Structure-Based Querying of Proteins Using Wavelets

Keith Marsolo
Srinivasan Parthasarathy
Department of Computer Science and Engineering
The Ohio State University
contact: 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 for representing protein structures that allows for the fast, flexible and efficient retrieval of similar structures. We use pairwise inter-atom distances to transform the 3D structure into a 2D distance matrix. We normalize this matrix to a specific size and apply a multi-level wavelet decomposition to generate a series of approximation coefficients, which serve as our 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 used previously in the literature, using labels from SCOP as our basis for comparison. 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 SuperFamily level and a total query time that is approximately 125 times lower than previously published techniques. In addition to processing queries based on global similarity, we also propose innovative extensions to effectively match proteins based purely on shared local substructures, allowing for a more flexible query interface.

1. INTRODUCTION

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

There has been a similar increase in information in the protein domain. 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. However, because of their functional significance, and the belief that there is a link between structure and function, there is also considerable interest 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, the PDB has doubled in size every four years. The expected development of high-throughput crystallization techniques means that the number of solved structures will increase dramatically in the coming years. 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 [12]. These simulations are expected to generate a tremendous amount of data and a corresponding increase in intermediate structures that will need to be examined.

While 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 use pairwise inter-atom distances to transform the 3D structure into a 2D distance matrix. To create our global representation, we apply a multi-level 2D wavelet decomposition on this matrix to generate a series of approximation coefficients, which serve as our feature vector. We compute the local representation using two 1D decompositions that allow us to effectively match proteins based purely on shared substructures. This allows for additional functionality that is not available through existing retrieval methods. We evaluate our technique by running classification and nearest-neighbor queries on three protein datasets used previously using la-

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

bels from the Structural Classification of Proteins database (SCOP) [17] as our basis for comparison. *We find that our global method significantly outperforms the retrieval performance of existing approaches [4, 10], 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 approximately 125 times lower 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 biological background on our 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 use protein databases to determine structural similarity. We conclude with a high-level presentation of the fundamentals of the wavelet transformations used to construct our structural representations.

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, all contain a central Carbon atom denote as C_α. The position of this atom is often used as an abstraction for the position of the entire residue. The sequential connection of the C_α atoms is called the protein backbone. Amino acids combine to form interacting subunits called secondary structures. 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 structure. In our experiments, we use the classifications from SCOP database as ground truth. The SCOP database arranges proteins into several hierarchical levels. The first four 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 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 in the database community to develop tools to search for similarity in protein databases. One of the first proposed an algebra to handle queries to determine similarity among protein sequences [11, 21]. While useful on sequences, it is not immediately apparent how to extend such an algebra to structure-based datasets. For instance, sequence 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 of finding similarity that created feature vectors based on triplets derived from SSEs [8]. These vectors were placed into an R^*-tree, which was used to retrieve similar proteins as a pruning step for the VAST alignment algorithm [13], greatly reducing the time needed to conduct a pair-wise alignment of small datasets. However, pair-wise alignment is not feasible on very large databases, even those containing just 30,000 structures. Finally, Tan and Tung proposed a method to convert a protein into a series of 3D substructures. These substructures were clustered and sequential substructures were merged [20]. However, this method again relies on pair-wise alignment and does not scale well to large datasets.

Recently, there have been a number of publications that have looked at representing proteins for large-scale structure retrieval [4, 6, 7, 10]. Three of the latest are PSIST [10], 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 sequenced-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, including information such as angle, area and amino acid type. Queries are executed on an inverted file index.

2.3 Wavelets

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 [14]. Wavelet transformations fall into two separate categories: continuous (*cwt*) and discrete (*dwt*). In this work, we deal with the discrete wavelet transform.

Given a one-dimensional signal of length N (typically a power of two), the *dwt* consists of at most $\log N$ stages. At each stage, two sets of coefficients are produced, the approximation and detail coefficients. The approximation coefficients are generated by convolving the original signal with a low-pass filter, the detail coefficients with a high-pass filter. After passing the signal through the filter, the results are downsampled by a factor of two. This step is repeated on the approximation coefficients, producing another smaller set of approximation and detail coefficients.

To operate on a two-dimensional signal, 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 r_L. The process is repeated on the original signal using a high-pass filter, which leads to matrix r_H. The columns of r_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 set of approximation ($r_L c_L$) and horizontal

detail coefficients ($r_L c_H$), respectively. These steps are executed once more, this time on the columns of r_H, resulting in a set of diagonal ($r_H c_H$) and vertical ($r_H c_L$) detail coefficients. The whole procedure can then be repeated on the approximation coefficients represented in matrix $r_L c_L$. 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.

Wavelets are used frequently within the vision and image processing community for clustering and retrieval [9, 18, 22]. In many cases, images are retrieved based on a subset of the most dominant coefficients that correspond to some underlying property of the image. While effective for such applications, we require a stronger measure of similarity. Our rationale for using wavelets is based on the belief that we can represent proteins in such a way that secondary structure information is captured in the transformation process. Databases like SCOP classify proteins based on the topological arrangement of secondary structures. Therefore, we must take into account the relative position and location of the secondary structures.

3. REPRESENTATION

In this section, we provide details on the techniques used to construct our wavelet-based feature vectors. We begin by describing our approach for creating a global representation of protein structure using a 2D wavelet decomposition and follow with a method designed to capture local substructures using two 1D transformations.

3.1 Global Structure

The first step in generating both of our 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 (or parallel to) the main diagonal, while anti-parallel β-sheets appear perpendicular to it.

To create our global structure representation, we apply a 2D decomposition to the distance matrix. 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 done either by normalizing the size of the input (distance matrix), or the output (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 to 128x128. This normalization occurs through interpolation or extrapolation, depending on whether the input protein was shorter or longer than 128 residues. We choose a value of 128 because discrete wavelet transformations are most effective on signals whose length is a power of 2. We prefer interpolation, smoothing or averaging the excess points, to 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. Examples of the wavelet decomposition can be seen in Figure 1. The figure on the left illustrates an original distance matrix, while the figures in the middle and on the right correspond to a 2nd and 4th level decomposition, respectively. 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.

We calculate the approximation coefficients for several different levels (2-5), to see how performance, in terms of accuracy, varies with the reduction in the dimensionality of the data. 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 set too low, the decomposition will result in a large, possibly noisy feature vector, making it difficult to distinguish between the different groups. Setting the level too high will lead to a large amount of information loss, discarding potentially-distinguishing information. Both are situations we wish to avoid. We find the performance of the 2nd, 3rd and 4th level coefficients to be roughly the same. Thus we elect to use the 4th level decomposition as it provides the largest reduction of data among the three. This leaves us with a *global descriptor* of 36 coefficients per protein structure.

3.2 Local Substructures

Typical protein retrieval methods base similarity on overall structure. Indeed, the techniques presented thus far were designed for that very purpose. However, there may be cases where a researcher is more interested in similarity based on small pieces of the 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 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 the subsequences on the 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, though this pruning strategy can be disabled to allow for approximate

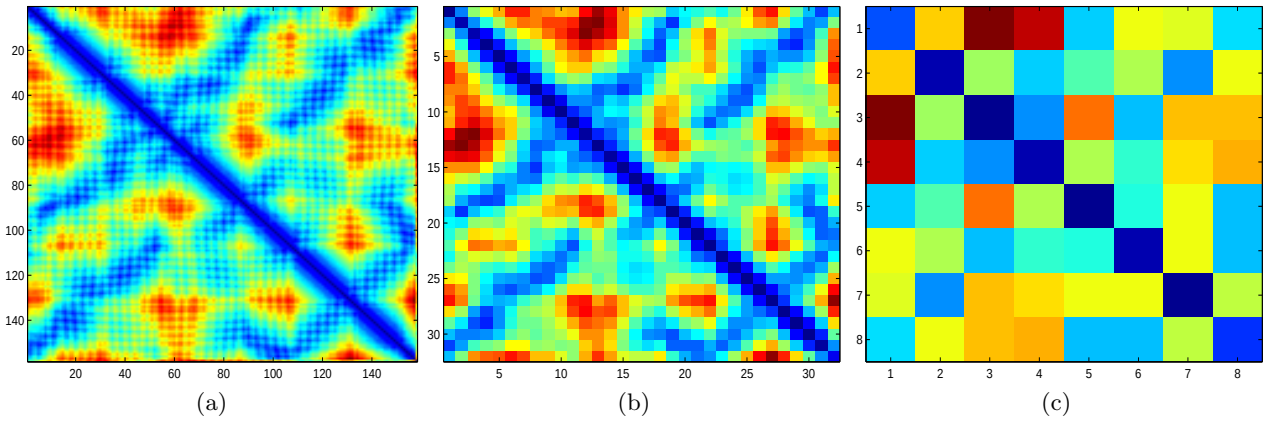

Figure 1: Left (a): Distance matrix for protein 1HLB. Larger distances a denoted with a lighter color. Middle (b), Right (c): Example 2D decompositions for protein 1HLB. The figure on the left (b) represents a 2nd level Haar decomposition. The image on the right (c) corresponds to a Haar decomposition of level 4. The matrix on the left is of size 128x128. The figure in the middle contains 1024 coefficients (32x32), while the one on the right is composed of just 64 (8x8).

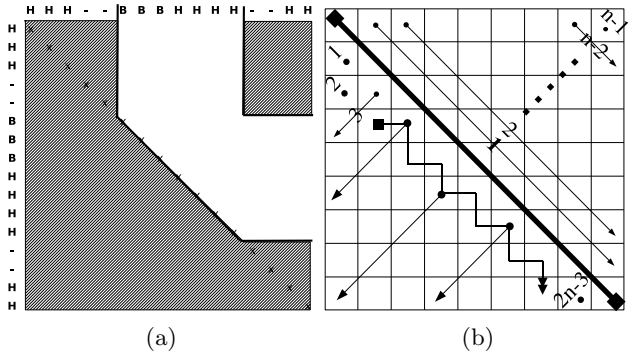

Figure 2: Left (a): Illustration of distance values selected using windowing strategy. Hatched matrix values are ignored. Right (b): Two methods for converting a distance matrix to a one-dimensional signal. The parallel conversion process is shown in the upper triangle, the anti-parallel version in the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

subsequence matches

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. We have recently shown that this strategy can be used in conjunction with decision trees to effectively classify protein structures [15]. 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 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 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. We apply a 5th-level 1D Haar decomposition to each signal and resample the final level of approximation coefficients to contain 10 values. We concatenate the parallel and anti-parallel signals and use the resulting 20 attributes as our *local descriptor*.

3.3 Potential Applications

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. Range queries are 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 that have a distance to the query of less than

r. In both cases, distance refers to the Euclidean distance 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 in the dataset.

4. DATASETS

To evaluate our representations, we conduct experiments on three different datasets, comparing results with two of the more recently-published techniques [4, 10]. In order to make a valid comparison, we try to use the same 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 two contains 200 proteins having less than 40% sequence homology and were taken from version 1.59 of the SCOP database. 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. We were unable to completely process 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 (again having less than 40% sequence homology). 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, 8], most recently by Gao and Zaki [10]. 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. We could not match 22 proteins, leaving us with a total of 1798. Please note that, approximately 1600 proteins from this dataset, which we call the *2K* set, are present in the *33K* set.

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 this section 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 using a 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 recursively repeated until the data is divided into mutually exclusive subsets. While a binary search tree uses the same key on all nodes on 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 existing approaches, namely PSIST and Protdex2. We were able to obtain the source code for PSIST[5], but were unable to do so for Protdex2. Thus, we evaluate our approach against the results returned by PSIST and where appropriate, the published results for Protdex2.

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, PSIST required more memory than was available on our machine (> 1.5 GB), and we were forced to abort the tests. 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 different parameter values reported in the original work [10], we evaluated the performance of the algorithm on the 2K dataset. We found that the default settings yielded an accuracy equivalent to those reported as the "best," along with a drastic reduction in running time. 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 that 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 [10]. 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, the PSIST code only provides accuracy values for Class and SuperFamily so we do not have results for the Fold and Family levels.

Table 1 shows the classification accuracy for our representa-

[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.

Table 1: Classification accuracy for PSIST and the 2D representation on the 2K dataset at different levels of the SCOP hierarchy. Results are given for the 2D representation on the 33K dataset. Accuracy is given as a percentage of correct classification.

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

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

tion 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 through the database. Our results on the 2K dataset are equal to those of PSIST at the Class level and almost 3% higher at the SuperFamily level. 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 classes or categories in the 33K dataset[6]. Table 2 provides the number of unique classes for the 2K and 33K datasets for the different levels of the SCOP hierarchy. 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 [10]. 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.

[6]Here we refer to "class" in the traditional classification sense. We use Class when referring to the SCOP level of the same name.

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

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

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

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 return 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 for three different range values (25, 50 and 125) on the 2K and 33K datasets. 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 5.5 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.*

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.

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

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 structure 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. 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 target database, we repeat our classification experiments, retrieving 3 nearest neighbors and comparing the SCOP labels to determine accuracy. We provide information on the number members of 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 Table 6. *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 (Table 5), we were able to choose the correct one in almost every case.

5.5 Performance

In this section we provide an analysis of query execution time and memory usage of the tested methods. For PSIST, we use our own measurements. With Protdex2, we rely on the information provided in the original paper.

Table 6: Accuracy for three different substructure windows at the Family level. Values represent percentage of correct classification on a 3-NN classifier.

Window	Family	Accuracy
HHHHHHHHHH	56113	100
	46463	100
BBBBBBBBBB	51534	75
	53851	100
$--BBBBBB--$	56113	100
	51751	100

Table 7: Execution time (in seconds) for various database and KNN query workloads ($k = 100$)

Database Size	Number of Queries	Total Time	Query Time	Time per Query
33,010	33,010	160	135	0.004
33,010	107	8	3	0.03
1798 - 2D	181	2	< 1	< 0.006
1798 - PSIST	181	107	92	0.51

Query Execution Time

Here we provide results on the running time for our KNN experiments where k equals 100, the longest of all our retrieval tests. We list these values in Table 7. 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* gives just the time spent conducting the queries.

On the 33K dataset, it was reported that the Protdex2 algorithm required a total of 14 minutes to execute 108 queries on a computer with a 1.6 GHz Pentium 4 and 256 MB of RAM [4]. The results in an average query time of almost 8 seconds (note: this was a different machine than our testbed). In our experiments, on the 2K dataset, PSIST required roughly 0.51 seconds per query. On the same machine, our method requires only 3 seconds to execute *all* queries, *a 168-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.

Memory Usage

In addition, our representation is much less memory-intensive than PSIST and at least comparable to Protdex2. Memory usage is not reported in the Protdex2 results, but since the test system had only 256 MB of main memory, one can assume that to be an upper bound. 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. Our wavelet representation coupled with the k-d tree requires just 80 MB to process the same dataset. 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

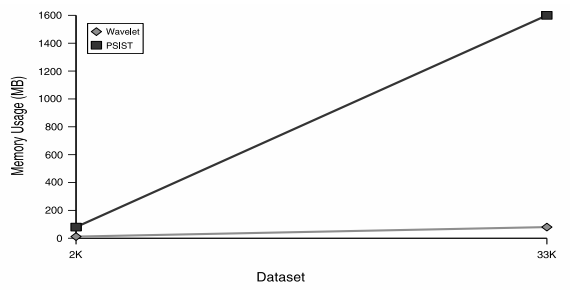

Figure 3: Memory usage (in MB) of the 2D wavelet approach versus PSIST on the 2K and 33K datasets.

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 3 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.

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.

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 [19] 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. Two more potential applications include defect tracking in molecular dynamics (MD) simulations and the modeling of protein folding pathways [12,16]. 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 others. 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 a tremendous 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.

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*, pp. 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 *PSB 2004*, vol. 9, pp. 264–275. 2004.

[7] T. Can and Y. Wang. Ctss: A robust and efficient method for protein structure alignment based on local geometrical and biological features. In *IEEE CSB 2003*, pp. 169–179. IEEE, 2003.

[8] O. Çamoğlu, T. Kahveci, and A. Singh. Towards index-based similarity search for protein structure databases. In *IEEE CSB 2003)*, pp. 148–158, 2003.

[9] P.-H. Chi, G. Scott, and C.-R. Shyu. A fast protein structure retrieval system using image-based distance matrices and multidimensional index. In *IEEE BIBE 2004*. IEEE, 2004.

[10] F. Gao and M. Zaki. Psist: Indexing protein structures using suffix trees. In *IEEE CSB 2005*, August 2005. IEEE.

[11] L. Hammel and J. Patel. Searching on the secondary structure of protein sequences. In *VLDB*, pp. 634–645, 2002.

[12] 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. *Computational Genomics*. 2002.

[13] T. Madej, J.-F. Gibrat, and S. Bryant. Threading a database of protein cores. *Protein Struct. Funct. Genet.*, pp. 356–369, 1995.

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

[15] K. Marsolo and S. Parthasarathy. Alternate representation of distance matrices for characterization of protein structure. In *IEEE ICDM 2005)*, 2005.

[16] 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 *SIAM 2005*, 2005.

[17] 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.

[18] K.-S. Oh and A. Makinouchi. Image classification and retrieval based on wavelet-som. In *DANTE 1999*, 1999.

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

[20] Z. Tan and A. K. H. Tung. Clustering substructure from sequential 3d protein database. In *ICDE 2004*, Boston, 2004.

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

[22] J. Wang, J. Li, and G. Wiederhold. Simplicity: Semantics-sensitive integrated matching for picture libraries. *IEEE TPAMI*, 23(9):947–963, 2001.