Structure-Based Querying of Proteins Using Wavelets [*]

Keith Marsolo
Srinivasan Parthasarathy
Department of Computer Science and Engineering
The Ohio State University
srini@cse.ohio-state.edu

Kotagiri Ramamohanarao
Department of Computer Science and Software Engineering
University of Melbourne

ABSTRACT

The ability to retrieve molecules based on structural similarity has use in many applications, from disease diagnosis and treatment to drug discovery and design. In this paper, we present a method to represent protein molecules that allows for the fast, flexible and efficient retrieval of similar structures, based on either global or local attributes. We begin by computing the pair-wise distance between amino acids, transforming each 3D structure into a 2D distance matrix. We normalize this matrix to a specific size and apply a 2D wavelet decomposition to generate a set of approximation coefficients, which serves as our global feature vector. This transformation reduces the overall dimensionality of the data while still preserving spatial features and correlations. We test our method by running queries on three different protein datasets that have been used previously in the literature, basing our comparisons on labels taken from the SCOP database. We find that our method significantly outperforms existing approaches, in terms of retrieval accuracy, memory utilization and execution time. Specifically, using a k-d tree and running a 10-nearest-neighbor search on a dataset of 33,000 proteins against itself, we see an average accuracy of 89% at the SCOP SuperFamily level and a total query time that is up to 350 times faster than previously published techniques. In addition to processing queries based on global similarity, we also propose innovative extensions to effectively match proteins based solely on shared local substructures, allowing for a more flexible query interface.

Categories and Subject Descriptors

E.2 [**Data Storage Representations**]: Object Representation; J.3 [**Life and Medical Sciences**]: Biology and Genetics

General Terms

Algorithms

Keywords

Protein Structure Similarity, Wavelets, Indexing

1. INTRODUCTION

The past decade has seen an explosion in the amount of publicly-available genomic and proteomic information. Low-cost shotgun sequencing has led to the determination of entire genomes, including cat, dog and human. The completion of these projects has generated vast repositories of sequence information. Scientists believe that these sequences hold the key to determining the cause and treatment of many diseases. The subsequent desire to mine this information has led to the development of query and search tools like BLAST and its variants [2, 3].

The protein domain has seen a similar, though smaller, increase in information. For the past decade, the UniProt[1] database, a repository of non-redundant protein sequences, has doubled in size every two years, to a current total of 2.8 million entries. Due to their functional significance, as well as the belief that there is a link between structure and function, there is also considerable interest in the structure of proteins. Solving the structure of a protein molecule, typically through techniques such as X-ray crystallography or Nuclear Magnetic Resonance (NMR), is much more difficult than determining its sequence. Consequently, the number of solved protein structures trails that of determined sequences.

In most cases, once a protein structure has been solved, it is deposited into the Protein Data Bank (PDB)[2]. As of May 2006, the PDB contained over 36,000 structures. While this number is far lower than the number of known sequences, in the past ten years, the size of the PDB has increased sevenfold. The expected development of high-throughput crystallization techniques means that the number of solved structures should increase dramatically. In addition, projects such as Folding@Home[3] are looking to simulate the folding of a protein as it progresses from an unfolded amino acid sequence to its final structure [9]. These simulations are expected to generate a tremendous amount of data and a corresponding increase in intermediate structures that will need to be examined.

[*]This work is supported in part by the following research grants: DOE Award No. DE-FG02-04ER25611; NSF CAREER Grant IIS-0347662.

[1]http://pir.uniprot.org
[2]http://www.rcsb.org
[3]http://folding.stanford.edu

While genomic tools such as BLAST have been adapted to work with protein amino acid sequences, proteome researchers are also interested in determining the *structural* similarity between proteins. Here we present two methods of protein representation that allow for the fast, efficient retrieval of similar structures, based on either global or local features. We begin by computing the pair-wise distance between residues to transform each 3D structure into a 2D distance matrix. To create our global representation, we apply a 2D wavelet decomposition on this matrix to generate a set of approximation coefficients, which serve as our feature vector. Our local representation, which allows us to effectively match proteins based purely on shared substructures, is computed using two separate 1D decompositions. This provides additional functionality that is not available through existing retrieval methods. We evaluate our technique by running classification experiments and nearest-neighbor queries on three previously-examined protein datasets, using labels from the Structural Classification of Proteins database (SCOP) [13], as our basis for comparison. *We find that our global method significantly outperforms the retrieval performance of existing approaches [4, 8], in terms of retrieval accuracy, memory usage and execution time.* Using a k-d tree and running a 10-nearest-neighbor search on a dataset of 33,000 proteins against itself, we see an average accuracy of 89% at the SuperFamily level and an individual query time that is up to 350 times faster than previously published results. We find our retrievals based on local substructure to be highly accurate as well. Finally, while we limit this work to protein data, our techniques can be applied to any structural dataset.

2. BACKGROUND & RELATED WORK

In this section we provide some biological background on the problem domain and a discussion of related work. We begin with a brief overview of proteins and protein structure databases. We follow with details on some of the publications that have looked to determine structural similarity in a database context.

2.1 Proteins

Proteins are comprised of a varying sequence of 20 different amino acids. Individual amino acids are called residues. Each type of amino acid is composed of a number of different atoms, but all contain a central Carbon atom denoted as C_α. The position of this atom is often used as an abstraction for the position of the entire residue. Amino acids combine to form interacting subunits called secondary structures. The interactions between these substructures give rise to the global shape, or backbone, of the protein. Two of the most common secondary structures are α-helices and β-sheets. Residues that belong to these secondary structures are said to have a secondary structure assignment. There are a number of different labeling systems used to assign residues, but in this work, we use the simplest system: a residue can either be part of an α-helix (H), a member of a β-sheet (B), or have no assignment ($-$). All of the 3D information and secondary structure assignments for a protein are obtained from the PDB.

There are several public databases that provide information on protein structures. In our experiments, we use the labels from the SCOP database as ground truth. SCOP arranges proteins into several hierarchical levels based on their structure. The first four levels are *Class*, *Fold*, *SuperFamily* and *Family*. Proteins in the same Class share similar secondary structure information, while proteins within the same Fold have similar secondary structures that are arranged in the same topological configuration. Proteins in the same SuperFamily show clear structural homology (i.e. they share a common structural ancestor) and proteins belonging to the same Family exhibit a great deal of sequence similarity and are thought to be evolutionarily-related.

2.2 Related Work

In the past, there has been a large effort by the database community to develop tools to search for similarity in protein databases. One of the first solutions proposed an algebra for queries that were designed to determine similarity among protein sequences [15]. While useful in that context, it is not immediately apparent how to extend such an algebra to structures. For instance, sequence-based queries allow for partial matches that include gaps. There is a clear notion on the meaning of a gap in terms of sequence, but not when applied to a 3D structure.

Çamoğlu, Kahveci and Singh proposed a method that created feature vectors based on triplets derived from SSEs [7]. These vectors were placed into an R^*-tree, which was used to retrieve similar proteins, pruning the search space for the VAST alignment algorithm, greatly reducing the time needed to conduct a pair-wise alignment of small datasets. However, pair-wise alignment is not feasible on databases with more than a few thousand members.

Recently, there have been a number of publications that have looked to represent proteins for large-scale structure retrieval [4,6,8]. Three of the latest are PSIST [8], ProGreSS [6] and Protdex2 [4]. In PSIST, a feature vector is generated for each protein based on the distances and angles between residues. These vectors are placed into a suffix-tree, which serves as the indexing structure. With ProGreSS, a sliding window is placed on the backbone and several structure and sequence-based features are extracted. Retrieval is handled by a maximum bounding rectangle-based index structure. Protdex2, on the other hand, converts the distances between residues into a matrix and uses SSE-based sub-matrices to derive a feature vector, also including information such as angle, area and amino acid type. Queries are executed on an inverted file index.

3. REPRESENTATION

Here we provide details on the techniques used to construct our wavelet-based feature vectors. We begin with an overview of the wavelet transformation itself. We then describe our approach for creating a global representation of protein structure with a 2D wavelet decomposition and follow with a method designed to capture local substructures using two 1D transformations.

The use of wavelets is natural in applications that require a high degree of compression without a corresponding loss of detail, or where the detection of subtle distortions and discontinuities is crucial [10]. Wavelet decompositions fall into two separate categories: continuous (*cwt*) and discrete (*dwt*). In this work, we deal with the discrete wavelet transform.

Given a decomposition level L and a one-dimensional signal of length N, where N is divisible by 2^L, the *dwt* consists of L stages. At each stage in the decomposition, two sets of coefficients are produced, the Approximation and the Detail. The Approximation coefficients are generated by convolving the input signal with a *low-pass* filter and down-sampling the results by a factor of two. The Detail coefficients are similarly generated, convolving the input with a *high-pass* filter and down-sampling by a factor of two. If the final stage has not been reached, the Approximation coefficients are treated as the new input signal and the process is repeated. In many cases, once the wavelet transformation is complete, the original signal will be represented using combinations of Approximation and Detail coefficients. In our applications, however, *we only use the final level of Approximation coefficients and ignore the rest.*

To operate on a two-dimensional signal (of size N x N), the decomposition proceeds as follows: First, the rows are convolved with a low-pass filter and downsampled by a factor of two, resulting in matrix L ($N/2$ x N). The process is repeated on the original signal using a high-pass filter, which leads to matrix H ($N/2$ x N). The columns of L are convolved two separate times, once with a low-pass filter and again with a high-pass filter. After passing through the filters, the signals are downsampled by a factor of two. This results in a matrix of approximation (LL) and horizontal detail coefficients (LH), respectively (both of size $N/2$ x $N/2$). These steps are executed once more, this time on the columns of H, resulting in a matrix of diagonal (HH) and vertical (HL) detail coefficients (again of size $N/2$ x $N/2$). The whole procedure can then be repeated on the approximation coefficients contained in matrix LL. As in the one-dimensional case, we only deal with the final level of approximation coefficients.

3.1 Global Structure

The first step in generating both types of feature vectors involves converting the protein structure into a distance matrix. This process occurs in the following manner: First, we obtain the 3D coordinates of the protein from the PDB and calculate the distance between the C_α atoms of each residue. We place these values into an n x n matrix D, where n represents the number of residues in the protein and *D(i,j)* represents the distance between the C_α atoms of residues i and j. Figure 1(a) provides a graphical depiction of this matrix (for protein 1HLB), with higher elevations, or larger distances, having a lighter color. In these matrices, secondary structures such as α-helices and parallel β-sheets emerge as dark bands along (parallel to) the main diagonal, while anti-parallel β-sheets appear perpendicular to it. The image in Figure 1(a) represents a pixelated version of the distance matrix. In contrast to this discretized approach, or the related binary 'contact map' representation, where distances below a certain threshold are set to 1 (0 otherwise), we operate on the actual distance values.

To create our global structure representation, we apply a 2D decomposition to the distance matrix. In order to use the results of this transformation as a feature vector, the final number of coefficients must be the same for every protein. This can be achieved either by normalizing the size of the input (the distance matrix), or the output (the approximation coefficients). It is not immediately clear how to normalize a variable-size coefficient matrix while still preserving the necessary spatial correlations. Thus, we elect to normalize the input signal, fixing the size of the distance matrix at 128x128. This normalization occurs either through interpolation or extrapolation, depending on whether the input protein is shorter or longer than 128 residues, respectively. We choose a value of 128 because discrete wavelet transformations are most effective on signals whose length is a power of 2. We prefer to interpolate and smooth, or average the excess points, over extrapolation, where we would be forced to generate additional data. Most of the proteins in our datasets are shorter than 256 residues, thus our choice of 128.

We perform a multi-level 2D decomposition on this normalized matrix and use the final level of approximation coefficients as our feature vector. There are a fairly large number of wavelet families that can serve as filters. We tested several, but found that the simplest, the Haar, worked well for our purposes. Examples of the wavelet decomposition for protein 1HLB can be seen in Figure 1. The figure on the left illustrates the original distance matrix, while the figures in the middle and on the right correspond to the approximation coefficients produced from a 2nd and 4th level decomposition, respectively. We only focus on the approximation values produced by the final level of the wavelet decomposition, so as the decomposition level increases, the number of coefficients decreases by a factor of 4. Despite this large reduction in data, important features such as secondary structures are still present in the figures. Since the matrix is symmetric across the diagonal, we only need to keep the coefficients in the upper (or lower) triangle, plus those that fall on the diagonal itself.

To summarize, the steps taken to create our global representation are as follows:

1. Compute distance matrix using pair-wise distance between C_α atoms.
2. Normalize matrix to size 128 x 128.
3. Apply 2D wavelet decomposition.
4. Extract final level approximation coefficients from the upper half of the matrix plus those that fall on the diagonal.

We calculate the approximation coefficients for several different levels, from 2-8, to see how performance, in terms of accuracy, varies as the dimensionality of the data is reduced. Again, for a given 2D signal, as the level of the wavelet decomposition is increased, the final number of approximation coefficients is reduced by a factor of 4. If this level is set too low, the decomposition will result in a large feature vector, making tasks such as classification difficult. Setting it too high will leave too few attributes, leading to a large amount of information loss and the discarding of potentially-distinguishing information. Both are situations we wish to avoid. Of all the decomposition levels that we tested, we found that the 4th level provided the best performance. Thus, to construct our *global descriptor*, we take those coefficients in the upper triangle of the matrix plus those that fall on the diagonal, which, given a 4th level decomposition, leaves us with a total of 36 coefficients per structure.

3.2 Local Substructures

Typical protein retrieval methods base similarity on overall structure. Indeed, the techniques presented thus far are designed for that very purpose. However, there may be cases where a researcher is more interested in similarity based on

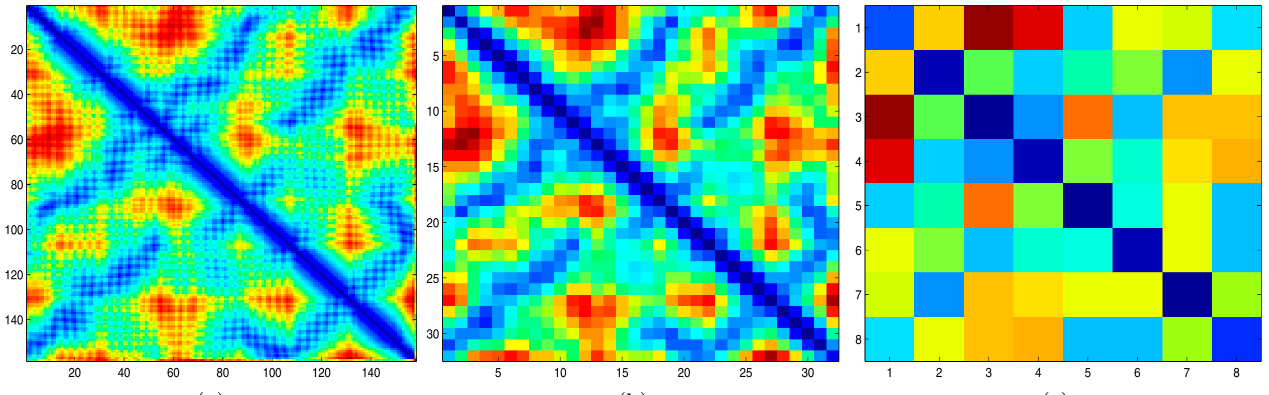
(a) (b) (c)

Figure 1: Left (a): Raw distance matrix for protein 1HLB. Larger distances a denoted with a lighter color. Middle (b), Right (c): Example 2D decompositions for protein 1HLB. The figure on the left (b) represents a 2nd level Haar decomposition. The image on the right (c) corresponds to a Haar decomposition of level 4. The distance matrix on the left is of size 158x158. The figure in the middle contains 1024 coefficients (32x32), while the one on the right is composed of just 64 (8x8).

small pieces of a protein - a particular subsequence, for instance. To that effect, we propose an alternate approach that uses a windowing strategy and two 1D wavelet decompositions to generate substructure-based feature vectors.

Figure 2(a) shows a distance matrix for a hypothetical protein sequence. The letters correspond to the secondary structure assignment of each residue (H - helix, B - sheet, $-$ - no assignment). If we were interested in a seven residue sequence with SSE values of "BBBHHHH," we would only be concerned with the distances contained between the thick black lines. Everything in gray can be ignored. In this manner, every subsequence corresponds to a windowed distance matrix.

While one can manually select a particular query subsequence, to construct subsequences on a target dataset, we need to employ a sliding window strategy. Given a subsequence of length n, for each protein, we start at the first residue, and if the first n residues have the same SSE values as the query, we create a windowed distance matrix for those residues. If not, we move to the next residue of the sequence and repeat the process. We only generate distance matrices for the matching subsequences in order to limit the number of target features that must be generated for each query. This pruning strategy can be disabled to allow for approximate subsequence matches, however.

There is no straightforward way to apply a 2D transformation to a windowed, non-square distance matrix like the one in Figure 2(a). Thus, we elect to use two 1D decompositions, employing a transformation designed to capture the interactions between secondary structures, a strategy based on previous classification experiments [11]. The first of the two decompositions is designed to model the interactions and secondary structures that are *parallel* to the main diagonal of the matrix (α-helices and parallel β-sheets), while the second focuses on those that are *anti-parallel* (anti-parallel β-sheets).

To convert the matrix to a parallel signal, we start with position *D(0,1)* of the matrix and place it at position 0 of the signal. We then place *D(1,2)* at position 1 and proceed down

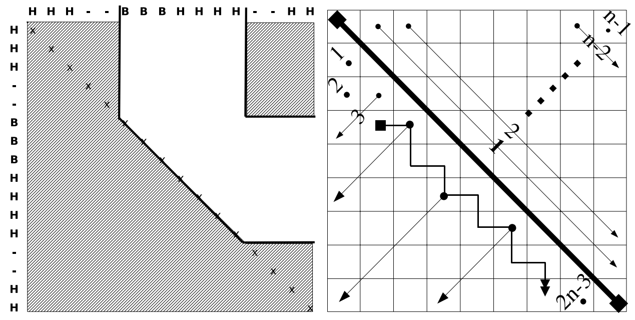
H H H - - B B B H H H H - - H H
H H H - - B B B H H H H - - H H
1 2 3
n-1
n-2
1 2
2n-3
(a) (b)

Figure 2: Left (a): Illustration of distance values selected using windowing strategy. Hatched matrix values are ignored. Right (b): Two methods for converting a distance matrix to a one-dimensional signal. The parallel conversion process is shown in the top triangle, the anti-parallel version in the bottom.

the diagonal. *D(0,2)* will be mapped to position n, *D(1,3)* to *n+1*, and so on, until the matrix has been converted. In this manner, we first add the values closest to the diagonal and gradually move toward the corner. This process is illustrated in the upper right triangle of Figure 2(b).

The anti-parallel conversion process is shown in the lower left triangle of the matrix in Figure 2(b). We normally perform this operation on the upper triangle (as with the parallel process), but in order to match the figure, we use the lower triangle in our description. We start with position *D(1,0)* and place it at position 0 of the signal. We then place *D(2,0)* at position 1. We move right to *D(2,1)*, place it at position 3 and travel away from the diagonal, placing *D(3,0)* at position 4. We continue this stair-step approach, moving our starting point down the diagonal and adding points in a perpendicular fashion until we reach the edge of

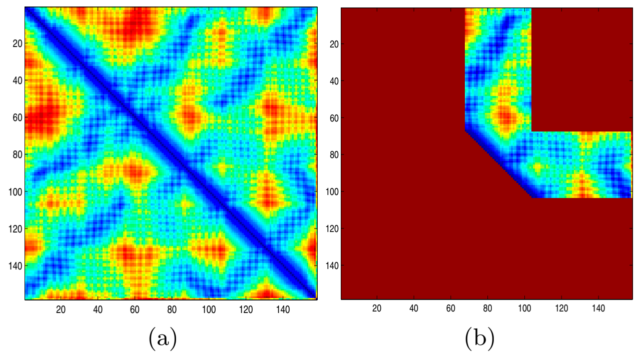

Figure 3: Left (a): Original distance matrix. Right (b): Windowed distance matrix.

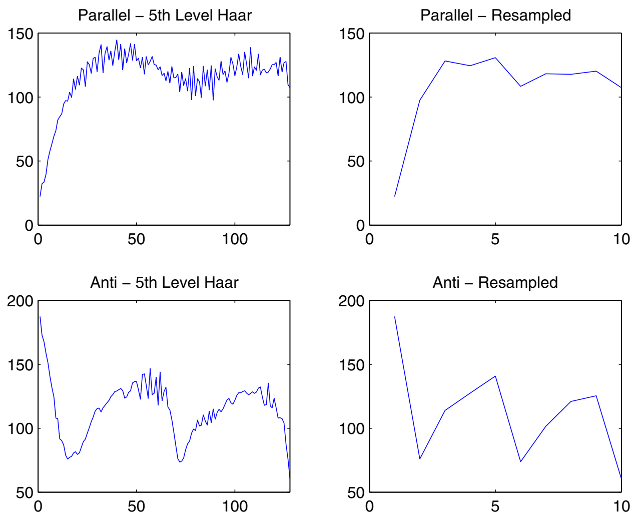

Figure 4: Parallel and Anti-Parallel windows (top and bottom, respectively). The figures on the left represent a 5th level Haar decomposition. The final reduced signals are shown on the right.

the matrix (the movement of the starting point is illustrated by the double arrow in Figure 2(b)).

When we apply these transformations to each windowed distance matrix, we simply ignore the values that fall outside the window. Since the signals generated by this process will be of varying length, we use symmetric extension to ensure that they are divisible by 32. From there, we apply a 5th-level 1D Haar decomposition to each signal and reduce the final level of approximation coefficients to 10 values using interpolation. We concatenate the parallel and anti-parallel signals and treat the resulting 20 attributes as our *local descriptor*.

To summarize the local transformation process:

1. Select query sequence.
2. Generate windowed distance matrices.
3. Convert each 2D windowed matrix to 1D parallel and anti-parallel signals.
4. Apply a 1D wavelet decomposition to each signal, extending them if they are not divisible by 32.
5. Interpolate last level of approximation coefficients to 10 values.
6. Concatenate parallel and anti-parallel coefficients to create local substructure representation.

We provide a graphical representation of the transformation process used to create the local descriptors in Figures 3 and 4. Figure 3(a) represents the original distance matrix for protein 1HLB while Figure 3(b) shows the same matrix after it has been "windowed." This window represents a sequence of 36 residues selected from the middle of the protein for illustrative purposes. The results of our parallel and anti-parallel conversions can be seen in Figure 4 (top and bottom, respectively). The left images contain the results of the 5th level Haar decomposition, while the rightmost images depict the final, reduced descriptors.

3.3 Benefits & Use Cases

Wavelets are used frequently within the vision and image processing community for clustering and retrieval. In many cases, images are retrieved based on a subset of the most dominant coefficients corresponding to some underlying property of the image. While effective in some applications, we cannot rely on such a subset. It is true that part of our rationale for using wavelets is that the secondary structure information will be contained in the dominant coefficients. However, databases like SCOP classify proteins based on the topological arrangement of secondary structures. Many proteins share the same types of secondary structures, and therefore, will have dominant coefficients that are almost identical. Thus, we must also take into account the relative position and location of the secondary structures. Our transformation was designed to capture and retain this topological information. It also has the added benefit of being robust in the presence of noise. If the noise is uniformly distributed, it will be spread throughout the transformation and will not affect the relative values of the results.

Another benefit to using wavelets is that the transformation algorithm is largely parameter-free. Aside from choosing the filter type and initial matrix size, little else is required from the end user. It is true that they must choose which coefficients to use as a feature vector, but it is possible to compute everything at once and make a decision at a later date. Switching from a feature vector composed of 4th level approximation coefficients to one of 5th level values does not affect the overall transformation, only the experimental validation. We feel this is a tremendous advantage over other representations that contain a large number of tunable parameters, all of which affect the computation time and final quality of the overall transformation.

Given a database of protein structures, there are several tests that can be used to evaluate the effectiveness of any representation. We evaluate our approach using *k-nearest-neighbor* (KNN) and *range*-based query retrievals [1]. Applied to a target database, a KNN search looks to find the k "nearest" neighbors to a given query. A range query is similar to KNN retrieval, except that instead of providing a value for k, the user specifies a range r and the search returns all the objects less than a distance r from the query. In our evaluation, distance refers to the Euclidean distance

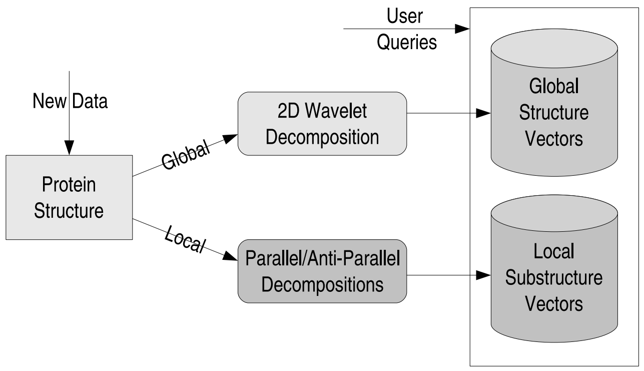

Figure 5: Proposed Workflow.

between attributes. Allowing the user to specify the range means they have more fine-tuned control over the quality of the results, which can be more effective when the number of potential matches is either quite low or very high. The drawback is that the number of matches is now dependent on the distance between the objects, making the range parameter a function of the dataset and the representation. This dependency occurs regardless of the transformation or model that is chosen.

We envision our transformation techniques being used in a database context, allowing users to submit queries in order to retrieve proteins based on structural similarity. Figure 5 provides an illustration of how this system would be implemented. Given a new protein structure, the global descriptors would be generated using a 2D wavelet decomposition. This can be done offline. The local substructure descriptors, generated using the windowed, 1D conversion, can also be generated offline, though it would be impractical to generate local descriptors for every possible subsequence. A more intelligent solution would be to create and store the local descriptors of the more popular subsequences offline and then wait to generate the descriptors of any other subsequence until after a user has submitted a specific substructure query. To avoid the need for continuous regeneration, the descriptors of the more frequent queries can be cached, similar to the strategy employed by BLAST. In either case, once the descriptors have been generated, all of the user interaction would be on the reduced feature vector space, not the original structure data.

4. DATASETS

To evaluate our representations, we conduct experiments on three different datasets, comparing our results with two of the more recently-published techniques [4, 8]. In order to make a valid comparison, we try to use the exact datasets tested in those works. Due to discrepancies or errors in the original data files, there are some differences, which we note below.

The first two datasets were used by Aung and Tan in their Protdex2 experiments [4]. The first of these datasets contains 200 proteins taken from version 1.59 of the SCOP database that have less than 40% sequence homology (similarity) to each other. We prefer to test proteins with a low sequence similarity because proteins with a high sequence similarity are likely to be almost identical structurally and

Dataset	Class	Fold	SuperFamily	Family
2K	4	133	181	281
33K	7	623	960	1632

Table 1: Number of unique groups for the 2K and 33K datasets at different levels of the SCOP hierarchy.

would thus be a poor choice for retrieval experiments. 10 proteins belonging to the *Globins* family and 10 to the *Serine/Threonin Kinases* were selected. These 20 proteins comprise a set of queries to be tested against 180 proteins randomly selected from the other families within SCOP. In trying to establish a link between the dataset ID and the SCOP ID, we were unable to match the names of 11 of these proteins, so our variant, which we refer to as the *Small* set, contains 189 members. We use this dataset for our substructure matching experiments. The second dataset, which we call the *33K* set, originally contained 34,055 members. 108 proteins belonging to 108 medium-sized families (having ≥ 40 members and ≤ 180 members) were chosen as representative queries. Our version of this dataset contained 33,010 proteins and 107 queries.

The final dataset has been examined in a number of publications [6, 7], most recently by Gao and Zaki [8]. Again taken from the SCOP database (though a different version than the one used in constructing the Small and 33K sets), this set contains a total of 1810 proteins, with 10 proteins selected from 181 different SuperFamilies. When running queries on this set, 1 protein was randomly selected from each SuperFamily. As before, when trying to establish a link between the dataset ID and the SCOP ID, we could not match 22 proteins, which leaves us with a total of 1798. Approximately 1600 proteins from this dataset, which we call the *2K* set, are present in the *33K* set. Table 1 provides the number of unique groups for the 2K and 33K datasets over the different levels of the SCOP hierarchy.

5. EXPERIMENTAL RESULTS

We validate our approach with four different sets of experiments. The first is designed to showcase the predictive ability of our technique by using it to classify selected query proteins according to their SCOP labels. The next two illustrate the robustness of our feature vector by using it, given a query protein, to retrieve a large number of structurally-similar proteins (as defined by SCOP). Our final experiments show how our window-based representation can be used to search for specific substructure sequences. We conclude with a performance analysis in terms of query execution time and overall memory usage.

All experiments were conducted on a 2.4 GHz Pentium 4 PC with 1.5 GB RAM running Debian Linux on a 2.6.9 kernel. We use a *k-d tree* as our target data structure [5]. The k-d tree is a generalization of the binary search tree. Both start with a root node and place the data into left and right subtrees based on the value of a particular attribute, or key. This process is repeated recursively until the data is divided into mutually exclusive subsets. While a binary search tree uses the same key on all nodes of all subtrees, a k-d tree will use a different key at each level. This leads to a data structure that effectively partitions the data but still allows the user to search for best matches without having to make a large number of comparisons. Rather than re-implement this data structure ourselves, we use k-d tree code obtained from the Auton Lab[4].

We also wish to compare our technique with PSIST and Protdex2. We were able to obtain the source code for PSIST[5], but were unable to do so for Protdex2. We were, however, able to download an evaluation version of Protdex2, though it provides limited functionality. As a result, we only use it to generate timing results, which we discuss at the end of this section.

To evaluate PSIST, we generate the input features required by the program and test our datasets. Unfortunately, while we were able to obtain results with the 2K dataset, to process the 33K dataset, the suffix tree generated by PSIST required more memory than was available on our machine (1.5 GB), and we were forced to abort the tests. It may be possible for PSIST to run on a machine with more memory, but both Protdex2 and our own solution require less than 100 MB of memory to process the same dataset.

The PSIST algorithm allows the user to tune a number of parameters, all of which have effects on both the accuracy and computation time. Using the parameter values reported in the original work [8], we evaluated the performance of the algorithm on the 2K dataset. We found that the default settings yielded an accuracy equivalent to those listed previously as the "best," along with a drastic reduction in running time. While another set of parameter values might provide better performance, we have no intuition to guide their selection. Thus, we report results obtained with the program defaults.

5.1 Classification

We examine the predictive accuracy our technique by using the k-d tree as a nearest-neighbor classifier. Given a protein, we retrieve from the target database the three nearest neighbors to the query. We then compare the SCOP labels of the query and the retrieved proteins. If the labels of at least two of the neighbors match, we say that the query has been classified correctly (the query proteins are not included in the target database). Similar classification experiments are detailed in the paper on PSIST [8]. In that work, proteins were only compared at the Class and SuperFamily level. We also examine the accuracy of our method at the Fold and Family levels. As one progresses down the SCOP hierarchy, the protein structures become increasingly similar and the distinctions between them more fine-grained. A truly useful representation should not lose accuracy during this descent, even though the classification problem itself becomes more difficult. We examine the effect of our approach on both the 33K and 2K datasets and compare against the results returned by PSIST on the 2K dataset. Unfortunately, as the PSIST code only provides accuracy values for Class and SuperFamily, we do not have results for Fold or Family.

Table 2 shows the classification accuracy for our representation on the 2K and 33K datasets at different levels of the SCOP hierarchy. Our technique strategy provides exceptional performance at the Class level, and highly accurate results as one progresses down the hierarchy. Our results on the 2K dataset are equal to those of PSIST at the Class level and almost 3% higher at the SuperFamily level. These accuracies are essentially equivalent, but we do see a greater difference in performance in the nearest-neighbor experiments, which we discuss in the section below.

Dataset	Class	Fold	SuperFamily	Family
2K - 2D	98.3	96.6	96.6	95.5
2K - PSIST	98.3	$--$	93.9	$--$
33K - 2D	100	92.5	91.5	89

Table 2: Classification accuracy for PSIST and the 2D representation on the 2K dataset at different levels of the SCOP hierarchy. Results are provided for the 2D representation on the 33K dataset as well. Accuracy is shown as a percentage of correct classification (total correct out of 181 for the 2K dataset, 107 for the 33K).

Dataset	Queries	$k = 4$	$k = 10$	$k = 50$	$k = 100$
2K - PSIST	181	3.75	7.04	7.50	7.74
2K - 2D	181	3.85	7.74	8.17	8.45
33K - 2D	107	3.75	8.59	29.4	42.5
33K - 2D	33K	3.86	8.93	32.7	51.5

Table 3: Average number of proteins in matching SuperFamily for varying values of k as returned by a nearest-neighbor query.

For the Fold, SuperFamily and Family levels, our performance on the 33K dataset is about 5% lower than the values reported on the 2K data. This is to be somewhat expected, though, as the 33K dataset presents a much tougher classification challenge than the 2K data. At the Class level, we do see a higher accuracy on the 33K dataset than the 2K dataset. This fact is not as odd as it may appear, as the 2K dataset is not a complete subset of the 33K dataset and they are evaluated with different sets of query proteins.

There are a number of reasons why we typically expect to see lower classification performance on the 33K data, especially at the lower levels of the SCOP hierarchy. First, the dataset is roughly 18 times larger than the 2K set. Second, there are many more potential categories in the 33K dataset. As illustrated in Table 1, at the SuperFamily level, the 2K dataset contains 181 unique groups, one per query. The 33K dataset, on the other hand, contains 960 unique SuperFamilies. The 108 query SuperFamilies represent just 11% of the total. At the Family level, that percentage drops to less than 7.

5.2 Nearest-Neighbor Retrieval

For our nearest-neighbor retrieval tests, we again use SCOP labels to determine whether a neighbor is correct. We examine the number of correct matches for k equal to 4, 10, 50 and 100. As before, these values have been used in previous retrieval experiments [8]. We evaluate the 2K and 33K datasets in our tests and compare against the 2K values returned by PSIST.

The results of our retrieval tests are presented in Table 3. Shown in the table are the average number of nearest-neighbors that belong to the same SuperFamily as the query protein for different values of k. *On the 2K dataset, we re-*

[4]http://www.autonlab.org

[5]We would like to thank F. Gao and M. Zaki for providing the source code and for their assistance in interpreting the results.

Dataset		$r = 25$	$r = 50$	$r = 125$
2K	Match	2.87	3.79	5.43
(181 Queries)	Total	2.87	3.79	5.43
33K	Match	5.46	7.8	17.4
(107 Queries)	Total	5.46	7.8	17.4
33K	Match	16.6	28.0	56.2
(33K Queries)	Total	16.6	28.0	56.2

Table 4: Average number of objects returned by a range query for different range values (r).

turn a larger number of correct neighbors as PSIST for all the tested values of k. For $k = 10$ and larger, we retrieve an additional 0.7 correct neighbors. There is little performance gain for values of k larger than 10, but that is because the target database only contains 10 members for each SuperFamily.

We also report good performance on the 33K dataset. When $k = 10$, the number of correct neighbors is between 8.5 and 9, depending on the query load. As k increases, we continue to return a larger number of nearest neighbors, increasing from around 9 when $k = 10$, to approximately 30 and 40-50 when $k = 50$ and $k = 100$, respectively. Performance tapers off to some degree for large k, partly because many SuperFamilies contain less than 50 members.

5.3 Range Queries

We test our approach with a range query for three different range values: 25, 50 and 125. The range is largely dependent on the representation. If set too low, very few objects will be returned. If the chosen value is too high, it is possible to return the entire dataset as a match for each query. These values, determined empirically, were selected because the number of objects returned covers between 25-50% of the possible total. We report the average number of objects retrieved (total number of objects retrieved divided by the total number of queries) as well as the average number of matches among those objects. A match occurs when the object and query belong to the same SuperFamily. The results of this experiment are provided in Table 4.

As we can see, range queries are highly accurate, but the number of retrievals is also highly variable. For instance, when the range value is set to 25, we retrieve an average of 2.87 objects on the 2K dataset, but 5.46 or 16.6 on the 33K dataset, depending on the number of queries. However, even as the range value is increased to 125, our accuracy still remains at 100% (average number of matches / average number of retrievals). Considering that the 2K dataset only contains 10 members per SuperFamily, and the 33K dataset contains between 40 and 108, *we are retrieving half of the possible total with no error.*

5.4 Substructure Matching

The results presented above illustrate the effectiveness of our technique at matching protein structures based on global similarity. Here we highlight the ability of our approach to retrieve structures based on local similarity. To test our proposed technique, we take the Small dataset and generate windowed distance matrices for all subsequences of size 10. For each matrix, we create a local descriptor using the process described in Section 3.2.

The Small dataset contains 189 proteins, but there are over 44,000 windows of size 10. We select three different subsequences, *HHHHHHHHHH*, *BBBBBBBBBB* and *−−BBBBBB−−* as our queries. These subsequences were chosen because they provide test sets large enough to validate our approach. There are 4899 *HHHHHHHHHH* windows, 341 *BBBBBBBBBB* and 163 *−−BBBBBB−−*. We select 67 of the *HHHHHHHHHH* windows, 8 of the *BBBBBBBBBB*, and 3 from the *−−BBBBBB−−* group to act as queries. The remaining windows serve as a target database. In this manner, a separate database is created for each window type.

Having removed the query windows from the database, we repeat our classification experiments, retrieving 3 nearest neighbors and comparing SCOP labels to determine accuracy. We provide information on the number of members in each Family in the query and target databases in Table 5. While we test this method on the entire Small dataset, we envision it being used as a refinement to the results of a nearest-neighbor retrieval. Generating window-based feature vectors on only a subset of the entire dataset would drastically reduce the number of overall features produced.

The results of our substructure classification experiments are presented in right column of Table 5. *Our local representation performs very well, misclassifying only one query out of 78.* Despite the fact that the datasets contained a relatively large number of families, we were able to choose the correct one in almost every case. When generating datasets for a given substructure window, there may not be many objects in the database that belong to the same Family as the query. In these cases, a range query may be more appropriate than a strict nearest-neighbor query.

5.5 Performance

In this section we provide an analysis of query execution time and memory usage of the tested methods. As stated previously, we were only able to obtain an evaluation version of the Protdex2 code. Therefore, we simply use the software to generate timing results. In addition, while all techniques were evaluated on the same machine, we had to execute the Protdex2 software using Windows. The other two methods were tested under Linux.

Query Execution Time

Here we discuss the running time for our KNN experiments where k equals 100, the longest of all our retrieval tests. We list these values in Table 6. The numbers in the *Total Time* column include the time necessary to read both datasets (target and query) into memory, build the *k-d tree*, execute the queries, and write the results to disk. The column labeled *Query Time* shows only the time spent conducting the queries.

To process 181 queries on the 2K dataset, the total query time required by PSIST is 92 seconds, or roughly 0.51 seconds per query. Our wavelet-based approach, in conjunction with a k-d tree, however, requires less than 1 second to process the same queries. This yields an average query time of less than 0.006 seconds, *an improvement of more than 80-fold.* To execute 107 queries on the 33K dataset, Protdex2 takes a total of 150 seconds, or an average of 1.4 seconds per query (the program does not provide a breakdown of the

Window Type	SCOP Family ID	Number of Queries	Number in Database	Query Accuracy
HHHHHHHHHH	56113	21	231	100
($F = 112$, $n = 4832$)	46463	46	372	100
BBBBBBBBBB	51534	4	3	75
($F = 45$, $n = 333$)	53851	4	17	100
−−BBBBBB−−	56113	2	6	100
($F = 76$, $n = 160$)	51751	1	9	100

Table 5: Membership information for the substructure window datasets. For each window type, the number of different families (F) and size of the target database (n) are given. Each window type has an associated query family, provided for which are the SCOP ID and number of query and database members. The last column lists the query accuracy for the three different substructure windows at the Family level. The values represent percentage of correct classification on a 3-NN classifier.

Database Size	Number of Queries	Total Time	Query Time	Time per Query
33K - 2D	33,010	160	135	0.004
33K - 2D	107	8	3	0.028
33K - Protdex2	107	150	–	1.4
1798 - 2D	181	2	< 1	< 0.006
1798 - PSIST	181	107	92	0.51

Table 6: Execution time (in seconds) for various database and KNN query workloads ($k = 100$)

time spent in the query phase). On the same machine, our method requires only 3 seconds to execute these queries, *a 50-fold decrease in running time.* In addition, we see that increasing the number of queries actually reduces the average time spent on an individual query. Querying the entire 33K dataset against itself increases the total number of queries by a factor of 308. The total query time, however, only increases by a factor of 45. Thus, we see a reduction in the average time per query, from 0.03 seconds to just 0.004. *This represents a 350-fold improvement over the average time per query reported by Protdex2.*

Memory Usage

Our representation is much less memory-intensive than PSIST and comparable to Protdex2. On the 2K dataset, PSIST requires roughly 80MB of memory, compared with around 12 MB for our technique. With 33,000 proteins, PSIST requests 1.6 GB of memory, maxing out our test machine and forcing us to abort the experiments. Protdex2 requires around 20MB of memory to execute a single query. Our wavelet representation, coupled with the k-d tree, requires just 80 MB to load the dataset and process 33K queries. The 33K dataset is roughly 18 times larger than the 2K set. PSIST requests 20 times the memory, which would imply a linear relationship between the size of the dataset and the memory required to process it. Our approach on the other hand, shows sublinear growth. *An 18-fold increase in the size of the dataset leads to just a 7-fold increase in memory usage.* Figure 6 provides a graphical representation of these relationships. This indicates that our representation is very scalable, and can easily be deployed as a simple, but accurate, system for determining structure similarity.

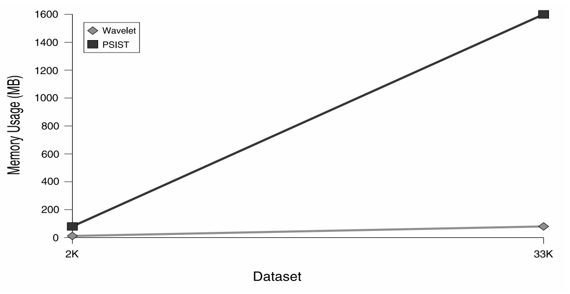

Figure 6: Memory usage (in MB) of the 2D wavelet approach versus PSIST on the 2K and 33K datasets.

Another factor to consider is the amount of disk space needed to store the query representations as well as the results. Protdex2 requires that each query be represented using a PDB-style coordinate file (PSIST requires a somewhat similar, though more compact, format). For the 107 query proteins tested, these files requires around 19MB of storage space. To evaluate 33K queries with Protdex2, one would need roughly 6GB to hold all the query information. This is in contrast to our wavelet-based approach, which can represent the entire 33K dataset in just a few MB. In addition, Protdex2 generates a score for the similarity of each query to every object in the database. On the 33K dataset, this yields a 2MB scoring summary for each query. For 107 queries, these summaries require a few hundred MB of storage space. To process 33K queries, on the other hand, one would need 65GB just to store the results. While it is true that storage is relatively inexpensive, managing and processing that much information is not a trivial task. With our approach, we only provide the number of nearest-neighbors requested by the user, drastically reducing the amount of data generated.

6. CONCLUSIONS AND FUTURE WORK

We present two methods of protein representation that allow for quick and efficient structure queries. We provide implementations for retrieving proteins based on the similarity of either their global shape or smaller substructures, an option that is not provided in any of the leading strategies. When viewed against current techniques, our method is superior in accuracy and more importantly, can answer user queries in a fraction of the time. We have tested our approach by running a number of queries on several different datasets of protein structures. We have validated our results against the labels and categorizations of a leading structural database. We find that even as we progress down the hierarchy of the database, there is not a significant decrease in the accuracy of our method, indicating that the proteins we retrieve are truly correct.

In the future, we plan to test and refine our substructure matching technique. While not immediately obvious, we would like to determine whether one can obtain meaningful results if we allow substructure matching with gaps in the query sequence. In addition, we plan to evaluate the use of space-filling curves [14] as a sampling method to see how they perform compared to our combined parallel/anti-parallel approach.

Thus far, we have focused only on finding similarity among solved protein structures. We do not feel that our technique is limited solely to this task, however. We believe that the methods presented here can be used on a number of different applications, particularly simulations that operate on structure-based data. Two examples include defect tracking in molecular dynamics (MD) simulations and the modeling of protein folding pathways [9, 12]. MD simulations are used to model the behavior of spurious atoms as they move through a lattice of a base material, often silicon. Given a set of simulation frames, we could apply our algorithm to create a sequence describing that set, which could then be compared against other sets. Being able to characterize the behavior of these defects would be a boon to those who work in the manufacture of semiconductors.

It is well-established that the amino acid sequence of a protein determines its final structure. What is unknown, however, is the true nature of the role that the primary structure plays in the folding process. In addition, the intermediate steps that a protein undergoes as it transitions from an unfolded chain to its final formed structure remain a mystery. On occasion, a protein will "misfold," resulting in an abnormal shape. These abnormalities can lead to diseases such as Alzheimer's, Cystic Fibrosis, and Creutzfeldt-Jakob's (the human equivalent of Bovine Spongiform Encephalopathy or Mad Cow). As a result, there has been tremendous effort to understand the protein folding process through computer simulations like Folding@Home. As these programs grow in popularity, they will generate an enormous amount of simulation data. Fast and efficient characterization techniques such as the one proposed here will be crucial if there is to be any hope at processing the results of these simulations in a timely fashion.

Given a set of simulation frames, we can apply our technique on the results from different proteins and cluster them based on similarity. Large clusters may be indicative of certain key points along the folding pathway. Clusters of consistently abnormal structures may give researchers a clue as to how and why proteins misfold. In addition, there may be instances where two groups of proteins share a portion of the pathway and then split before folding into their final shape. Such a characterization could provide insight into the evolution of protein structures and remains a rich direction of future study.

7. REFERENCES

[1] D. Aha and D. Kibler. Instance-based learning algorithms. *Machine Learning*, 6:37–66, 1991.

[2] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *J. of Mo. Biol.*, 215:403–410, 1990.

[3] S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Anang, W. Miller, and D. J. Lipman. Gapped BLAST and PSI-BLAST: A new generation of protein database search programs. *Nucleic Acids Research*, 25:3389–3402, 1997.

[4] Z. Aung and K.-L. Tan. Rapid 3d protein structure database searching using information retrieval techniques. *Bioinformatics*, pages 1045–1052, 2004.

[5] J. L. Bentley. Multidimensional binary search trees used for associate searching. *Comm. ACM*, 18(9):509–517, 1975.

[6] A. Bhattacharya, T. Can, T. Kahveci, A. Singh, and Y. Wang. ProGreSS: Simultaneous searching of protein databases by sequence and structure. In *Pacific Symposium on Biocomputing*, volume 9, pages 264–275. World Scientific Press, 2004.

[7] O. Çamoğlu, T. Kahveci, and A. Singh. Towards index-based similarity search for protein structure databases. In *2nd IEEE Computer Society Bioinformatics Conference (CSB)*, pages 148–158, 2003.

[8] F. Gao and M. Zaki. PSIST: Indexing protein structures using suffix trees. In *IEEE Computational Systems Bioinformatics Conference*, Palo Alto, CA, August 2005. IEEE.

[9] S. M. Larson, C. D. Snow, M. Shirts, and V. S. Pande. Folding@home and genome@home: Using distributed computing to tackle previously intractable problems in computational biology. In R. Grant, editor, *Computational Genomics*. Horizon Press, 2002.

[10] S. Mallat. *A Wavelet Tour of Signal Processing*. Academic, New York, 2nd edition, 1999.

[11] K. Marsolo and S. Parthasarathy. Alternate representation of distance matrices for characterization of protein structure. In *5th IEEE International Conference on Data Mining (ICDM05)*, 2005.

[12] S. Mehta, S. Barr, A. Choy, H. Yang, S. Parthasarathy, R. Machiraju, and J. Wilkins. Dynamic classification of anomalous structures in molecular dynamics simulation data. In *Proceedings of the SIAM Conference on Data Mining*, 2005.

[13] A. G. Murzin, S. E. Brenner, T. Hubbard, and C. Chothia. Scop: a structural classification of proteins database for the investigation of sequences and structures. *J. Mol. Biol*, 247:536–540, 1995.

[14] L. Platzman and J. Bartholdi. Spacefilling curves and the planar travelling salesman problem. *J. Assoc. Comput. Mach*, 46:719–737, 1989.

[15] S. Tata and J. Patel. PiQA: An algebra for querying protein data sets. In *SSDBM*, pages 141–150, 2003.