bridge.append((nmap2[v],nmap2[u]))
#print "\nbridge(s): ",Bridge
ofile.write("bridge(s):\n")
for u in Bridge:
    ofile.write("%s - %s\n" % (u[0],u[1]))

#get inverse graph
```

```
direct2 = {}
for a in V:
    for b in direct[a]:
        if b in direct2.keys():
            temp = direct2[b]
            temp.append(a)
            direct2[b] = temp
        else:
            direct2[b] = [a]

#order nodes by decreasing p-time and topo-sort. At end update comp#
    #get 2 sets: isolated and connected
sc = 0
scc = {}
iso = []
con = []
for x in p.keys():
    if x not in direct2.keys():
        #give each isolated node its own sc#
        iso.append(x)
        scc[x]=sc
        sc+=1
    else:
        con.append(x)

T=con+[] #use nodes with 'reverse' edges
T.sort(lambda x, y: p[y]-p[x])

#get SCCs with rseek
t = 0
while(len(T)!=0):
    #for n in T:
        t = 0
        t = 1+ rseek(T[0],t,sc)
        sc+=1
#put in nice list
sccomps = []
for a in range(0,sc):
    temp = []
    for b in scc.keys():
        if scc[b] == a:
            temp.append(nmap2[b])
    sccomps.append(temp)
#print "\nSCC(s): ",sccomps
ofile.write("SCC(s):\n")
for u in scc.keys():
    ofile.write("%s %d\n" % (nmap2[u],scc[u]))

ofile.write("Spanning edges:\n")
for b in scc.keys():
    #print nmap2[b],scc[b]
    #identify spanning edges:
    for d in direct[b]:
        if d not in scc.keys():
            pass
            #print nmap2[b],"->",nmap2[d]
        elif scc[b] != scc[d] and len(sccomps[scc[b]])>1:
            ##print nmap2[b],"->",nmap2[d]
            ofile.write("%s %s\n" % (nmap2[b],nmap2[d]))

ofile.close()
```

:::step_one.py:::

```
#Author: Abraham Anderson
#© 2002 The Regents of the University of California

import sys
sys.path.insert(0,'c:\\my documents\\bin')
line = sys.argv
import time
import os

badcmd = 1
if len(line) <= 1:
    badcmd = 1
else:
    try:
        name = line[1]
        edge = line[2]
        badcmd = 0

        if os.access('c:/my documents/bin/'+name,1)==0:
            print "Node file not found, retry:"
            badcmd = 1
        if os.access('c:/my documents/bin/'+edge,1)==0:
            print "Edge file not found, retry:"
            badcmd = 1

    except:
        pass

if badcmd:
    print "Name of file with node list?:"
    while 1:
        name = raw_input('\t')
        if name == "":
            name = "blankspace"
        if os.access('c:/my documents/bin/'+name,1)==0:
            print "Not found, retry:"
        else:
            break
    print "Name of file with edge list?:"
    while 1:
        edge = raw_input('\t')
        if edge == "":
            edge = "blankspace"
        if os.access('c:/my documents/bin/'+edge,1)==0:
            print "Not found, retry:"
        else:
            break

#copy them to xnodes.txt and xedges.txt
import shutil
```

```
shutil.copy('c:/my documents/bin/'+name,'c:/my
documents/bin/xnodes.txt')
shutil.copy('c:/my documents/bin/'+edge,'c:/my
documents/bin/xedges.txt')

#this was placed after the SCC code, but since it makes file for input
into the CC code I removed it here.  AA 8.6.01
print "Converting from Access to Metis > graph.metis, node_map.txt,
edges.txt, dedges.txt"
 #need to choose single component for this step. But METIS can handle
separted components in a graph
start = time.time()
import ac2met
print time.time()-start,'s'

print "Converting from Access to BioNet > graph.html"
start = time.time()
import ac2bnt
print time.time()-start,'s'

print "Looking for connected components > CC.txt"
start = time.time()
import cc
print time.time()-start,'s'

print "Looking for strongly connected components, articulation points,
and bridges. > SCC.txt"
start = time.time()
import scc
print time.time()-start,'s'

print "Calculating Centralities > D_centrality.txt, U_centrality.txt"
#These have their own timers
import get_centrality
import get_centralityU

print "Summaraizing topology and centrality results > summary.txt"
start = time.time()
import x_SUMMARY
print time.time()-start,'s'

print "Run Partitoner(Y/N):"
while 1:
    c = raw_input("\t")
    if c == 'Y' or c == 'y':
        import step_two
        break
    elif c == 'N' or c == 'n':
        print "OK."
        break
    else:
        print "Type Y or N."

statusfile = open("c:/my documents/bin/xstatus.txt",'w')
statusfile.write(name+';'+edge)
statusfile.close()
print "Done w/ Analysis"
```

:::step_two.py:::

```
#Author: Abraham Anderson
#© 2002 The Regents of the University of California

#analyse partition files:
#run metis
import os
import string
import fpformat
def runMETIS():
    print "Partitioning graph.metis into n peices.\n"
    print "Smallest number of partitions:"
    while 1:
        c = raw_input("\t")
        break
    print "Largest number of partitions:"
    while 1:
        p = raw_input("\t")
        break
    com = "c:\\progra~1\\metis-4.0-compiled\\pmetis.exe
c:\\mydocu~1\\bin\\graph.metis"
    for a in range(string.atoi(c),string.atoi(p)+1):
        command = string.join([com,fpformat.fix(a,0)])
        os.system(command)

runMETIS()

#assemble partitions into one file: x_partitons.txt
def assembleP():
    #read all metis files into list
    print "Assembling all partition results to one file:
x_partitions.txt\n"
    mydir = "c:\\mydocu~1\\bin\\"
    files = os.listdir(mydir)
    junkf = []
    for a in files:
        if string.count(a,"graph.metis.part") == 0:
                junkf.append(a)
    for a in junkf:
        files.remove(a)
    dict = {}
    for a in files:
        b = string.split(a,".")
        dict[string.atoi(b[3])] = a
    ks = dict.keys()
    ks.sort()
    files = []
    for a in ks:
        files.append(dict[a])
    print files #
    rfiles = []
    for a in files:
        rfiles.append(open(string.join([mydir,a],''), 'r'))
    placehold = open(string.join([mydir,files[1]],''),'r')
    outfile = open(string.join([mydir,'x_partitions.txt'],''),'w')
    #for each file read a line and print to output file with tab delim
    while placehold.readline():
        outline = ""
        for a in rfiles:
            b = a.readline()
            b = string.replace(b,'\012','\t')
            outline = string.join([outline,b],'')
        outline = string.join([outline,"\n"],'')
        outfile.write(outline)

    placehold.close()
    outfile.close()
    for a in rfiles:
        a.close()

assembleP()

import sys
sys.path.insert(0,'c:\\my documents\\bin')

print "Looking for nodes on the edge of the partitions, and writing to
partition_eval.txt.\n"
import peval
```

:::x_SUMMARY.py:::

```
#Author: Abraham Anderson
#© 2002 The Regents of the University of California

import string
#this prog was originally just for obesity results...
#dir = 'c:/my documents/bin/obesity_res/raw/'
#..but now its for generic results.
dir = 'c:/my documents/bin/'

#cc file
#scc file
#Centrality file(s)
#table summary: id, cc[#,-1], scc[#,-1], articulation pt[1/0], bridge
[0,-1,1], span[0,-1,1], Ucloseness[#,-1], Ubetweenness, Dclose, Dbet...
#8-2-01 added group centrality scores for significant edges
#8-8-01 added undirected group centrality scores
table = {} #dictionary[id:vector]; vector = above data
file = open(dir+'CC.txt','r')
for line in file.readlines():
    line = line.rstrip()
    [id,cc] = string.split(line,'\t')
    try:
        table[id][0] = cc
    except:
        table[id] = ['-1','-1','0','0','0','-1','-1','-1','-
1','0','0','0','0']
        table[id][0] = cc
file.close()

file = open(dir+'SCC.txt','r')
while string.count(file.readline(),'articulation') != 1:
    pass
while 1:
    line = file.readline()
    if string.count(line,'bridge') != 1:
        pass
    else:
        break
    line = line.rstrip()
    [id] = string.split(line,'\t')
    try:
        table[id][2] = '1'
    except:
        table[id] = ['-1','-1','0','0','0','-1','-1','-1','-
1','0','0','0','0']
        table[id][2] = '1'

bridges = []
while 1:
    line = file.readline()
    if string.count(line,'SCC') != 1:
        pass
    else:
        break
    line = line.rstrip()
    [fr,to] = string.split(line,' - ')
    bridges.append([fr,to])
    try:
        table[fr][3] = '-1'
    except:
        table[fr] = ['-1','-1','0','0','0','-1','-1','-1','-
1','0','0','0','0']
        table[fr][3] = '-1'
    try:
        table[to][3] = '1'
    except:
        table[to] = ['-1','-1','0','0','0','-1','-1','-1','-
1','0','0','0','0']
        table[to][3] = '1'

while 1:
    line = file.readline()
    if string.count(line,'Spann') != 1:
        pass
    else:
        break
    line = line.rstrip()
    [id,sc] = string.split(line,' ')
    try:
        table[id][1] = sc
    except:
        table[id] = ['-1','-1','0','0','0','-1','-1','-1','-
1','0','0','0','0']
        table[id][1] = sc

spans = []
for line in file.readlines():
    line = line.rstrip()
    [fr,to] = string.split(line,' ')
    spans.append([fr,to])
    try:
        table[fr][4] = '-1'
    except:
        table[fr] = ['-1','-1','0','0','0','-1','-1','-1','-
1','0','0','0','0']
        table[fr][4] = '-1'
    try:
        table[to][4] = '1'
    except:
        table[to] = ['-1','-1','0','0','0','-1','-1','-1','-
1','0','0','0','0']
        table[to][4] = '1'

file.close()

file = open(dir+'U_centrality.txt','r')
while string.count(file.readline(),'Vertex') != 1:
    pass
for line in file.readlines():
    line = line.rstrip()
    [id,b,c] = string.split(line,' ')
    try:
        table[id][5] = c
        table[id][6] = b
    except:
        table[id] = ['-1','-1','0','0','0','-1','-1','-1','-
1','0','0','0','0']
        table[id][5] = c
        table[id][6] = b
file.close()

file = open(dir+'D_centrality.txt','r')
while string.count(file.readline(),'Vertex') != 1:
    pass
for line in file.readlines():
    line = line.rstrip()
    [id,b,c] = string.split(line,' ')
    try:
        table[id][7] = c
        table[id][8] = b
    except:
        table[id] = ['-1','-1','0','0','0','-1','-1','-1','-
1','0','0','0','0']
        table[id][7] = c
        table[id][8] = b
file.close()

#remove false entries:
g = 0 #number of nodes
for id in table.keys():
    if table[id][5] == '-1':
        if table[id][6] == '-1':
            if table[id][7] == '-1':
                if table[id][8] == '-1':
                    del table[id]
                    continue
    g = len(table.keys())

for id in table.keys():
    neighbors = []
    for e in bridges:
        if id == e[0]:
            if e[1] in table.keys():
                neighbors.append(e[1])
        if id == e[1]:
            if e[0] in table.keys():
                neighbors.append(e[0])
    if len(neighbors)==0:
        pass
    else:
        #calc group centrality
        gc = 0
        m = 0
        mn = ''
        neighbors.append(id)
        for n in neighbors: # find max centrality score
            if m > float(table[n][8]):
                pass
            else:
                mn = n
                m = float(table[n][8])
        for n in neighbors:
            gc = gc + (float(table[mn][8])-float(table[n][8]))/((g-
1)*(g-2)/2)
        table[id][9] = '%f' % (gc/(len(neighbors)-1))
        #calc group undirected centrality
        gc = 0
        m = 0
        mn = ''
        neighbors.append(id)
        for n in neighbors: # find max centrality score
            if m > float(table[n][6]):
                pass
            else:
                mn = n
                m = float(table[n][6])
        for n in neighbors:
            gc = gc + (float(table[mn][6])-float(table[n][6]))/((g-
1)*(g-2)/2)
        table[id][11] = '%f' % (gc/(len(neighbors)-1))
    neighbors = []
    for e in spans:
        if id == e[0]:
            if e[1] in table.keys():
                neighbors.append(e[1])
        if id == e[1]:
            if e[0] in table.keys():
                neighbors.append(e[0])
    if len(neighbors)==0:
        pass
    else:
        #calc group centrality
        gc = 0
        m = 0
        mn = ''
        neighbors.append(id)
        for n in neighbors: # find max centrality score
            if m > float(table[n][8]):
                pass
            else:
                mn = n
                m = float(table[n][8])
        for n in neighbors:
            gc = gc + (float(table[mn][8])-float(table[n][8]))/((g-
1)*(g-2)/2)
        table[id][10] = '%f' % (gc/(len(neighbors)-1))
        #calc group undirected centrality
        gc = 0
        m = 0
        mn = ''
        neighbors.append(id)
        for n in neighbors: # find max centrality score
            if m > float(table[n][6]):
                pass
            else:
                mn = n
                m = float(table[n][6])
        for n in neighbors:
            gc = gc + (float(table[mn][6])-float(table[n][6]))/((g-
1)*(g-2)/2)
        table[id][12] = '%f' % (gc/(len(neighbors)-1))


file = open(dir+'summary.txt','w')
file.write('ID,CC,SCC,Art,Bridge,Span,UCC,UBC,DCC,DBC,bgcs.d,sgcs.d,bgc
s.u,sgcs.u\n')
for id in table.keys():

file.write(id+','+table[id][0]+','+table[id][1]+','+table[id][2]+','+ta
ble[id][3]+','+table[id][4]+','+table[id][5]+','+table[id][6]+','+table
[id][7]+','+table[id][8]+','+table[id][9]+','+table[id][10]+','+table[i
d][11]+','+table[id][12]+'\n')
    #print id,table[id]
file.close()
```