UCSF
UC San Francisco Electronic Theses and Dissertations

Title
Finding spatially conserved residues in protein structure

Permalink
https://escholarship.org/uc/item/9q2194x0

Author
Chuang, Jer-Yee John

Publication Date
2005

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

Finding Spatially Conserved Residues in Protein Structure Alignments

by

Jer-Yee John Chuang

THESIS

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

BIOLOGICAL AND MEDICAL INFORMATICS

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA SAN FRANCISCO

Author: ________ Date: 6/15/2005

Approved by:

Thomas E Ferrin Date: 6/15/2005

Conrad Huang Date: 6/15/2005

Elaine C. Meng Date: 6/15/2005

UCSF

University of California, San Francisco
OFFICE OF ADMISSION AND REGISTRAR (OAR)

PRINT: Chuang Jer-Yee
Last name, First Middle

Grad / Biological & Medical Informatics 433-67-7725
school/major student number

Instructions: State request and give your reasons in detail. Use back of petition if necessary. Obtain signature approvals as directed by deputy. File petition with Registrar.

Univeristy Registrar,

I wish to petition to leave my academic program (PhD program in Biological and Medical Informatics) with a terminal Masters of Science degree. I have fulfilled all the requirements for the MS, and have gotten the approval of both my academic adviser and program chair.

Student's Signature: **Date:** 6/15/05

☐ **DEAN:** ☐ **STUDENT AFFAIRS:**

☐ **ADVISOR:** ☐ **CURRICULAR AFFAIRS:**

☐ **INSTRUCTOR:** ☐ **GRADUATE DIVISION:**

☒ **THESIS / DISS. CHAIR:** Thomas E Ferrin 6/15/05 ☐

c:\wpfiles\as\stud.pet
rev2/2000

Copyright 2005
by
Jer-Yee John Chuang

Abstract

With the wealth of protein structures determined in recent structural genomics initiatives, the task of identifying and characterizing functional residues will be of increasing importance. Traditional approaches relying on sequence conservation analysis are insufficient, as sequence conservation does not imply structure conservation. In this study, we describe the implementation and testing of an algorithm to efficiently identify spatially conserved amino acid residues from a set of aligned protein structures. The algorithm was applied to several diverse sets of proteins, including cAMP binding proteins and triosephosphate isomerases. In the former set, our algorithm was able to recover not only residues identified via traditional surface matching methods, but also an important conserved hydrophobic residue not apparent from the multiple sequence alignment. In the latter set, we compared our results with those from mutagenesis experiments. Our algorithm's performance in identifying functionally significant residues was comparable, but had the additional benefit of being automated. Lastly, we demonstrated two further applications of our method: as a quantitative measure of alignment quality to supplement traditionally reported RMSD values and as a technique for characterizing information redundancy when used in conjunction with leave-one-out testing.

Acknowledgements

The completion of this thesis would not have been possible without the generous encouragement and support from a number of individuals. First, I would like to acknowledge my research adviser Tom Ferrin; and Conrad Huang and Elaine Meng of the UCSF Computer Graphics Lab (CGL). Your ideas and suggestions have been invaluable to this work, and I am truly grateful for the opportunity to have worked with you. In addition, my thanks go out to the other members of the CGL that have contributed to this project: Dan Greenblatt with Python coding, Eric Pettersen with the Multalign Viewer tool in Chimera, and Tom Goddard with the `drawbox` routines. I want to extend my deepest thanks to my academic adviser here at UCSF, Patricia Babbitt, and my former adviser at Duke University, Alex Hartemink. Without your guidance in my intellectual development, I would not be where I am today.

Many friends have helped to carry me through my journey at UCSF these past three years. I want to acknowledge:
Colleagues: Erik Hom, Hector Aldaz, Muluye Liku, Michael Cohen, and Richard Tjhen.
Fellow BMI classmates: Juanita Li, Simona Carini, Christina Chaivorapol, Ranyee Chiang, Libusha Kelly, Mike Kim, Mark Peterson, and Tuan Pham.

Outside of UCSF, I have been blessed by the prayers and the encouragement of dear friends: Anna Tseng, Susan Lee, Mike Toy, Mike Yee, Hubert Chen, Erika Gee, Mindy Kim, Jason Donald, and Tan Gao.

I have saved my most important acknowledgements for last. Truly, I owe everything to my family- my parents Cherng-Zee and Jin-Chuan, my brother Jer-Chin, and my dog Poggi. You are the most precious gifts that I have been blessed with this side of heaven. Finally, I want to give thanks to my Father in heaven. Lord, may this work and all that I do bring you glory.

Table of Contents

Abbreviations

Chapter 1: Introduction.. 1

Chapter 2: The voxelGrid Algorithm 3

2.1 Basic Algorithm... 3

2.2 Implementation and Enhancements... 4

2.2.1 Code base and platforms..4

2.2.2 Residue groups...6

2.2.3 Neighborhood query..6

2.2.4 Side-chain centroid...9

2.3 Running voxelGrid..10

2.3.1 Quick start...10

2.3.1.1 Understanding voxelGrid command line flags..........10

2.3.1.2 Interpreting voxelGrid output..............................12

2.3.2 Performance considerations..13

2.3.2.1 Effect of voxel size..13

2.3.2.2 Effect of random offsetting..................................15

2.3.2.3 Systematic vs. random grid offsetting....................16

2.3.2.4 Granularity of systematic sampling......................18

2.3.2.5 Parallelization..19

2.4 Dependence of voxelGrid Algorithm on Alignment Quality................19

2.4.1 Considerations..19

2.4.2 Preparation of alignments..20

2.4.3 Example: Comparison of structure alignment web servers.......22

Chapter 3: Example: the cAMP Binding Proteins 25

3.1 Introduction to cAMP Binding Proteins...25

3.2 Analysis of Four Representative cAMP Binding Structures................26

3.2.1 Preparation of datasets and results.....26
3.2.2 Analysis.....28
3.3 Analysis of Broad Set of cAMP Structures.....29
3.3.1 Preparation of dataset.....30
3.3.2 Results and analysis.....30
3.4 Measuring Information Redundancy.....33

Chapter 4: Example: the TIM Proteins **35**
4.1 Introduction to TIM Proteins.....35
4.2 The Open and Closed Conformations.....36
4.2.1 Preparation of dataset and results.....36
4.2.2 Analysis.....38
4.3 Analysis of a Diverse Set of TIM Proteins.....39
4.3.1 Preparation of dataset.....39
4.3.2 Results.....40
4.3.3 Analysis: Comparison of sequence and structure alignments.....41
4.4 Comparison of voxelGrid with CoC Predictions.....44
4.5 Comparison of voxelGrid with Mutagenesis Results.....46
4.5.1 Large scale mutagenesis of TIM residues.....46
4.5.2 Comparing mutagenesis results with CoC and voxelGrid.....48
4.5.2.1 Mutagenesis results and voxelGrid.....48
4.5.2.2 Mutagenesis results and the CoC.....49
4.5.2.3 Significance of residues inside and outside the hydrophobic core.....49

Chapter 5: Conclusion.....**52**

Appendix

References

Abbreviations

cAMP: cyclic AMP
CAP: catabolite activator protein
CDD: NCBI conserved domain database
CoC: conservation of conservation database
DHFR: dihydrofolate reductase
HCN: hyperpolarization-activated cyclic nucleotide-modulated channels
MAV: Multalign Viewer tool in Chimera
MSA: multiple sequence alignment
NQ: neighborhood query
PBC: phosphate binding cassette
PDB: protein data bank
PKA: protein kinase A
RMSD: root mean square distance
SFLD: structure-function linkage database
TIM: triosephosphate isomerase

Chapter 1: Introduction

As the structures of more proteins become available, automating annotation of protein function using structure becomes increasingly important. For enzymes, a superfamily classification system is particularly useful. This system is based on conservation of a partial chemical reaction amongst superfamily members and presupposes a tight coupling between conserved functional and conserved structural characteristics [1]. Frequently, these conserved structural characteristics are a small number of key active site residues that directly participate in the enzymatic reaction. These residues are subject to strong evolutionary constraints [2] and have been demonstrated to be sufficient for use as a template in assigning superfamily membership [3]. Significant efforts have been made on automated identification of active sites in proteins. These techniques run the gamut from mathematical data mining [4], to structural motif matching [5], to spatial clustering and averaging of residues bound to a ligand [6]. For enzymes with identical functions, it may possible to identify many if not all active site residues from a multiple sequence alignment (MSA). However, sequence conservation does not imply structure conservation, and we postulate that it is the conserved structural features of a structure that give rise to function.

Furthermore, studies on enzyme catalysis highlight the fact that the catalytic elements in an active site need to be precisely positioned for optimal catalysis, and this often requires a transient reorganization of the active site or more distal regions of the protein [7]. An excellent example of this was described in dihydrofolate reductase (DHFR) by Agarwal and coworkers [8]. In their study, they combined kinetic measurements from multiple mutation experiments and molecular dynamics simulations to characterize the conformational change of a 14-residue loop distal to the active site upon substrate binding. Movement of the loop was shown to facilitate the hydride transfer between the substrate and cofactor. A similar example is provided by triosephosphate isomerase (TIM), which also has a movable loop. In this case, the motion of the loop is more pronounced and controls access to the active site in a lid-like fashion upon substrate binding [9]. Identification of spatially conserved residues throughout the entire protein structure, and not only in the active site, would be most useful.

Most studies that examine the entire protein for conserved residues perform the search only on the sequence level. For instance, Li and coworkers used a combination of an MSA and mutual information[1] to predict residues responsible for enzyme-substrate specificity in protein kinases [10]. This is understandable, as sequences are far more plentiful and easier to obtain than structures. However, the number of structures available is increasing rapidly. Additionally, structural characteristics such as the movable loop discussed above cannot be identified solely from sequence analysis. There may also be spatially conserved residues that are not conserved in the MSA. Hence, it would make more sense to integrate both sequence and structure information when identifying spatially conserved residues, and we propose such a method.

---

[1] Mutual information is defined as the relative entropy between the joint distribution and the product distribution.

Chapter 2: The voxelGrid Algorithm

2.1 Basic Algorithm

voxelGrid is an algorithm that identifies spatially conserved amino acid residues in proteins. At its simplest, it is a residue counting algorithm over a set of aligned structures. The algorithm begins by first reading in a set of aligned structures then overlaying a voxel grid[2] (Figure 2.1). It then examines each voxel, counting up the number of residues contributed by each structure. Voxels that contain a residue of the same type from every structure are saved. By nature of residing in the same voxel, these residues are in proximity and therefore spatially conserved.

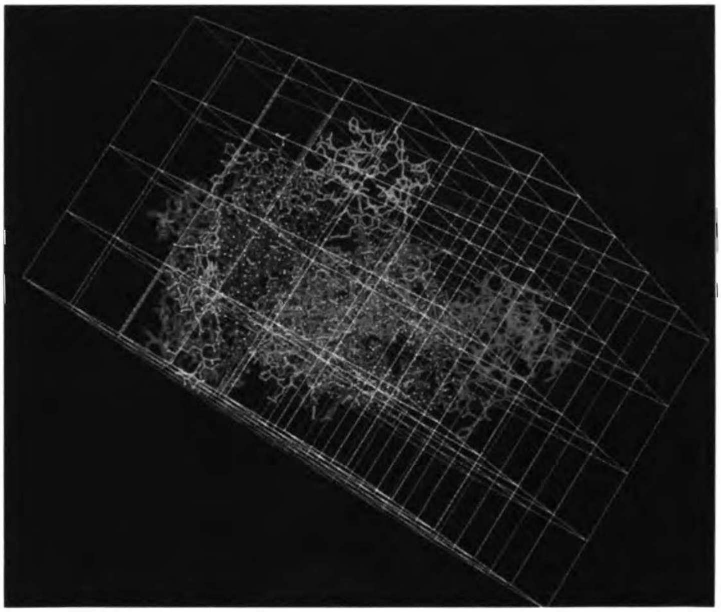


Figure 2.1: Example of a voxel grid overlaid on a set of aligned structures

From Figure 2.1, it is evident that there are two key parameters that can be varied: the size of each voxel (i.e., the granularity of the grid) and the position of the grid in space (i.e., offset). Care must be taken in defining both. The size of a voxel determines the

---

[2] A voxel grid is a 3D rectangular grid, where each cell is called a voxel. A voxel is the generalization of a pixel from 2D to 3D (i.e. square pixels in 2D become cubic voxels in 3D).

scope of what is considered spatially conserved. For example, the voxels in Figure 2.1 are much too large and are shown for visual representation only. A more thorough discussion of the effect of voxel size is provided later in Section 2.3.2.1. The position of the grid is also important, since depending on the placement of the grid, it may unintentionally separate a cluster of spatially conserved residues. To overcome this, the voxelGrid algorithm may be run multiple times, each with a different grid offset. We keep track of the list of conserved residues for each run. By taking the union of all these lists and removing duplicate entries, we arrive at a "complete" list of all spatially conserved residues for the set of aligned structures. Furthermore, we observe that voxelGrid is much more efficient than computing the inter-residue distances among all structures. Given M structures of N residues each, our algorithm scales as $O(M*N)$ compared to $O(M^2*N^2)$ for the latter.

The above discussion highlights the basic voxelGrid algorithm. The subsequent sections in this chapter will cover its implementation and execution. In addition, we discuss the effects of voxel size, grid placement, and the quality of the input structure alignments on voxelGrid performance. Finally, we conclude this chapter with an example of how voxelGrid can be used as a quantitative measure of structure alignment quality in addition to traditional RMSD values.

2.2 Implementation and Enhancements

2.2.1 Code base and platforms

The voxelGrid algorithm was coded in C++ and has been successfully built on Windows and Linux platforms. We recommend compiling with the '-static' flag to increase portability. C++ was chosen as the language for performance considerations and to utilize an existing C++ library for parsing PDB [11] files. Briefly, the voxelGrid algorithm class hierarchy is:

voxelGrid(main) ← Voxel(class) ← VoxStruct(class) ← VoxResidue(class)

Voxel: a voxel in our grid
VoxStruct: a single protein structure
VoxResidue: a single residue in a protein structure

We begin by reading in the set of aligned structures. To reduce the complexity of the calculations, the hydrogen atom coordinates are ignored, and each residue is only represented by its centroid (however, the coordinates of all atoms are still preserved). In addition, no adjustment is made for the molecular weight of each type of atom in computing these *whole-residue* centroids. The minimum and maximum centroid coordinates in each dimension across all of the structures define the boundaries of our voxel grid. The residues in each structure are binned into the corresponding voxels. If two residues from the same structure map into the same voxel, we issue a warning message. Once the residues have been binned, we can then examine each voxel for *identical* residues from each structure. If so, we have found a spatially conserved residue (i.e., a 'hit') and write it to an output file.

Python scripting was used to repeatedly run the voxelGrid algorithm with varying offsets to the voxel grid. This is necessary since the grid may be placed such that it inadvertently cuts through a cluster of spatially conserved residues. An additional Python script was used to merge all the output files into two script files for visualization in Chimera [12]: a script that displays the conserved residues found by voxelGrid and another that draws the voxels enclosing those residues (examples of these scripts can be found in the Appendix).

We now discuss three enhancements to the basic voxelGrid algorithm: residue groups, neighborhood query, and side-chain centroids. Only neighborhood query is enabled by default, though each of these can be enabled/disabled on the voxelGrid command line (Section 2.3.1.1).

2.2.2 Residue groups

We wanted to extend the comparison of residues within a voxel to include not only those that are identical but those that belong to the same group (e.g., polar, hydrophobic, etc.). Residue groups allow for matching residue substitutions in the structural alignment. This is particularly useful when it is known that the structures have diverse sequences, and voxelGrid returns few hits.

Residue groups were implemented through an optional plain text file (named 'residueGroups' that is placed in the same directory as the voxelGrid executable). This file contains user-defined groups described with the standard three-letter amino acid codes. These groups do not have to be based on chemistry (can be whatever the user defines), and not every amino acid must belong to a group. Furthermore, group membership is not mutually exclusive, so an amino acid may belong to multiple groups. Comments are allowed and preceded by '#'. An example of a 'residueGroups' file is given in the Appendix. If the 'residueGroups' file does not exist, then voxelGrid automatically uses the default setting of 20 groups which tests for amino acid identity.

2.2.3 Neighborhood Query (NQ)

As the voxelGrid algorithm examines each voxel, there will be instances when it finds that the voxel contains a residue from all but one of the structures. This is not too surprising, since that structure may not be well aligned with the others such that its residue lies in an adjoining voxel. Alternatively, our voxel grid may unintentionally separate a cluster of spatially conserved residues. To account for these possibilities, voxelGrid should be able to look in the neighboring voxels for a residue from the missing structure and associate it with the current voxel.

Neighborhood Query Algorithm:

NQ is an expensive operation since it needs to look in the 26 adjacent voxels. Hence, it should only be used in limited situations, such as when the voxel being examined is missing *exactly* one structure. If so, then:

1. In the neighboring voxels, determine the residues from the missing structure.
2. For each residue, if inclusion in the reference voxel will satisfy the residue threshold (see Section 2.3.1.1) then keep, else discard this residue. Observe that if inclusion of the residue will not satisfy the residue threshold, it will not be output anyway.
3. For each remaining residue, compute the pairwise Euclidean distance to all the residues in the reference voxel. Select the residue closest to an existing residue in the reference voxel, and report it as a *NQ* hit. This is in contrast to *voxel* hits, those hits that are found without NQ[3].

To get an idea of how NQ works, consider the following example using an alignment of four structures (Figure 2.2). For simplicity, we focus on a single voxel shown here with polar residues from three different structures (yellow, cyan, and magenta). NQ identifies residues from the fourth structure (white) in adjacent voxels. The closest residue has been colored red and is selected by NQ for this voxel.

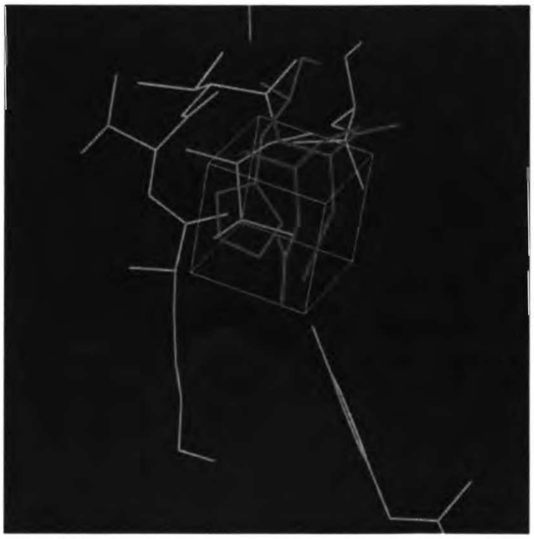


Figure 2.2: Example of neighborhood query

---

[3] For this study, all hits reported are voxel hits unless otherwise specified as NQ hits.

It should be noted that NQ assumes nothing about the correctness of the structure alignments. It will *not* solve the case where the voxel has a residue from each structure, but one of the residues is incorrect (i.e., due to misalignment, the correct residue is in an adjacent voxel). Rather, NQ allows us to deal with poor alignments that contain a single outlier. Additionally, the grid is overlaid on the structure alignments prior to examining the voxels, and therefore, the grid's placement affects which voxels NQ is performed on. For instance, the grid may be placed such that some voxels are missing more than one structure. Since NQ allows at most one missing structure, NQ will not be performed on those voxels. Lastly, when merging the output files from multiple voxelGrid runs, we discard the NQ hit if an identical voxel hit exists from another run.

It should be observed that NQ is not the same as simply running voxelGrid multiple times using different grid offsets. NQ is able to recover residues farther away than random offsetting alone. Consider Figure 2.3, which shows the residues from an alignment of three structures (numbered 1-3). We have simplified the voxel representation to 1D for illustration purposes.

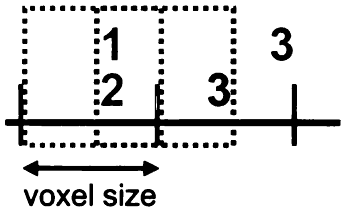


Figure 2.3: Difference between random offsetting and NQ

The dotted red box shows the starting voxel location. Random offsetting allows the voxel grid to shift by at most the size of one voxel (Section 2.3.1.1), allowing us to recover the third residue (dotted green box). However, if the third residue were located farther out (the '3' colored blue instead of green), then random offsetting would not be able to recover it. If random offsetting were to position the voxel to include the blue-colored '3', the first two residues would be excluded. In contrast, NQ is able to recover the third residue at either location.

2.2.4 Side-chain centroid

By default, voxelGrid computes the centroids of *whole* residues. Alternatively, the centroid could be calculated based on only the residue side-chain, yielding the *side-chain* centroid. This option can be enabled with a flag on the voxelGrid command line (see Section 2.3.1.1). When this option is enabled, the backbone atoms of a residue are excluded from the centroid calculation.

We compared the two centroid calculation methods using a set of four aligned structures and varying voxel sizes (Table 2.1). A more detailed guide to interpreting voxelGrid results is given in Section 2.3.1.2. For now it suffices to note that the values given in the 'Whole-Residue' and 'Side-Chain' columns are the number of voxel and NQ hits found by voxelGrid (reported as voxel/NQ). We observed that the two methods performed comparably. Using whole-residue centroids yields slightly more voxel hits, while side-chain centroids reduce the number of NQ hits. This is probably because the structures were aligned using alpha carbons (i.e., C_α). Hence, the backbone will be aligned better than the side chains.

Voxel Size	Step Size	Whole-Residue	Side-Chain
N=2.00	0.26	24/23	22/22
N=2.25	0.30	27/35	25/31
N=2.50	0.33	31/56	29/35
N=2.75	0.375	35/67	31/43
N=3.00	0.40	41/83	37/63

Table 2.1: Comparison of whole-residue and side-chain centroids

2.3 Running voxelGrid

2.3.1 Quick start

2.3.1.1 Understanding voxelGrid command line flags:

Running voxelGrid without any or with an incorrect number of parameters yields a summary of command line parameters and usage:

```
voxelGrid, written by Jer-Yee John Chuang (chuang@gmail.com)

Usage: voxelGrid.exe xVox yVox zVox [-r <xOff yOff zOff>] [-sc] [-xnq] [-t
strThr resThr] file1 [file2...]

Explanation:
xVox yVox zVox        voxel size in xyz
-r xOff yOff zOff     fix grid at these offsets (default: random)
-sc                   use side-chain centroid (default: whole-residue)
-xnq                  disable neighborhood query (default: enabled)
-t strThr resThr      set structure/residue thresholds (default: #files)
file1 file2...        first PDB file, second PDB file, ...
```

At minimum, voxelGrid must be provided with a set of aligned structures and the voxel size. If a set of aligned structures is not available, it can be obtained in a variety of ways, such as through structure alignment web servers (see Section 2.4.2). voxelGrid assumes default values for all other parameters not specified via flags on the command line. This is what is meant when we use the phrase running voxelGrid with 'default settings' later in this study. Note that residue groups can be specified is a separate 'residueGroups' file.

It is useful to briefly discuss each of the flags used by voxelGrid. The '-sc' flag instructs voxelGrid to compute side-chain centroids, overriding the default use of whole-residue centroids. The '-xnq' flag disables NQ in voxelGrid, resulting in faster performance at the cost of search sensitivity (i.e., no NQ hits).

The '-r' flag is used to fix the grid at a particular offset. By default, the grid is randomly offset from the structure alignment; otherwise, repeated runs of voxelGrid will give the same results. The size of this offset is at most the size of one voxel and is determined for each dimension independently. However, there are two instances where it is desirable to fix the grid at a known location. The first is for reproducibility of results. Using the same aligned structures, voxel size, and grid offset will always yield the same list of spatially conserved residues. The second instance is when we want to perform a search of the structure space by overlaying a grid then systematically shifting the grid in each dimension independently (see Section 2.3.2.3).

The '-t' flag is used to redefine the structure and residue thresholds. Basically, these two parameters define the criteria for a voxel hit in voxelGrid. Given a voxel, both thresholds need to be satisfied in order to be considered a voxel hit (i.e., the residues in the voxel are spatially conserved). The structure threshold specifies the minimum number of structures that must be present in a voxel. Likewise, the residue threshold specifies the minimum number of residues that belong to the same residue group that must be present in a voxel. By default, we use the most conservative setting: both equal to the number of structures being examined. This means that for a voxel to be considered a voxel hit, (i) it must have at least one residue from every structure, and (ii) the residue group with the largest representation has as many members as the number of structures. Finally, note that relaxing the structure threshold does not affect the NQ algorithm. The NQ algorithm only considers voxels that are missing exactly one structure, which may not be the same as (strThr-1).

A brief example may help to clarify the usage of the '-t' flag. Consider running voxelGrid on an alignment of four structures and without a 'residueGroups' file (i.e., default of 20 residue groups corresponding to amino acid identity). In some voxel, we observe a residue from each structure and the following makeup: Arg, Arg, Asp, Glu. This voxel will *not* be returned as a voxel hit since it satisfies the structure but not the residue threshold (it only has two residues in the same group: Arg). It is true that we have residues from each structure, but we are looking for spatial conservation of *identical* residues. If we truly wanted this voxel to be counted as a voxel hit, then there are two ways to rectify this. The simplest is to relax the residue threshold from four to two.

Alternatively, we can use a 'residueGroups' file and define a polar group[4] that includes the three residue types above (Arg, Asp, Glu). Now, the residue threshold is satisfied (four residues in the same group). More importantly, by using residue groups, we have changed from looking for spatially conserved *identical* residues to looking for spatially conserved *polar* residues.

2.3.1.2 Interpreting voxelGrid output:

The voxelGrid output from a single run is a plain text file containing all the residues that were identified as spatially conserved. As discussed previously, it is advisable to run voxelGrid repeatedly with different grid offsets, and then merge the results using a separate script. This returns the number of voxel and NQ hits found across all the merged runs. These are the numbers that are reported in our analysis, usually in the format (voxel/NQ). Consider the example given in Table 2.2. Here, voxelGrid has been run at three different voxel sizes for three datasets. Datasets 1 and 2 have both their voxel and NQ hits reported (as voxel/NQ), whereas Dataset 3 only has its voxel hits reported. Unless otherwise noted in this study, pairs of numbers separated by a '/' are always voxel/NQ hits.

Voxel Size	Dataset 1	Dataset 2	Dataset 3
N=2.50	17/14	18/7	23
N=2.75	21/21	26/9	27
N=3.00	28/29	35/8	38

Table 2.2: Example of results from using voxelGrid analysis on a dataset

Observe that in Dataset 2, the number of NQ hits decreases from N=2.75 to N=3.00. This is normal, as the number of NQ hits does not necessarily increase with increasing voxel size. In addition to the number of voxel and NQ hits, two readable output files (example can be found in the Appendix) are created by voxelGrid. These can be used to visualize the results in Chimera.

---

[4] See the Appendix for an example of a 'residueGroups' file that defines a polar group.

2.3.2 Performance considerations

2.3.2.1 Effect of voxel size:

The size of a voxel determines the scope of what is considered spatially conserved. There is no single best value, since this depends on the alignment quality[5] of the set of structures being examined. For well-aligned structures, smaller voxel sizes are better as they yield higher resolution (i.e., fewer false positives). On the other hand, if the alignment is poor, then larger voxels should be used. We note that although voxelGrid does allow non-cubic voxels, we only considered cubic ones in this study.

As a rule of thumb, voxelGrid should be executed for a small number of runs using random offsetting at several voxel sizes. Two factors should be considered when examining the results:

1. Number of hits: If 20% of the protein is being pulled back as hits, then obviously the voxel size should be decreased. The actual percentage cutoff is subjective. For well-studied systems, it should be large enough to recover known conserved functional residues.
2. Residue double-counting: For larger voxel sizes, it is possible that multiple residues from the same structure are mapped into the same voxel. When this happens, voxelGrid will write a warning message into the output file and specify for each structure, the number of residues that have been 'double-counted'. This double-counting is undesirable as it makes it difficult to interpret which residue should be correctly associated with the others in the voxel. Ideally, only one residue per structure maps into the same voxel. Although some double-counting is unavoidable as the voxel size is increased, it should be kept to a minimum (just a few residues).

---

[5] The alignment quality also depends on the conformational flexibility of the structures. For simplicity, we will not consider that aspect in our study.

Residue conservation histogram:

In addition to containing the list of spatially conserved residues identified, the voxelGrid output file also contains a residue conservation histogram. This histogram reflects the distribution of all voxels that satisfy the structure threshold criteria. It provides an indirect way to assess the effect of voxel size on performance. Each bin in the histogram corresponds to the maximum number of residues in any of the residue groups. Hence, the bins range from zero up to the number of structures in the set of alignments.

Consider the following example. Suppose we run voxelGrid on an alignment of four structures and obtain a residue conservation histogram of [0 0 6 3 3]. This means that 12 voxels satisfied the structure threshold. Each of these 12 voxels contained at least one residue from each of the four structures. There were three voxels that contained four residues in the same residue group, and another three voxels that contained three residues in the same residue group. Six other voxels only contained two residues in the same residue group.

Note that the number of elements in this histogram may exceed the number of voxel hits since it may contain voxels that satisfy the structure threshold but fail to satisfy the residue threshold criteria. We ran voxelGrid using random offsetting and default settings 10 times for varying voxel sizes on an alignment of four structures (~230 residues each). Since we are using random offsetting, we averaged the residue conservation histograms from these runs. The value of the structure and residue thresholds is four (i.e., the number of structures). Results are presented in Figure 2.4.

```
Voxel Size    Residue Conserv. Histogram (Avg. over 10 runs)
-----------------------------------------------------------
N=1.0         [0.0  0.0  0.0  0.0  0.7]
N=1.5         [0.0  0.0  0.2  0.2  3.0]
N=2.0         [0.0  0.0  1.0  0.6  5.3]
N=2.5         [0.0  0.0  1.3  1.4  7.5]
N=3.0         [0.0  0.0  2.5  3.9  8.5]
N=3.5         [0.0  0.0  4.0  5.2  14.1]
```

Figure 2.4: Effect of voxel size on the residue conservation histogram

We observed that for voxel sizes less than N=2.0, there are only a few residues found. Between N=2.0 and N=3.0, we found 7-15 residues. With systematic offsetting (Section 2.3.2.3), this number will increase. So, in practice this is roughly a good number of residues to examine (less than 10% of our structure). Lastly, note that the residue conservation histogram is affected by the use of residue groups. However, this does *not* change the total number of voxels that satisfy the structure threshold. It only affects the distribution of these voxels within the residue conservation histogram. Consider the following example. On a particular voxelGrid run without residue groups (i.e., identity), we obtained 19 voxels that satisfy our structure threshold. Running it again with residue groups, we obtain the same 19 voxels, but the conservation histogram has shifted to the right ([0 3 6 6 4] → [0 0 6 3 10]). This makes sense since identity is most restrictive, and the use of residue groups relaxes this constraint.

2.3.2.2 Effect of random offsetting:

Random offsetting randomly positions the voxel grid on top of the set of aligned protein structures and computes the results. Since there will be variation in the number of hits found from random grid offsetting, we performed a box plot analysis with respect to the number of random runs. This tells us something about the distribution of voxel and NQ hits. For each setting of number of runs (10, 30, 50, 100), we run voxelGrid 30 times. Hence, for the 10 runs setting, voxelGrid is executed for a total of 300 times. The results are displayed in Figure 2.5.

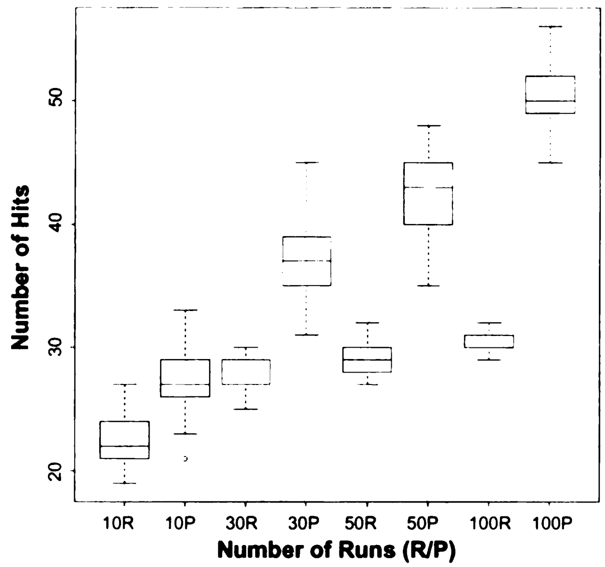


Figure 2.5: Boxplot analysis of random offsetting

In Figure 2.5, each pair of columns is the distribution of voxel and NQ hits for that number of runs (i.e., 10R and 10P are for the voxel and NQ hits from 10 runs respectively). As more of the structure space is sampled through the increased number of runs, more hits are recovered, and the number of voxel hits begins to plateau (and its deviation diminishes). The limit is the 'actual' number of spatially conserved residues in our set of alignments. The continual rise in NQ hits is a consequence of the multiple possibilities in selecting a nearest neighbor in the NQ algorithm. Depending on the location of the voxel boundaries, NQ may choose different residues as the 'closest' to the unrepresented structure.

2.3.2.3 Systematic vs. random grid offsetting:

As an alternative to sampling the alignment space using random offsets, a systematic search can be performed. Systematic offsetting is more rigorous, as it systematically and independently shifts the voxel grid in each dimension by a finite step for each voxelGrid run. We compared the two grid offsetting methods by running voxelGrid roughly the same number of times using both methods, on the same set of aligned structures and with the same voxel size. We could not perform exactly the same number of runs since systematic offsetting required an equal number of steps (see Section 2.3.2.4) in each dimension, resulting in a cubed number (i.e., $2^3=8$, $3^3=27$, etc.). Here, we have used four structures and N=2.5 as the voxel size for the comparison in Table 2.3.

Grid Offset	Num Runs	Voxel/NQ Hits
Random	10	22.4/27.2 (Avg. from column 1 in boxplot in Fig. 2.5)
Systematic	8	25/28
Random	30	27.5/37.5 (Avg. from column 3 in boxplot in Fig. 2.5)
Systematic	27	28/31
Random	50	29.2/42.4 (Avg. from column 5 in boxplot in Fig. 2.5)
Systematic	64	28/41
Random	100	30.8/50.3 (Avg. from column 7 in boxplot in Fig. 2.5)
Systematic	125	27/49
Systematic	216	29/45
Systematic	512	31/56
Systematic	1331	31/55
Systematic	2197	32/64

Table 2.3: Comparison of systematic vs. random grid offsetting

For random offsetting results, we have taken the average from the corresponding column in the boxplot presented in Figure 2.5. This gives us the average number of hits we would expect voxelGrid to find for that number of runs. For less than 125 runs, we observe that random offsetting seems to perform just as well as systematic offsetting. However, the results for random offsetting are averages, which imply that some runs did better and others worse. This is somewhat undesirable, since we really want an accurate count for the number of hits, not just an average.

For systematic offsetting, we actually do slightly worse going from 64 to 125 runs, losing one voxel hit. This is simply a consequence of the nature of sampling. For greater than 125 runs, however, the number of voxel hits returned by systematic offsetting increases monotonically. Unlike random offsetting whose result is an average, the number of hits returned by systematic offsetting is fixed and hence, preferable. However, it is useful to run voxelGrid several times using random offsetting to determine a few suitable voxel sizes (see Section 2.3.2.1 for discussion) at which to perform the systematic offsetting.

2.3.2.4 Granularity of systematic sampling:

The granularity of the systematic sampling is determined by the combination of the voxel size and the step size. The latter affects the number of times voxelGrid is run. The more runs we perform, the better our sampling of the structure space, and hence, the more complete our list of spatially conserved residues. From Table 2.3, it appears that 512 runs seems like a good number of runs to perform as it is roughly equivalent in performance to 2197 runs. To verify this, we tested an additional two datasets. We performed voxelGrid analysis using residue identity and a voxel size of N=2.5. Our results are summarized in Table 2.4, with the number of structures in each dataset appearing in parentheses. Dataset 1 is the one used in Table 2.3.

Step Size	Num Runs	Dataset 1 (4)	Dataset 2 (6)	Dataset 3 (11)
1.3	8	25/28	17/3	40/4
1.0	27	28/31	17/3	42/3
0.83	64	28/41	17/3	42/3
0.6	125	27/49	18/3	42/4
0.5	216	29/45	18/3	42/4
0.33	512	31/56	19/4	42/4
0.25	1331	31/55	19/5	43/6
0.2	2197	32/64	19/4	44/10

Table 2.4: Effect of step size on voxelGrid performance

We observed that performing 512 runs does appear sufficient. This translates to seven steps along each axis (i.e., the cube root of 512 is eight evaluations). The step size then for seven equally sized steps could be approximated as the integer part of: stepSize= voxelSize/7. We have rounded the step size to approximate values in Table 2.4. For all of the voxelGrid analyses done in the following chapters, we performed 512 runs using systematic offsetting, where an appropriate step size is computed using the preceding formula.

2.3.2.5 Parallelization:

We observed that each voxelGrid run is self-contained and does not use results from any previous voxelGrid run. This makes the algorithm easily parallelizable. Multiple voxelGrid runs could be executed simultaneously on either a single or multiple processors. In the end, the output files are simply merged together into a single result. The only caveat is that the output files being merged must be created from running voxelGrid using the same set of aligned structures and at the same voxel size. Otherwise, it would not make sense to merge the results.

2.4 Dependence of voxelGrid Algorithm on Alignment Quality

2.4.1 Considerations

The goal of voxelGrid is to identify spatially conserved residues. One of the limitations of our algorithm, and any algorithm that examines a set of structural alignments, is the quality of that set of alignments. Despite the plethora of structure alignment algorithms, there is really no way around this limitation since structure alignments are not unique. There is no one correct alignment given a set of structures (unless they are all the same structure!). At minimum, an algorithm that examines these structure alignments should be robust to small variations in the alignment quality. voxelGrid handles this through the use of voxels. However, even with voxels, serious misalignments can lead to poor results. Given a set of structures, possibilities for generating an alignment include:

1. We know some of the functional residues (e.g., active site residues) and choose to align the structures based on these. This will most likely lead to a reasonably 'correct' alignment.
2. We do not know any functional residues and perform a structural alignment using one of the many publicly available web servers (see Section 2.4.2). These alignments are not necessarily 'correct' and can be misleading. However, within in the same fold class, we expect that residues important for function are conserved. Additionally,

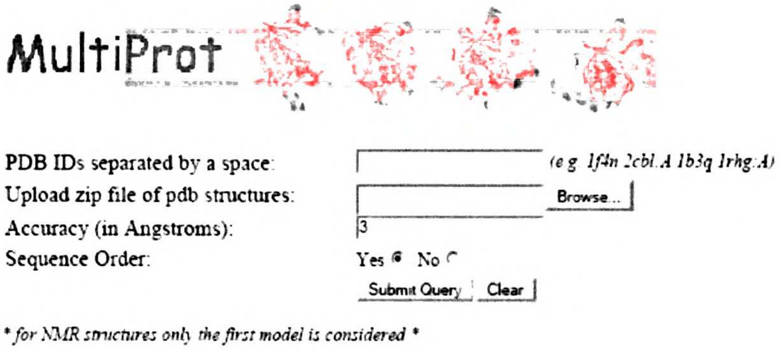
some algorithms allow for the inclusion of a multiple sequence alignment to assist the structure alignment process.

3. We know the ligand that a set of receptors bind. We can get an alignment of the receptors by aligning the ligand.

In our testing of voxelGrid on multiple datasets, we tested the first two situations, and the next section briefly discusses the structure alignment tools that we used.

2.4.2 Preparation of alignments

In this study, we performed structure alignments using either public web servers or the molecular visualization package Chimera. Structure alignment web servers are an excellent choice when no functional residues are known for the set of structures being aligned. We have compared six of these web servers as an example of using voxelGrid in Section 2.4.3. We note that most servers only perform pairwise alignments (i.e., are limited to two structures at a time). However, some servers (such as CE-MC [13] and MultiProt [14]) are able to perform multiple structure alignments. An example of the query form used by one of these servers is given in Figure 2.6.



PDB IDs separated by a space: (e.g. 1f4n 2cbl:A 1b3q 1rhg:A)
Upload zip file of pdb structures:
Accuracy (in Angstroms): 3
Sequence Order: Yes ● No ○

* *for NMR structures only the first model is considered* *

Figure 2.6: Example of structure alignment web server (MultiProt)

The structures may be specified as PDB IDs or uploaded. It is recommended that a chain be specified in addition to the PDB ID. The final alignment is returned as a single PDB file. This file must be unpacked, since voxelGrid expects that the aligned structures each reside in a separate file. The Appendix contains additional notes specific to the processing of output from specific web servers.

Chimera may be used as an alternative to using structure alignment web servers. The PDB files for the structures of interest should be first downloaded from the PDB.

If some of the functional residues for this set of structures are known:
Some online databases contain lists of conserved functional residues across a superfamily or family. One such example is the Structure Function Linkage Database (SFLD) [15]. A structure alignment can be done in Chimera using the `match` command on these residues. However, the residues must be totally conserved across all the structures. The aligned structures are then written back out as individual PDB files[6].

If no functional residues for this set of structures are known:
In the absence of known functional residues, an MSA can be used in Chimera to assist with the structure alignment. The MSA can be built using any of the numerous publicly available sequence alignment programs (such as ClustalX [16]). Alternatively, several online databases offer curated sequence alignments, including several based on structural alignments (e.g., S4 [17], CDD [18]). The structure alignment was then performed using the MSA and the Multalign Viewer (MAV) tool in Chimera (Figure 2.7). The MAV tool automatically associates a sequence in the MSA with a structure if their sequences can be aligned without too many mismatches. Residues to be used for aligning the structures can then be selected manually (shown as pink highlighted columns), or if none are selected, the full sequence is used.

---

[6] See Appendix for note specific to writing PDB files in Chimera on Windows platforms.

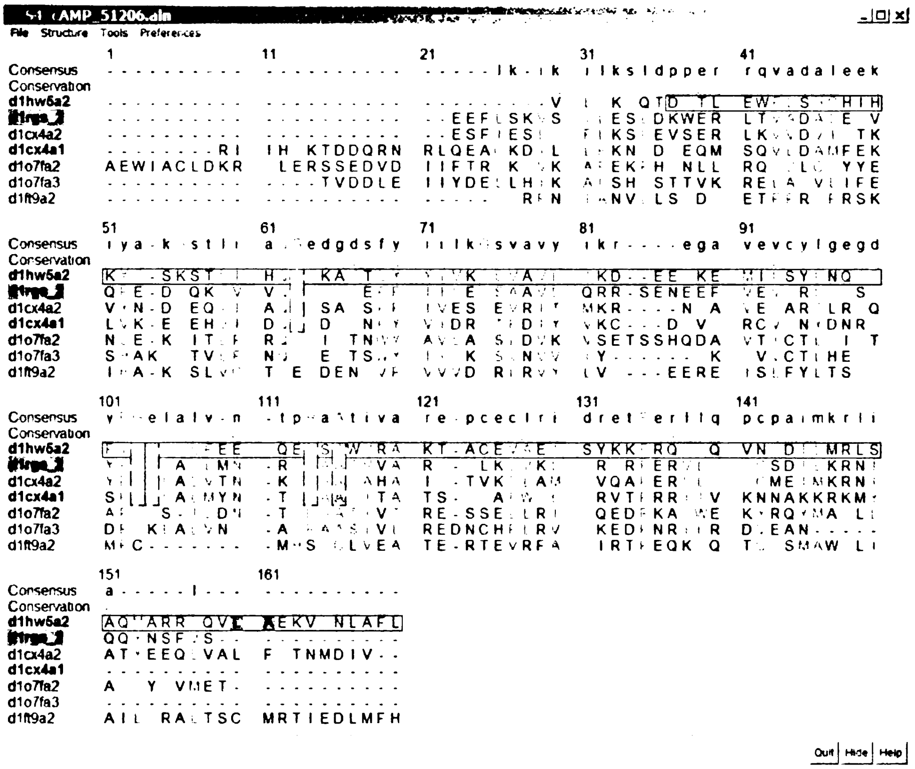


Figure 2.7: Multalign Viewer tool in Chimera

We refined this alignment through a 'bootstrapping' process where a MSA is created from the structure alignment using the Match->Align tool in Chimera. This MSA is then fed back into the MAV tool to align the structures. This back and forth process between the two tools is repeated for a few iterations. The aligned structures are then written back out as individual PDB files.

2.4.3 Example: Comparison of structure alignment web servers

Curiously, although the performance of voxelGrid is dependent on the quality of the set of structural alignments, it could also be used to measure that quality. Traditionally, the quality of structural alignments is reported as a single RMSD value, quantifying the overall alignment quality. However, RMSD is a value averaged across the entire structure and reveals nothing about the alignment of key residues. We can use voxelGrid to supplement reported RMSD values. The quality of structural alignments is correlated with the number of hits voxelGrid returns at a particular voxel size. A larger number of hits at smaller voxel sizes indicate higher quality alignments. Since we expect more residues to be found with increasing voxel size, matches made in large voxels are not as significant as those made in smaller voxels.

We considered a pairwise alignment between two diverse proteins from the enolase superfamily (2MNR and 4ENL) using six structure alignment web servers. From visual inspection, the alignments looked very similar. We downloaded each set of alignments then performed voxelGrid analysis (512 runs, residue groups, default settings). The results are summarized in Table 2.5 and plotted in Figure 2.8.

Alignment Algorithm	Voxel Hits (voxel size=1.0/1.5/2.0/2.5)
CE [19]	16 / 39 / 94 / 160
DaliLite [20]	21 / 42 / 89 / 164
Fast [21]	16 / 44 / 88 / 173
K2 [22]	15 / 46 / 91 / 160
MultiProt	19 / 56 / 92 / 160
Superpose [23]	9 / 41 / 82 / 148

Table 2.5: Using voxelGrid to compare structure alignment web servers

Superpose clearly performed the worst, but there does not appear to be a clear winner from the remaining five. Without examining exactly which residues were aligned, these five seemed to perform equally well. However, we preferred MultiProt and CE[7] since they had the added benefit of supporting multiple structure alignments.

In the subsequent chapters, we will discuss the application of voxelGrid to two datasets: cAMP binding proteins and TIM proteins. Two additional datasets (enolase superfamily and DHFR) were examined but will not be discussed. The techniques used for the analyses of those two are similar to and better illustrated by the cAMP binding and TIM datasets.

---

[7] Actually, the multiple structure version of the CE server is called CE-MC, but it uses the same algorithm.

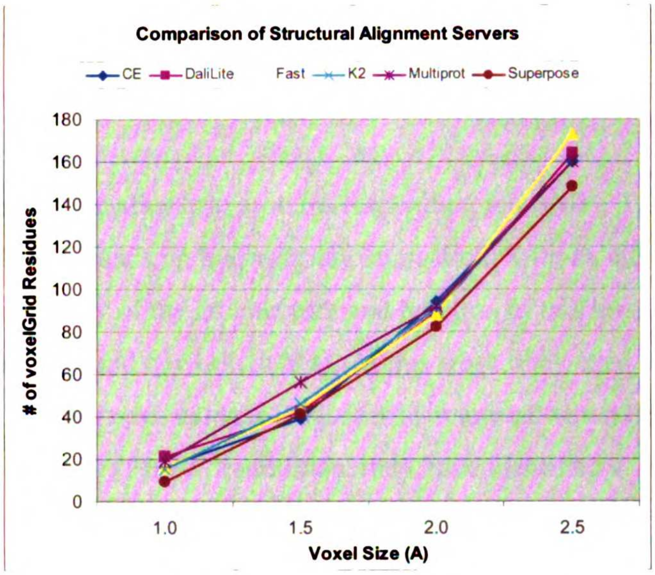


Figure 2.8: Graph of Table 2.5

Chapter 3: Example: the cAMP Binding Proteins

3.1 Introduction to cAMP Binding Proteins

cAMP binding proteins are allosteric proteins responding to cAMP. The structure change subsequent to binding depends on hydrophobic interactions with an adenine ring of cAMP. An excellent study of the architecture and function of this binding domain was recently done by Berman and coworkers [24]. In this work, they used surface matching to identify 11 highly conserved residues (Figure 3.1), many of which lie in the conserved phosphate binding cassette (PBC). In addition, they have identified a single hydrophobic residue that cannot be predicted from an MSA but is spatially conserved. Instead of lining up nicely in a column in the MSA, the residue lies scattered across the multiple sequences (colored red Figure 3.1). This residue was only identified after tedious visual inspection of a multiple *structure* alignment. To truly appreciate how difficult this would be, consider Figure 3.2. Additionally, this residue was identified as significant as it seals the hydrophobic pocket in which the adenine ring resides.

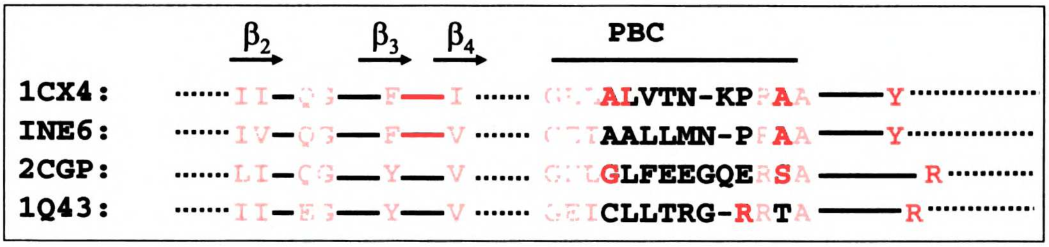


Figure 3.1: MSA of four cAMP binding proteins. The conserved hydrophobic residue and the residues identified via surface matching by Berman et al. are colored colored red and orange respectively. Figure adapted from from [24].

Motivated by the Berman study, we wanted to investigate whether voxelGrid is able to automatically identify this hydrophobic residue. More importantly, we are seeking to identify the broader class to which this hydrophobic residue belongs; residues that are spatially conserved but do not line up in an MSA. We began by first examining the same four structures used in [24] and then extending our analysis to include additional cAMP binding structures.

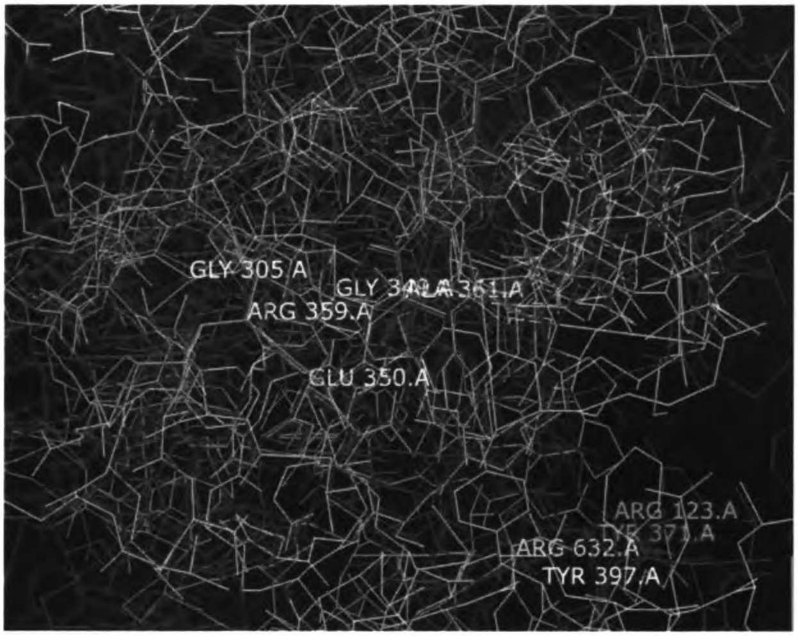


Figure 3.2: Why visual inspection of structural alignments is so difficult! An alignment of four cAMP binding structures, showing the conserved hydrophobic residue (colored red in Figure 3.1) labeled in the lower right hand corner.

3.2 Analysis of Four Representative cAMP Binding Structures

3.2.1 Preparation of dataset and results

The first step was to prepare a set of structural alignments of the four proteins. As discussed in Chapter 2, this can be done through (i) web-based structure alignment servers or directly in Chimera with either (ii) a list of known conserved residues or (iii) an MSA. We investigated each of these three options. To simplify the alignment process, we only considered chain A from each of the proteins:

Four representative structures: `1CX4:A, 1NE6:A, 2CGP:A, 1Q49:A`

1. CE-MC alignment web server:

   We used the web-based form for submission of our four proteins of interest.

2. Five conserved residues:

   The Berman study identified five totally conserved residues via surface matching (see Figure 3.1). For reference, these five residues in 1CX4 are: G305, G349, E350, R359, A361. Using Chimera, we based our structure alignment on these.

3. Bootstrapping using an MSA:

   A sequence alignment of cAMP binding proteins was obtained from the S4 database. Only three of our four structures associated[8] with sequences in this alignment (1Q49 did not). Using the MAV and the Match->Align tools in Chimera, we bootstrapped these and aligned 1Q49 manually using the five residues above. Three iterations were performed. Parameters for the MAV and the Match->Align tools for pruning and residue-residue cutoff were 3.0 A and 5.0 A respectively. The alpha carbon trace of this alignment is shown in Figure 3.3. Each structure is represented by a different color, and selected key residues are labeled.

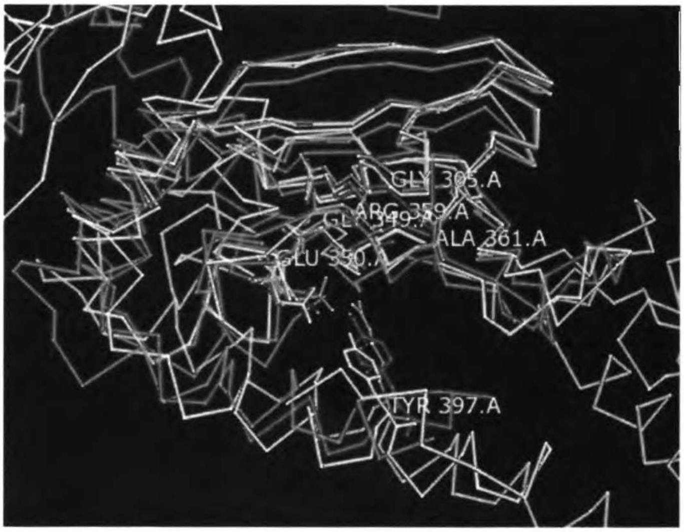


Figure 3.3: Alignment of cAMP structures via bootstrapping

---

[8] Recall from Section 2.4.2 that the MAV tool in Chimera automatically associates each structure with a sequence in the MSA if possible.

We performed the voxelGrid analysis using residue groups, default settings[9], and systematic offsetting at varying voxel sizes. The results are summarized in Table 3.1. We observe that alignments done using either the five conserved residues or bootstrapping performed comparably. However, the CE-MC alignment is of poorer quality[10].

Voxel Size	Step Size	CE-MC	Five Residues	Bootstrap
N=2.00	0.26	3/28	22/20	24/23
N=2.25	0.30	4/34	25/25	27/35
N=2.50	0.33	8/39	31/33	31/56
N=2.75	0.375	12/45	33/71	35/67
N=3.00	0.40	17/71	45/97	41/83

Table 3.1: Comparison of alignment methods for 4 representative cAMP binding structures

3.2.2 Analysis

We searched the output from each of the three methods and did not find among the voxel hits the hydrophobic residue identified by Berman and coworkers. However, observe that TYR397 is too far from the corresponding residue in the other three structures (Figure 3.4). Instead, it was found as a hit in all three methods if we consider the adjoining voxel containing TYR397 (i.e., via NQ). Surprisingly, the residue was found at N=2.0 and onwards for CE-MC, compared to N=2.25 and onwards for the other two methods.

---

[9] Recall from Section 2.3.1.1 that default settings imply: whole-residue centroids, NQ enabled, and structure and residue thresholds set equal to the number of aligned structures.

[10] Recall from Section 2.4.3 that a larger number of hits at smaller voxel sizes indicates higher quality alignments.

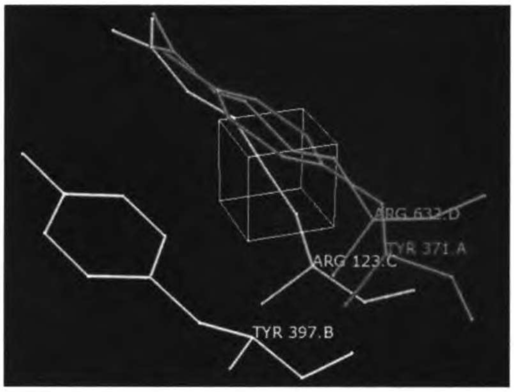


Figure 3.4: Hydrophobic residue of interest (CE-MC alignment with N=2.0)

In addition to this conserved hydrophobic residue, voxelGrid identified other residues as being spatially conserved. We screened this list for all the conserved residues identified via surface matching in the Berman study. We were able to recover all but one of these residues (directly below the β3 strand label at the top of Figure 3.1). This demonstrates that voxelGrid can be used to identify spatially conserved residues. Furthermore, it is able to do so even in instances where the overall structural alignment is of low quality. This is particularly encouraging as it confirms that our method is not tied to any one structure alignment algorithm.

3.3 Analysis of Broad Set of cAMP Structures

We extended the above analysis on four representative structures to include additional cAMP binding structures identified in [24]. This combined set consisted of 20 proteins. We excluded four proteins that did not associate with the same S4 alignment used above, leaving us with 16 proteins, partitioned into three groups:

CAP group: 2CGP:A, 1CGP:A, 1J59:B, 1RUN:B, 1RUO:B, 1LB2:A, 1G6N:B, 1HW5:A, 1I5Z:B, 1I6X:A
PKA group: 1CX4:A, 1NE6:A, 1NE4:A, 1RGS:_
HCN group: 1Q43:A, 1Q50:A

We investigated how similar these structures were both (i) within the same and (ii) across different groups.

3.3.1 Preparation of dataset

We prepared the 16 structures using the same procedure outlined in Section 3.2.1. The only difference was the use of the CE-MC alignment web server on each of the three groups separately.

3.3.2 Results and analysis

How similar are structures within the same group?

We used the CE-MC alignment web server on each of the three aforementioned groups. Visual inspection of the alignments within each group indicated that the structures were quite similar. We performed voxelGrid analysis at varying voxel sizes, using default settings and *identity* (to avoid pulling back too many hits). The number of structures in each group is given in parentheses after the group name in Table 3.2. For the HCN group, the two structures aligned so well (pulling back ~89% of the protein), based on results at N=1.5, that results using larger voxel sizes would be meaningless (hence skipped).

Voxel Size	Step Size	CAP (10)	PKA (4)	HCN (2)
N=1.50	0.20	61/19	21/46	168/11
N=1.75	0.24	80/20	26/55	-
N=2.00	0.26	97/19	36/55	-

Table 3.2: Comparison of cAMP structures within groups

On the whole, the structures within each group are very similar, given that we are still pulling back this many hits using identity and small voxel sizes. The PKA group was the most diverse, while the HCN group was the most similar. Since the structures aligned so well within each group, it was not necessary for us to use the other two alignment methods. The similarity within groups also explains why Berman et al. were able to choose one member each from CAP and HCN and two members from PKA as representative structures.

How similar are structures across different groups?

We approached this in the same manner as we did previously for the four representative structures.

1. CE-MC alignment web server:

   The CE-MC server limited our submission to 15 structures, so 1Q50 was arbitrarily excluded.

2. Five conserved residues:

   In order to manually align the set of structures in Chimera, we used the five conserved residues to add the remaining 12 structures to the existing alignment of four representative structures. The resulting alignment is shown in Figure 3.5. For simplicity, only the five conserved residues used in alignment and the conserved hydrophobic residue are shown. The alignment quality appears to be good.

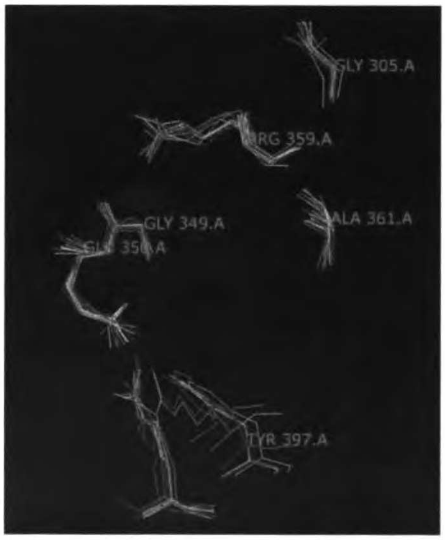


Figure 3.5: Five conserved residues and conserved hydrophobic residue

3. Bootstrapping using an MSA:

   We used the same S4 alignment as before. Since 1Q49 did not associate previously, neither did 1Q50 which is from the same group (HCN). Using the MAV and the Match->Align tools in Chimera, we bootstrapped the remaining 14 structures that did associate and aligned the remaining two structures manually using the five residues determined above. Three iterations were performed. Parameters for the MAV and the Match->Align tools for pruning and residue-residue cutoff were 3.0 A and 4.0 A respectively.

We performed the voxelGrid analysis using residue groups, default settings, and systematic offsetting at varying voxel sizes. The results are summarized in Table 3.3. We observe that bootstrapping gave a slightly better alignment than using the five conserved residues. Not surprisingly, CE-MC once again gave the poorest alignment, but performed better than before. The improved performance is probably a result of the increase in information provided by the addition of structures.

Voxel Size	Step Size	CE-MC	Five Residues	Bootstrap
N=2.50	0.33	17/14	18/7	23/8
N=2.75	0.375	21/21	26/9	27/14
N=3.00	0.40	28/29	35/8	38/20

Table 3.3: Comparison of alignment methods for a broad set of cAMP binding structures

Unlike previously, we are not looking for the conserved hydrophobic residue identified by Berman and coworkers. In fact, voxelGrid will not be able to identify it in this set of alignments. The reason is that NQ only works when the voxel is missing exactly one structure. From Figure 3.5, we observe that the hydrophobic residue can be in one of two locations depending on whether it is an Arg or a Tyr in the respective structure.

3.4 Measuring Information Redundancy

We already know that the structures within each group are very similar to one another, which implies that there is information redundancy. To test this hypothesis, we performed leave-one-out testing on our set of 16 structures by leaving out each of the four representative structures, one at a time. If there is information redundancy within each group, then deletion of one structure should have negligible effect. We performed voxelGrid analysis with a fixed voxel size of N=2.5, using both residue groups and default settings. The results are summarized in Table 3.4.

Voxel Size	# Structures	Alignment	# of Runs	Voxel/NQ
N=2.5	15 (no 1Q50)	CE-MC	1331 (Step: 0.25)	18/16
N=2.5	15 (no 1CX4)	Bootstrap	1331 (Step: 0.25)	27/7
N=2.5	15 (no 1NE6)	Bootstrap	1331 (Step: 0.25)	25/10
N=2.5	15 (no 2CGP)	Bootstrap	1331 (Step: 0.25)	25/7
N=2.5	15 (no 1Q50)	Bootstrap	1331 (Step: 0.25)	25/21
N=2.5	16	Bootstrap	1331 (Step: 0.25)	25/7

Table 3.4: Measuring information redundancy in the broad set using leave-one-out testing

As expected, the deletion of any one of the representative structures has almost no effect on the number of voxel hits when compared with all 16 structures. Deleting 1NE6, 2CGP, or 1Q43 has no effect, whereas deleting 1CX4 results in a small increase. However, the structures within each group are not identical and *do* contribute some unique information. This is illustrated by the case where we only have the four representative structures (i.e., Table 3.1, N=2.50, 31 hits). Hence, moving from 16 structures down the four representative structures increases the number of voxel hits from 25 to 31 (i.e. we are comparing the last row of Table 3.4 with the middle row of Table 3.1).

If we perform leave-one-out testing with only the four representative structures, removal of any one results in a significant increase in the number of hits (Table 3.5). Again, this is consistent with our observation that these four structures are significantly different from one another. The inclusion of more structures makes the identification of spatially conserved residues more robust, but at the cost of increased noise. The use of fewer but representative structures decreases the noise yielding better signal.

Voxel Size	Step Size/#runs	Voxel/NQ Hits (3)	Voxel/NQ Hits (4)
N=2.5 (no 1cx4.pdb)	0.33/512	40/81	31/56
N=2.5 (no 1ne6.pdb)	0.33/512	37/100	31/56
N=2.5 (no 2cgp.pdb)	0.33/512	52/96	31/56
N=2.5 (no 1q43.pdb)	0.33/512	51/90	31/56

Table 3.5: Measuring information redundancy in the representative set using leave-one-out testing. The number of structures is given in parentheses.

In this chapter, we have demonstrated using voxelGrid on a set of cAMP binding proteins. We have illustrated its usefulness in conjunction with leave-one-out testing to characterize information redundancy in a set of aligned structures. More importantly, even when the set of structures was not well aligned, the algorithm was still able to identify spatially conserved residues. This included the conserved hydrophobic residue and the residues identified via surface matching by Berman et al.. However, we do not know if the other residues found in addition to these are functionally significant. In the following chapter, we examine this question in a set of triosephosphate isomerase proteins by comparing voxelGrid's results against other computational predictions and experimental results.

Chapter 4: Example: the TIM Proteins

4.1 Introduction to TIM Proteins

The triosephosphate isomerase (TIM) proteins represent the canonical example of $(\beta/\alpha)_8$ barrel architecture, the most common fold among protein catalysts. The barrel architecture is composed of eight repeated units of a single strand-loop-helix-turn motif (Figure 4.1). The strands form the core of the barrel with helices surrounding it. The active sites of all known $(\beta/\alpha)_8$ barrel enzymes have been identified in the $\beta \rightarrow \alpha$ loops [25]. There is also a loop which operates like a flap that moves upon ligand binding. This leads to an open and a closed conformation for TIM crystal structures.

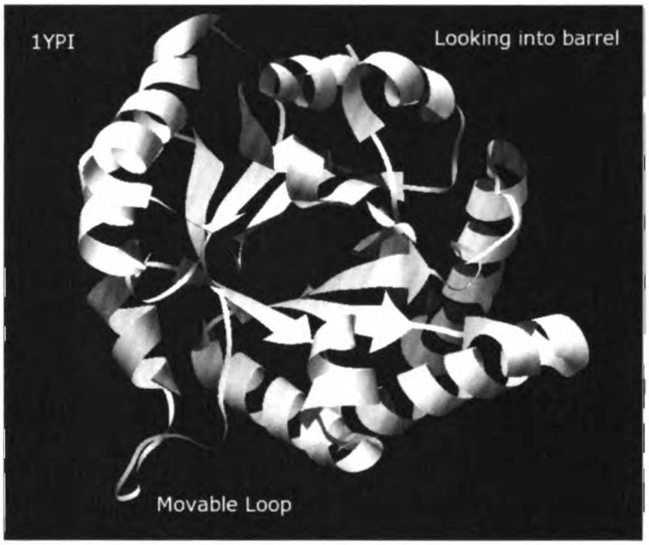


Figure 4.1: Barrel architecture of TIM protein (1YPI from yeast)

In a recent study, Silverman and coworkers [26] performed mutagenesis experiments on nearly half of the residues in the yeast TIM protein in an attempt to elucidate all functional residues. They concluded that mutations in either the central core of the β-barrel or the β-strand stop motifs would significantly reduce catalytic activity and provided a list of these residues. Motivated by their study, we performed voxelGrid analysis on a diverse set of TIM proteins. We first analyzed the TIM structures in the open and closed conformations separately. Later we combined the two sets together in the same alignment for analyses. The goal was to compare our list of spatially conserved residues against other computational predictions and experimental results. For the former, we used the Conservation of Conservation (CoC) database [27], and for the latter, the results from Silverman et al..

4.2 The Open and Closed Conformations

As discussed above, the TIM proteins adopt either the open or the closed conformation depending on the presence of a bound ligand. We were not sure how significantly the conformation change would affect the overall structure. If it were significant, voxelGrid would find few if any spatially conserved residues using the combination of open and closed structures. Hence, we decided to consider each conformation separately.

4.2.1 Preparation of dataset and results

We began by examining the 43 SWISSPROT [28] sequences (i.e., species) used in the study by Silverman et al.. We only identified 18 unique structures, since many of the SWISSPROT sequences do not have an associated structure. Two of these structures, 1YPI and 7TIM, are both yeast TIM proteins[11] and exhibit the open and closed conformations, respectively. We assigned each of the remaining 16 structures to either the open or the closed conformation group by visual inspection of the movable loop region in its alignment with 1YPI and 7TIM. This gave us the following groups:

Closed (11 structures):

`1AMK:_, 1AW1:A, 1B9B:A, 1BTM:A, 1LYX:A, 1MO0:A, 1NEY:A, 1TCD:A, 1TPH:1`
`2BTM:A, 7TIM:A`

---

[11] Both 1YPI and 7TIM are 249 amino acids in length.

Open (7 structures):

`1AW2:A, 1HTI:A, 1TRE:A, 1YDV:A, 1YPI:A, 3TIM:A, 6TIM:A`

We then used the CE-MC alignment web server to align the structures within each of these groups. The alignments were then downloaded then run through voxelGrid using systematic offsetting with default settings. The results are presented in Table 4.1, and a ribbon representation of the alignments is presented Figure 4.2.

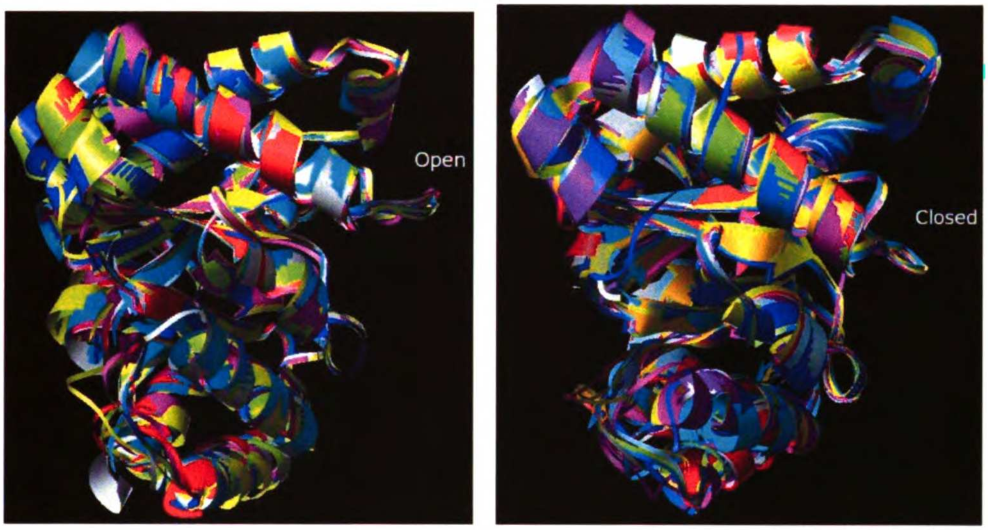


Figure 4.2: Alignment of TIM structures (each a different color) and viewed from the side. Both conformations are shown (left: open, right: closed). For reference, the conformation labels are located just beside the movable loop.

Voxel Size	Residues	Step Size	Open Voxel/NQ	Closed Voxel/NQ
N=2.0	Identity	0.26	50/4	41/4
N=2.5	Identity	0.33	53/3	42/4
N=2.0	Group	0.26	100/29	88/21
N=2.5	Group	0.33	117/40	99/24

Table 4.1: Comparison of TIM structures within open and closed conformations

4.2.2 Analysis

We observed that the alignment quality is actually quite good, even when we used identity for residue matching. The number of hits is comparable at N=2.0 and N=2.5 voxel sizes, and roughly doubles when we used residue groups. Indeed, the structures within each group are quite similar. We have mapped the spatially conserved residues found by voxelGrid onto the two yeast TIM proteins, 1YPI and 7TIM (Figure 4.3). Again, for reference, 1YPI and 7TIM belong to the open and closed conformation groups respectively.

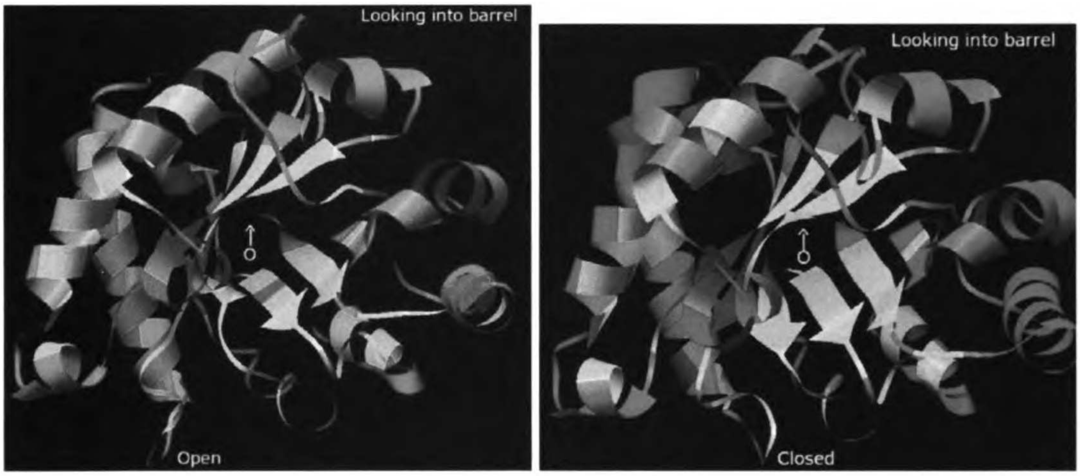


Figure 4.3: Conserved residues found by voxelGrid mapped onto TIM structures (left: 1YPI, right: 7TIM). Residues found at voxel size N=2.0 appear in yellow while additional ones found at N=2.5 appear in green.

The highlighted regions indicate the spatially conserved residues identified by voxelGrid (using residue groups). We observed that the highlighted sets are very similar, including a number of the turn regions (i.e., β strand stop motifs) and the β barrel core. This is consistent with the findings of Silverman et al. (discussed in more detail in Section 4.5). In the next section, we will combine the open and closed groups and look for spatially conserved residues across the entire set of proteins. However, notice that this will automatically exclude any movable residues such as those in the movable loop region.

4.3 Analysis of a Diverse Set of TIM Proteins

4.3.1 Preparation of dataset

Unlike previously in Section 4.2.1, we will forego the use of structure alignment web servers, and instead perform our alignment in Chimera with the assistance of an MSA. There are two reasons for this. First, the web servers usually limit the number of structures that can be aligned concurrently. Second, we wanted to demonstrate not only how a sequence alignment can be used to supplement a structural alignment but how the two are fundamentally different, even though both are representations of similarity across multiple proteins.

We used a S4 alignment for bootstrapping of the cAMP dataset. S4 alignments include information from structural alignments. It is useful to consider the more common scenario: where we have no prior knowledge from other sources about our structures. We want to see if voxelGrid is able to identify interesting spatially conserved residues, despite starting from a structurally uninformed MSA. In this case, we constructed an MSA of the 18 structures by loading their sequences into ClustalX and aligning them using default parameters. The result is shown in Figure 4.4.

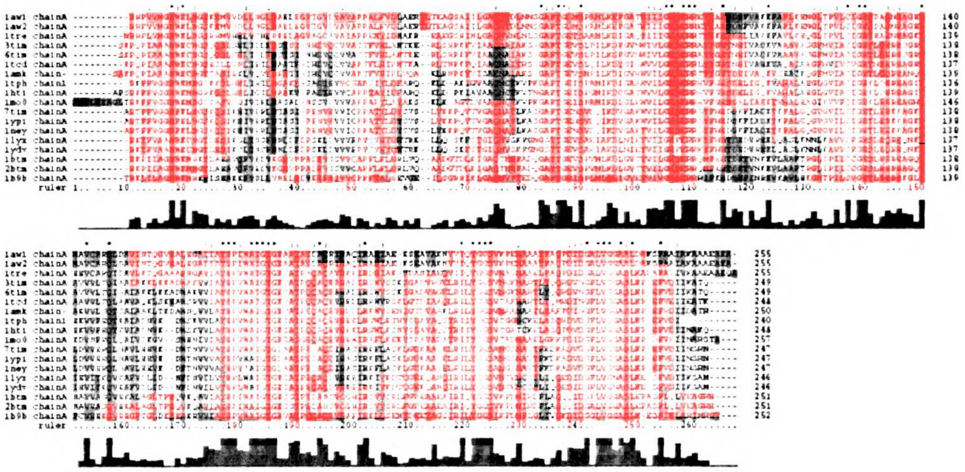


Figure 4.4: Multiple sequence alignment of TIM sequences using ClustalX

From the multitude of solid-colored bands and asterisks[12] above the columns, we observe that this is a good (though not necessarily correct) alignment. We will use this MSA as the starting point for bootstrapping our TIM structures in Chimera.

We loaded this MSA into the MAV tool in Chimera and bootstrapped for three iterations. Parameters for the MAV and the Match->Align tools for pruning and residue-residue cutoff were 3.0 A and 1.5 A respectively. After this procedure, the aligned structures had pairwise RMSD values below 0.63 and were written out into separate files.

4.3.2 Results

We ran voxelGrid using systematic offsetting with default settings on the aligned structures. We first tried two voxel sizes using either identity or residue groups. The results are presented in Table 4.2. Again, we observed that using residue groups pulls back roughly twice the number of hits when compared to identity. In addition, roughly 60-75% of the residues identified earlier in Table 4.1 are found here. This implies that the open and closed groups are quite similar: only a small fraction of residues is lost with the addition of another group. The conformational change resulting from ligand binding is probably localized to the movable loop region.

Voxel Size	Residues	Step Size	Voxel/NQ Hits
N=2.0	Identity	0.26	32/1
N=2.5	Identity	0.33	37/8
N=2.0	Group	0.26	67/13
N=2.5	Group	0.33	85/27

Table 4.2: Results for combined set of TIM structures (open and closed conformations)

To get a sense of which residues are the most spatially conserved, we reduced the voxel size and reran voxelGrid using default settings and residue groups. The results are presented in Table 4.3 and shown on structures in Figure 4.5.

---

[12] ClustalX's notation for indicating residue conservation in that column.

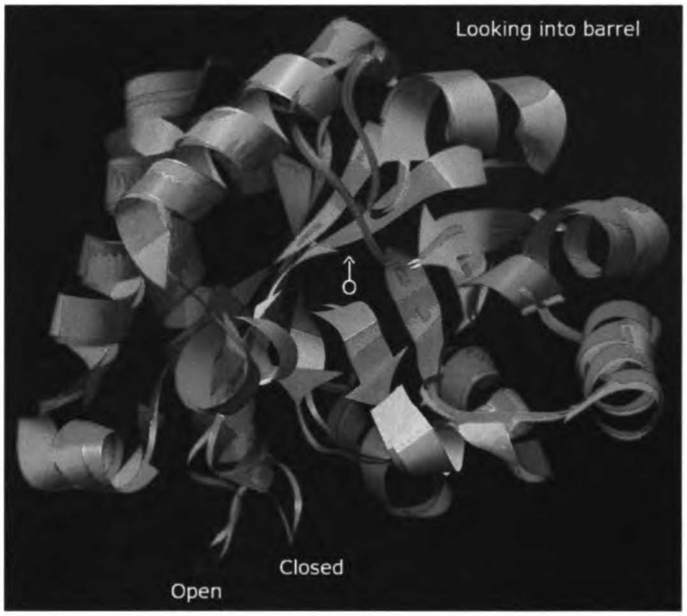


Figure 4.5: Conserved residues found by voxelGrid mapped onto representative TIM structures (red: 1YPI, blue: 7TIM). Residues found at voxel size N=1.0 appear in yellow while additional ones found at N=1.5 appear in green.

Voxel Size	Step Size	Voxel/NQ Hits
N=1.0	0.14	16/18
N=1.5	0.20	49/13
N=2.0	0.26	67/13
N=2.5	0.33	85/27

Table 4.3: Results for combined set of TIM structures (using residue groups)

4.3.3 Analysis: Comparison of sequence and structure alignments

In an MSA, the conserved residues appear in columns, while in a multiple structure alignment, they are the residues found by voxelGrid. We can map these voxelGrid residues onto the MSA for a combined view of residue conservation in both sequence and structure space (Figure 4.6). Residues identified by voxelGrid have a green border around them. Residues conserved in the MSA are shown in capital letters on the

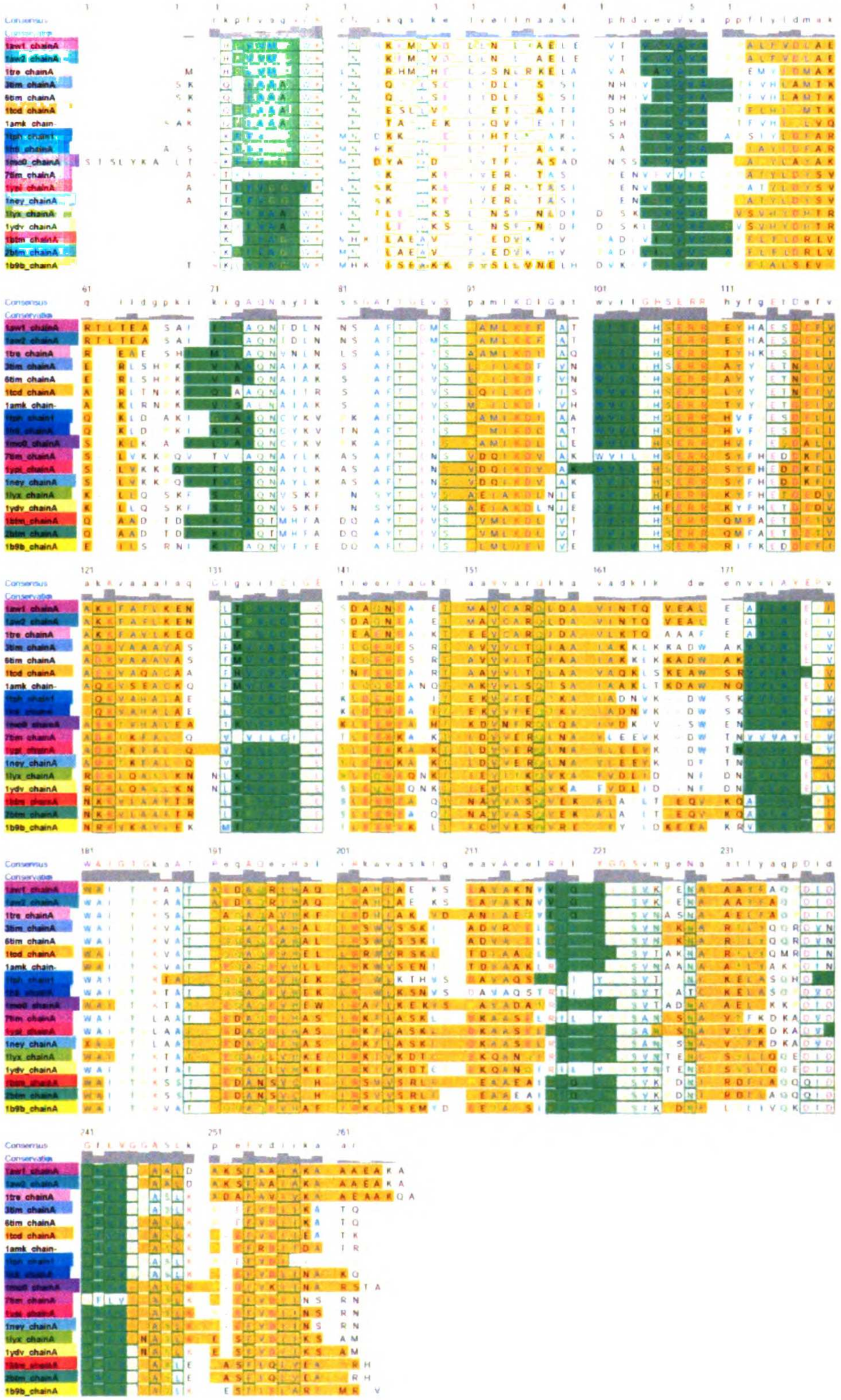


Figure 4.6: Combined view of residues conserved in sequence and structure

consensus line beneath the residue numbers. The green and gold highlighting indicates the location of secondary structure elements (strand and helix respectively).

From Figure 4.6, we can ask three key questions:

1. *Are there spatially conserved residues that are not conserved in the MSA?*
   Yes. We observe that there are indeed multiple residues found by voxelGrid but not identified as conserved in the MSA (i.e., lower case letters in the consensus line). Note that voxelGrid residues are spatially conserved but need not be identical (however, they must belong to the same residue group). The ones that are not identical will obviously not be conserved in sequence. More interesting are residues that are spatially conserved but do not align in the same column in the MSA (e.g., the case with the conserved hydrophobic residue in cAMP binding proteins). However, we do not find any such examples here.

2. *Do all highly conserved sequence locations have voxelGrid identified residues?*
   No. This is expected since sequence conservation does not imply structural conservation. This is compounded by the fact that we have two conformations (open and closed). In the $(\beta/\alpha)_8$ barrel architecture, the region in unit 6 between the β sheet and α helix is well conserved in sequence space but not identified by voxelGrid. This is because that sequence represents the movable loop region of the TIM proteins. Hence, this is one method of identifying movable regions: short stretches conserved in sequence but not in structure.

   We observed that residues 83-90 in the MSA represent another stretch of highly conserved residues that is not well covered by voxelGrid. We suspected that this may also be a movable region. The residues mapped into a turn region on the opposite side from our previous movable loop. However, closer examination in Chimera showed that it is a consequence of alignment drift and not an actual movable loop.

3. *Is there a secondary structure bias for spatially conserved residues?*

   We counted up the number of voxelGrid residues that were in either strands or helices (colored green and gold respectively in Figure 4.6). We used 1YPI as the reference structure and counted up the number of residues in each type of secondary structure. We found a total of 46 residues in the strands and 133 residues in the helices. The results are summarized in Table 4.4.

2nd Structure (total)	N=1.0	N=1.5	N=2.5
Helix (133)	5 (3.8%)	13 (9.8%)	28 (21.0%)
Sheet (46)	5 (10.9%)	16 (34.8%)	25 (54.4%)
Neither (68)	6 (8.8%)	20 (29.4%)	29 (42.6%)
Total (1YPI: 247)	16 (6.5%)	49 (19.8%)	82 (33.2%)

Table 4.4: Distribution of voxelGrid residues by secondary structure

We observed that the voxelGrid residues tend to lie in the strands, implying that the strand regions are better aligned. Geometrically, this makes sense since the core of a rigid structure is less prone to movement than the periphery.

4.4 Comparison of voxelGrid with CoC Predictions

The CoC database presents the conservation of residue positions in folds across protein families, and residues with high CoC are universally conserved in every family of homologous proteins that acquire a particular fold [27]. The CoC is a computational prediction, and may be used to identify putative functional residues. We were interested in comparing the predictions of CoC with voxelGrid.

When we queried CoC with 1YPI (yeast), the database returned 1AMK (*Leishmania mexicana*) as the best hit. The CoC results on 1AMK are reported based on two free parameters: p-value and entropy cutoffs. We have chosen to use the '6 amino acid types' setting, which roughly corresponds to residue groups in voxelGrid. Plots for these two parameters with respect to residue number in 1AMK are shown in Figure 4.7.

The entropy plot is not very informative, as it simply shows an oscillatory waveform with eight peaks (corresponding to the eight repeated units of the TIM barrel). The p-value plot is somewhat more informative. Using the least stringent settings (entropy: 0.9, p-value: 0.001) in the CoC database, we obtained 11 residues (colored yellow in Figure 4.8). Interestingly, the active sites of all known $(\beta/\alpha)_8$ barrel enzymes are located in the $\beta \rightarrow \alpha$ loops[25], but these may not be the same residues as those found by CoC.

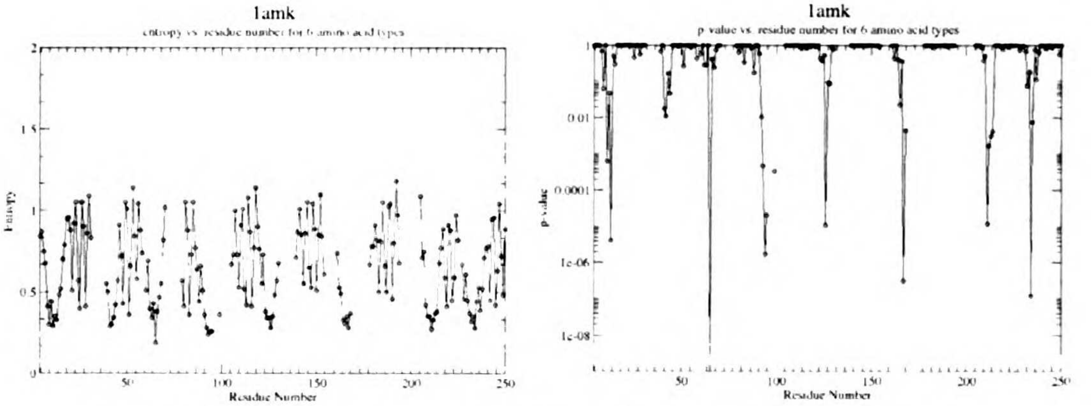


Figure 4.7: Entropy (left) and p-value (right) plots for 1AMK from the CoC database

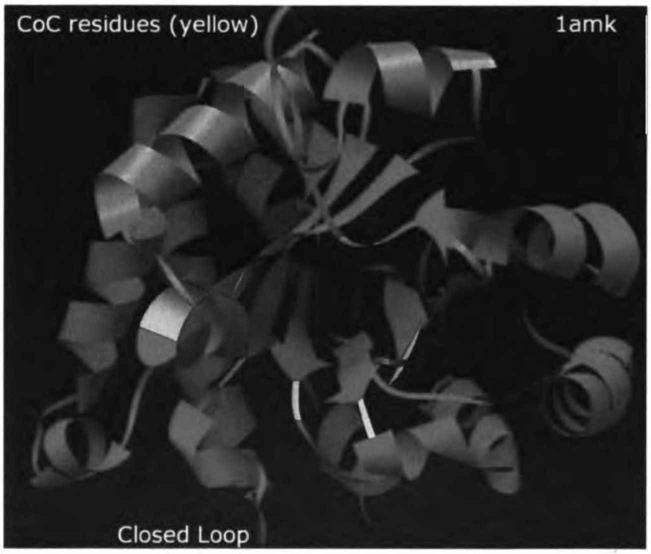


Figure 4.8: CoC residues for 1AMK

Since we already have an alignment between 1AMK and 1YPI (from Section 4.3), we can map the CoC residue numbers listed for 1AMK onto 1YPI, which allows us to compare voxelGrid and CoC results:

1AMK: N11, K13, E65, L93, G94, H95, R99, C126, E167, G211, G234
1YPI: N10, K12, Q64, L93, G94, H95, R99, C126, **E165,** G209, G232

The 11 CoC residues from 1AMK are shown on the top line, while their corresponding residues in 1YPI are below. We have colored the residues based on the resolution at which we were able to pull back that residue in our voxelGrid analysis (Section 4.3): red (N=1.0), green (N=1.5), blue (N=2.0), black (not found). Curiously E165 was not found by voxelGrid. One possible explanation may be that the residue has moved, since 1YPI is open conformation while 1AMK is closed conformation.

4.5 Comparison of voxelGrid with Mutagenesis Results

4.5.1 Large scale mutagenesis of TIM residues

In their mutagenesis study, Silverman et al. [26] performed 109 unique mutations on the yeast TIM protein and measured the resulting change in catalytic activity. The residues that were mutated were chosen based on an examination of the MSA of 43 unique TIM sequences from a wide range of species. The results are summarized in Table 4.5. Each mutation is assigned to one of three categories based on the resulting decrease in apparent k_{cat}/K_M relative to wild type. This table effectively identifies which residues in the TIM protein are functionally significant. However, it should be noted that a few residues appear in both 'Not significant' and 'Intermediate' categories depending on the substitution made. Furthermore, the range of values in the 'Intermediate' category is quite broad.

Not signifkant'			Intermediate						Deficient'	
R3A	'W90F	Y164L	I124V	0.7	G87A	3.2	N213A/N216A	130	G228V	13300
T4A	V91L	Y164V	N35A	1.0	A201L	3.8	G210A	150	G209V	15000
V7L	L93V	P166A	A200L	1.0	H185I	3.8	G232A	180	D227L	22000
F11V	I109L	T177A	F240L	1.1	L230V	4.5	K107A	210	R189M	>22000
S16A	K112Q	P178A	V24L	1.1	F6V	5.8	D225A	210	R189M/D227L	>22000
I20L	T113L	A181L	I243L	1.2	Q182A	6.3	G9V	230		
E37Q	L125V	D183A	D106A	1.2	G8V	6.3	F220L	250		
V39L	I127V	I184L	I92V	1.3	V36L	6.4	'W90V	300		
I40L	T139V	I188L	I244L	1.3	C126V	13	G128A	340		
I40V	V142A	F191L	L207V	1.4	A116L	14	Y208F	1800		
Y46F	V143L	I206L	C41V	1.6	G94A	20	G62V	2800		
Y46L	Q146L	I206V	L204A	1.7	R205M	28	N10A	3500		
S50L	V150L	Y208V	I23L	1.8	V123L	39	A110L	4100		
V51A	V154L	V226L	I127L	2.3	F5V	39	G122V	4900		
V54L	'W157F	F229L	D105A	2.3	V38L	39				
V61L	V160L	V231I	G87L	2.4	A217L	67				
A63V	V161L	G233A	S79A	2.6	Q58V/T60V	76				
T75V	V162L	S246L	G120A	2.7	F229V	76				
V80L	A163V		D85N	2.7						
I83L	Y164F		G120L	2.8						

Table 4.5: Results for mutagenesis of TIM residues (adapted from Silverman et al. [26])

The residue counts for the three groups listed in Table 4.5 are:

Not Significant: 58 (53 unique) Intermediate: 54 (52 unique) Deficient: 4 (4 unique)

In Figure 4.9, we mapped the point mutations of Table 4.5 onto the structures of 1YPI (open) and 7TIM (closed). We observed that most of the mutations performed were in the hydrophobic core, and that the majority of these had an effect (green and yellow in the figure). This is probably what led Silverman and coworkers to conclude that '[i]n contrast to the α/β interface, residues in the central core of the β barrel are extremely sensitive to substitution" [26]. However, we also observed that there are 'Intermediates' and one 'Deficient' residue outside of the core. This leads us to question how significant residues in the core are compared to those not in the core. We will examine this question in more detail in Section 4.5.3.

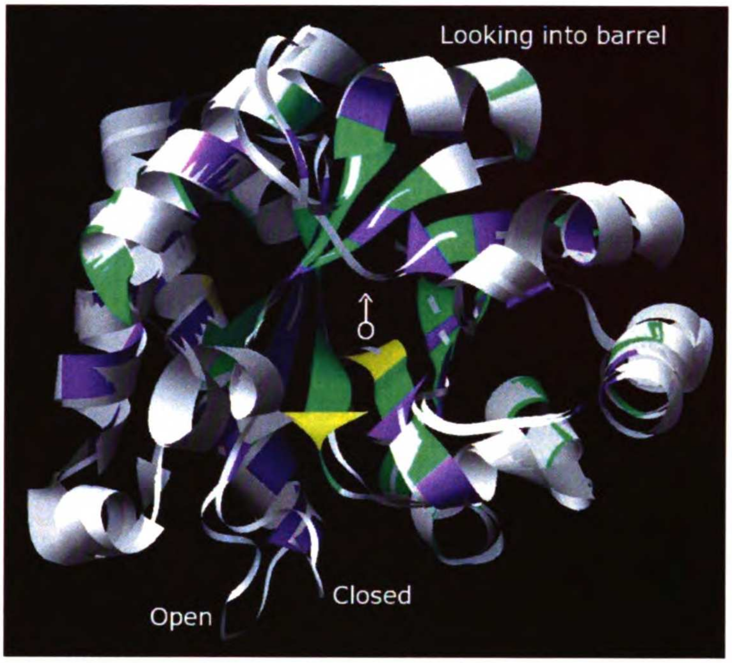


Figure 4.9: Location of residues in mutagenesis experiment. Color coding based on residue membership in Table 4.5: yellow (Deficient), green (Intermediate), purple (Not significant). Residues not colored were not tested (i.e., not included in Table 4.5).

4.5.2 Comparing mutagenesis results with CoC and voxelGrid

4.5.2.1 Mutagenesis results and voxelGrid

voxelGrid can be used to identify promising sites for mutagenesis. We wanted to know if these residues were more likely to be functionally significant than those Silverman et al. selected. For each voxel size, we classified the voxelGrid hits into one of the three categories of Table 4.5 (or not tested). The result is presented in Table 4.6. A few residues were counted twice, as they appeared both under both the 'Not significant' and 'Intermediate' categories in Table 4.5. The number of unique hits is the number in parentheses. By N=1.5, voxelGrid is able to find all four 'Deficient' residues. This is promising, but voxelGrid also finds many residues not listed in Table 4.5. Ideally, we would like to know if these have an effect when mutated.

Group (# Residues)	N=1.0	N=1.5	N=2.0	N=2.5
Deficient (4)	2	4	4	4
Intermediate (52)	4	15	21	26
Not Significant (53)	4	17	23	32
Not in Table 4.5	6	15	20	22
Total (109)	16	51(49)	68(66)	84(82)

Table 4.6: Breakdown of voxelGrid residues based on category in mutagenesis results

4.5.2.2 Mutagenesis results and the CoC:

We have colored the 11 CoC residues based on which group they belong to in Table 4.5: red (Deficient), green (Intermediate), blue (Not significant), black (not tested). The results are not particularly significant, as CoC only managed to identify one 'Deficient' residue.

1AMK: N11, K13, E65, L93, G94, H95, R99, C126, E167, G211, G234
1YPI: N10, **K12, Q64,** L93, G94, **H95, R99,** C126, **E165,** G209, G232

Compared to voxelGrid, CoC made far fewer predictions, but likewise identified many residues not listed in Table 4.5.

4.5.3 Significance of residues inside and outside the hydrophobic core

One of the chief conclusions reached by Silverman and coworkers was that residues in the central core of the β barrel are extremely sensitive to substitution. We investigated whether this conclusion could also be extended to non-core residues by removing the core (i.e., the strands) from the structure (Figure 4.10). This allowed us to see the non-core residues whose mutations had a significant effect on enzymatic activity. To our surprise, there are quite a few (colored red or green in Figure 4.10), including one 'Deficient' residue (Arg 189).

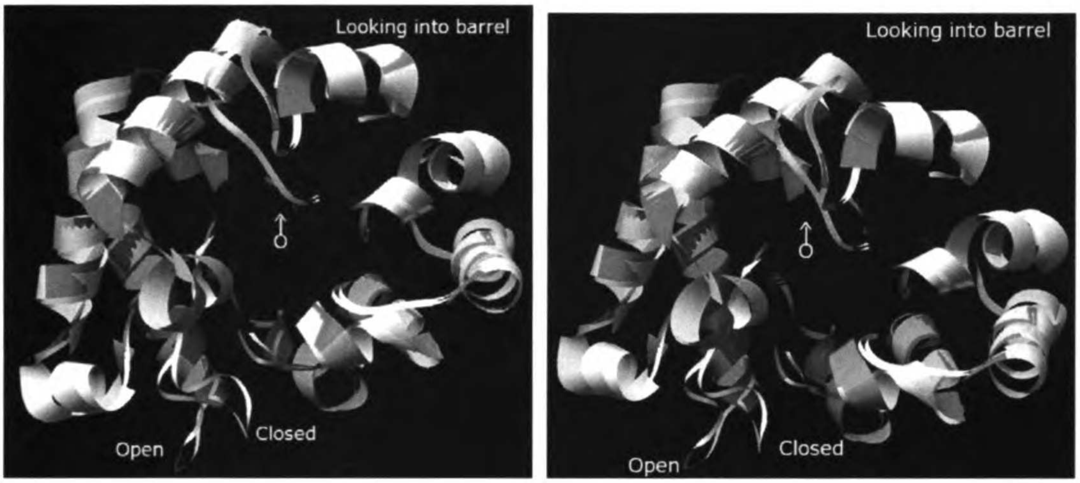


Figure 4.10: Left: yeast TIM proteins (1YPI and 7TIM) with hydrophobic core removed. Right: same image with residues having values greater than 10 in Table 4.5 colored in red.

This leads us to consider what fraction of voxelGrid residues belonged to the core compared the non-core and what fraction of each were found to be significant. The results are summarized in Table 4.7. The number of core versus non-core residues for a cell is separated by a '/'. The total number of residues for each cell is given in parentheses.

Group	Silverman et al. core/non (total)	N=1.0	N=1.5	N=2.0	N=2.5
Deficient	3/1 (4)	2/0 (2)	3/1 (4)	3/1 (4)	3/1 (4)
Intermediate	22/30 (52)	2/2 (4)	8/7 (15)	9/12 (21)	11/15 (26)
Not Significant	21/32 (53)	3/1 (4)	9/8 (17)	13/10 (23)	16/16 (32)
Not in Table 4.5	-	0/6 (6)	0/15 (15)	0/20 (20)	0/22 (22)
Total	109	16	51(49)	68(66)	84(82)

Table 4.7: Significance of core/non-core residues identified by voxelGrid

We observed that almost all of the residues in the core were mutated, but only a little more than half of the mutations (25) significantly decreased enzymatic activity. Additionally, it appears that more than half of the residues classified as 'Intermediate' are actually *not* in the core. If we assume that the mutagenesis study identified all functionally important residues, then these residues comprise about 10%[13] of the protein.

voxelGrid does find a fair number of residues that were not mutated, all of which lie outside the core. This is interesting since these are spatially conserved in the set of structures we examined, but were deemed to be not conserved in the MSA analysis used to select residues for mutagenesis. The discrepancy may lie in the difference in diversity and number of sequences (or structures examined): 43 sequences compared to 18 structures.

Since we do not know the significance of residues that were not mutated, we cannot conclude whether voxelGrid performs better than Silverman et al. in selecting functional residues. Performance appears to be comparable, though voxelGrid also has the advantage of being automated.

---

[13] The length of 1YPI is 249 amino acids, so 25/249 is roughly 10%.

Chapter 5: Conclusion

In this study, we have described and implemented an algorithm (voxelGrid) for identifying spatially conserved residues. The algorithm is based on binning amino acid residues into a voxel grid, and is therefore more efficient than traditional methods based on inter-residue distance calculations. The structure alignments used in this study were generated through web-based alignment servers, manual alignment of known functional residues, or Chimera bootstrapping with an MSA. Although voxelGrid's performance depends on the quality of the set of aligned structures, it was shown to be robust to variations among these different alignment methods.

We have reported the results of voxelGrid analysis of two diverse sets of proteins: cAMP binding family of proteins and TIM family of proteins. For cAMP binding proteins, we were able to recover a conserved hydrophobic residue that was not apparent from examining the multiple sequence alignment alone. This is an example of a spatially conserved residue that is not conserved in the corresponding MSA. We extended this type of comparative analysis of sequence and structure alignments to TIM family of proteins, focusing on three key aspects:

1. Spatially conserved residues not conserved in the MSA
2. Residues conserved in the MSA but not spatially conserved in the multiple structure alignment
3. Secondary structure bias in spatially conserved residues

Additionally, our results were comparable to that of a previous large-scale mutagenesis study in identifying functionally significant residues in TIM proteins. Lastly, we demonstrated two further applications of our voxelGrid method: as a quantitative measure of structure alignment quality to supplement traditionally reported RMSD values and as a technique for characterizing information redundancy in a set of aligned structures when used in conjunction with leave-one-out testing.

Proposed Future Work:

One of the shortcomings of a computational study such as we have described here is the need to verify predictions experimentally. Unfortunately, there are few large-scale mutagenesis studies done on a specific protein (or class of proteins). This presents a dilemma as we are unable to assess the functional significance of the residues our algorithm identifies. Even using the extensive mutagenesis study done by Silverman et al., we identified 22 residues which were not tested. These residues are particularly interesting as they are spatially conserved in the set of TIM structures we examined but were deemed to be not conserved in the MSA analysis used to select residues for mutagenesis. In the future, we hope to collaborate with experimentalists to test the functional significance of these residues.

Even without experimental results, voxelGrid can still be a very powerful predictive tool. One interesting application is to search the PDB for all ligands which have at least five structures in the database that bind to it. For each ligand and its associated binding structures, voxelGrid analysis can be used to identify the spatially conserved residues. We hope this will provide insight in identifying the key residues responsible for binding of a specific ligand.

Appendix

From Section 2.2.2:

The ‘residueGroups’ file should contain one group per line. Each line consists of the residues’ three-letter abbreviations separated by spaces. Comment lines are allowed and specified with a leading ‘#’ character. Example:

```
# residue groups
#
# polar (subset: acidic hydrophilic)
ASP GLU TYR
#
# polar (subset: basic hydrophilic)
ARG HIS LYS
#
# polar (subset: neutral)
ASN CYS GLN HIS SER THR TRP TYR
#
# polar
ASP GLU ARG HIS LYS ASN CYS GLN SER THR TRP TYR
#
# non-polar/hydrophobic
ALA GLY ILE LEU MET PHE PRO TRP VAL
```

From Section 2.3.1:

Below is an example of the two script files for visualizing voxelGrid output in Chimera. The first file is simply a series of commands for opening the structures used and displaying only those residues found as hits. The second file is a Python script for displaying the voxels that enclose the spatially conserved residues found as hits. Figure A.1 shows an example of visualizing voxelGrid output in Chimera.

----- File for displaying residues found -----

```
open 1cx4.pdb
open 1ne6.pdb
open 1q43.pdb
open 2cgp.pdb
~show
# Real hits below
disp #0:380.A  #1:354.A  #2:612.A  #3:103.A
disp #0:305.A  #1:284.A  #2:548.A  #3:33.A
# Predicted hits below
disp #0:397.A  #1:371.A  #2:632.A  #3:123.A
disp #0:348.A  #1:322.A  #2:580.A  #3:70.A
disp #0:321.A  #1:300.A  #2:564.A  #3:49.A
```

----- File for drawing voxels enclosing residues found -----

```
import drawBox
drawBox.drawBox((81.214400,55.360000,12.413000),(83.464400,57.610000,14.663000))
drawBox.drawBox((81.214400,66.610000,28.163000),(83.464400,68.860000,30.413000))
drawBox.drawBox((90.214400,59.860000,23.663000),(92.464400,62.110000,25.913000))
drawBox.drawBox((90.214400,66.610000,25.913000),(92.464400,68.860000,28.163000))
drawBox.drawBox((92.464400,68.860000,16.913000),(94.714400,71.110000,19.163000))
```

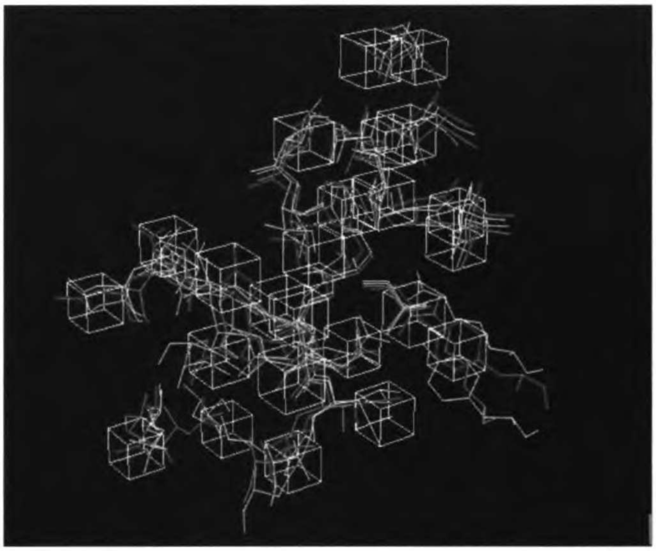


Figure A.1: Example of visualizing voxelGrid output in Chimera

From Section 2.4.2:

PDB files written by Chimera on Windows platform:

When using Chimera to write out PDB files on the Windows platform, it is necessary to remove the return carriage ('^M') from the end of each line before passing the files to voxelGrid. This additional step is not necessary on any other platform, and can be done easily using the translate command: `tr -d '\r' < myStruct > myStruct_noCtrlM`

Multiple structure alignments returned by the CE-MC web server:

The CE multiple structure alignment web server returns PDB files with a non-standard format. It was necessary to remove an extra column in the residue number field (which should be columns 23-26). This was done using a simple Python script. Afterwards, we still had to unpack the single PDB file into separate files for each structure.

References

[1] Babbitt, P.C. "Definitions of enzyme function for the structural genomics era." (2003) *Curr Opin Chem Biol..* **7**, 230-7.

[2] Todd, A.E. et al. "Plasticity of enzyme active sites." (2002) *Trends Biochem Sci..* **27**, 419-26.

[3] Meng, E.C. et al. "Superfamily active site templates." (2004) *Proteins.* **55**,962-76.

[4] Oldfield, T.J. "Data mining the protein data bank: residue interactions." (2002) *Proteins.* **49**, 510-528,

[5] Stark, A. & Russell, R.B. "Annotation in three dimensions. PINTS: Patterns in Non-homologous Tertiary Structures." (2003) *NAR.* **31**, 3341-3344.

[6] Panchenko, A.R. et al. "Prediction of functional sites by analysis of sequence and structure conservation." (2004) *Protein Science.* **13**, 884-892.

[7] Benkovic, S.J. & Hammes-Schiffer, S. "A perspective on enzyme catalysis." (2003) *Science.* **301**, 1196-1202.

[8] Agarwal, P.K. et al. "Network of coupled promoting motions in enzyme catalysis." (2002) *PNAS.* **99**, 2794-2799.

[9] Tousignant, A. & Pelletier, J.N. "Protein motions promote catalysis." (2004) *Chem Biol.* **11**, 1037-42.

[10] Li, L. et al. "Amino acids determining enzyme-substrate specificity in prokaryotic and eukaryotic protein kinases." (2003) *PNAS.* **100**, 4463-4468.

[11] Berman, H.M. et al. "The Protein Data Bank". (2000) *NAR.* **28**, 235-242.

[12] Pettersen, E.F. et al. "UCSF Chimera - A Visualization System for Exploratory Research and Analysis." (2004) *J. Comput. Chem..* **25**:1605-1612.

[13] Guda, C. et al. "CE-MC: a multiple protein structure alignment server." (2004) *NAR.* **32**, W100-3.

[14] Shatsky, M. et al. "A method for simultaneous alignment of multiple protein structures." (2004) *Proteins.* **56**, 143-56.

[15] Pegg, S.C. et al. "Representing structure-function relationships in mechanistically diverse enzyme superfamilies". (2005) *Pac Symp Biocomput.* 358-69.

[16] Thompson, J.D. et al. "The ClustalX windows interface: flexible strategies for multiple sequence alignment aided by quality analysis tools." (1997) *NAR.* **25**, 4876-4882.

[17] Casbon, J. & Saqi, M.A. "S4: structure-based sequence alignments of SCOP superfamilies." (2005) *NAR.* **33**, D219-D222.

[18] Marchler-Bauer, A. et al. "CDD: a Conserved Domain Database for protein classification." (2005) *NAR.* **33**, D192-6.

[19] Shindyalov, I.N. & Bourne, P.E. "Protein structure alignment by incremental combinatorial extension (CE) of the optimal path." (1998) *Protein Eng.* **11**, 739-47.

[20] Holm, L. & Park, J. "DaliLite workbench for protein structure comparison." (2000) *Bioinformatics.* **16**, 566-7.

[21] Zhu, J. & Weng, Z. "FAST: a novel protein structure alignment algorithm." (2005) *Proteins*. **58**, 618-27.

[22] Szustakowski, J.D. & Weng, Z. "Protein structure alignment using a genetic algorithm." (2000) *Proteins*. **38**, 428-440.

[23] Maiti, R. et al. "SuperPose: a simple server for sophisticated structural superposition." (2004) *NAR*. **32**, W590-4.

[24] Berman, H.M. et al. "The cAMP binding domain: An ancient signaling module". (2005) *PNAS*. **102**, 45-50.

[25] Reardon, D. et al. "The structure and evolution of alpha/beta barrel proteins." (1995) *FASEB J*. **9**, 497-503.

[26] Silverman, J.A. et al. "Reverse engineering the $(\beta/\alpha)_8$ barrel fold". (2001) *PNAS*. **98**, 3092-3097.

[27] Donald, J.E. et al. "CoC: a database of universally conserved residues in protein folds." (2005) *Bioinformatics*. **21**, 2539-2540.

[28] Boeckmann, B. et al. "The Swiss-Prot protein knowledgebase and its supplement TrEMBL." (2003) *NAR*. **31**, 365-370.

UC
San Francisco
LIBRARY
UNIVERSITY OF CALIFORNIA

For reference

Not to be taken
from the room.

8071253

3 1378 00807 1253