A Hybrid Approach for Indexing and Searching Protein Structures

Tarek F. Gharib

Faculty of Computer and Information Sciences, Ain Shams University, Cairo 11566, Egypt
E-mail: tgharib@eun.eg

Abstract: - Searching for structural similarities of proteins has a central role in bioinformatics. Most tasks of bioinformatics depends on investigating the homologous protein's sequence or structure these tasks vary from predicting the protein structure to determine sites in protein structure where drug can be attached. Protein structure comparison problem is extremely important in many tasks. It can be used for determining function of protein, for clustering a given set of proteins by their structure, for assessment in protein fold prediction. Protein Structure Indexing using Suffix Array and Wavelet (PSISAW) is a hybrid approach that provides the ability to retrieve similarities of proteins based on their structures. Indexing the protein structure is one approach of searching for protein similarities. The suffix arrays are used to index protein structure and the wavelet is used to compress the indexed database. Compressing the indexed database is supposed to make the searching time faster and memory usage lower but it affects the accuracy with accepted rate of error.The experimental results, which are based on the structural classification of proteins (SCOP) dataset, show that the proposed approach outperforms existing similar techniques in memory utilization and searching speed. The results show an enhancement in the memory usage with factor 50%.

Key-Words: - protein structures, indexing, suffix array, wavelet

1 Introduction

Searching for structural similarities has a critical role in many applications like prediction of protein's structure and functions, classification of proteins and drug design and discovery. Proteins with homologous sequence or structure can be concluded to have a common ancestor which is helpful for better understanding of life tree. There have been several methods proposed to compare protein structures and measure the degree of structural similarity between them. These methods have been based on alignment of secondary structure elements as well as alignment of intra and inter-molecular atomic distances [5].

The following are some of the reasons why the structure comparison problem is also extremely important [7]:

1. For determining function: The function of a new protein can be determined by comparing its structure to some known ones. That is, given a set of proteins whose fold has already been determined and whose function is known, if a new one has a fold highly similar to a known one, then its function will similar as well. This type of problems implies the design of search algorithm for 3D databases, where a match must be based on structure similarity. Analogous problems have already been studied in Computational Geometry and Computer Vision, where a geometric form or object has to be identified by comparing it to a set of known ones.
2. For clustering: Given a set of proteins and their structures, we may want to cluster them in families based on structure similarity. Furthermore, we may want to identify a consensus structure for each family. In this case, we would have to solve a multiple structure alignment problem.
3. For assessment of fold Predictions: The Model Assessment Problem is the following: Given a set of “tentative” folds for a protein, and a “correct” one (determined experimentally), which of the guesses is the closest to the true? This is, e.g., the problem faced by the CASP (Critical Assessment of Structure Prediction) jurors, in a biannual competition where many research groups try to predict protein structure from sequence. The large number of predictions submitted

(more than 10,000) make the design of sound algorithms for structure comparison a compelling need. In particular, such algorithms are at the base of CAFASP, a recent Fully Automated CASP variant.

The rapid growth of the Protein Databank (PDB) current holdings, > 50000 proteins at the first quarter of 2009, raises the need for new tools that perform proteins similarity searching to clarify the similarities in the three dimensional structures between related or similar proteins.

Searching the protein structure has another problem, besides the rapidly growing rate of proteins in PDB, which is the complexity. The protein structure alignment is a NP-hard problem. Many methods were proposed to solve this problem.

This problem is approached by different three approaches they are:

1. Pair-wise structure-based alignment
2. Multiple structure-based alignment
3. Database indexing

Pair-wise structure alignment methods can be classified into three classes [8]. The first class works at the residue level [9, 10]. The second class focuses on using secondary structure elements (SSEs) such as alpha-helices and beta-strands to align two proteins approximately [11, 12, and 13]. The third approach is to use geometric hashing, which can be applied at both the residue [14] and SSE level [9].

The combinatorial extension (CE) method is an example of pair-wise approach. It breaks each structure in the query set into a series of fragments that it then attempts to reassemble into a complete alignment. A series of pairwise combinations of fragments called aligned fragment pairs, or AFPs, are used to define a similarity matrix through which an optimal path is generated to identify the final alignment. Only AFPs that meet given criteria for local similarity are included in the matrix as a means of reducing the necessary search space and thereby increasing efficiency [10]. A number of similarity metrics are possible; the original definition of the CE method included only structural superposition and inter-residue distances but has since been expanded to include local environmental properties such as secondary structure, solvent exposure, hydrogen-bonding patterns, and dihedral angles [10]. An alignment path is calculated as the optimal path through the similarity matrix by linearly progressing through the sequences and extending the alignment with the next possible high-scoring AFP pair. The initial AFP pair that nucleates the alignment can occur at any point in the sequence matrix. Extensions then proceed with the next AFP that meets given distance criteria restricting the alignment to low gap sizes. The size of each AFP and the maximum gap size are required input parameters but are usually set to empirically determined values of 8 and 30 respectively [10].

The SSAP (Sequential Structure Alignment Program) method uses double dynamic programming to produce a structural alignment based on atom-to-atom vectors in structure space. SSAP algorithm is an example for the second class of pairwise protein structure alignment which aligns proteins at SSE level by using α helices and β strands. Instead of the alpha carbons typically used in structural alignment, SSAP constructs its vectors from the beta carbons for all residues except glycine, a method which thus takes into account the rotameric state of each residue as well as its location along the backbone.

SSAP works by first constructing a series of inter-residue distance vectors between each residue and its nearest non-contiguous neighbors on each protein. A series of matrices are then constructed containing the vector differences between neighbors for each pair of residues for which vectors were constructed. Dynamic programming applied to each resulting matrix determines a series of optimal local alignments which are then summed into a "summary" matrix to which dynamic programming is applied again to determine the overall structural alignment [13].

The second approach for solving the problem is multiple structure alignments. This approach is based on geometric hashing [16], or SSE information [15]. The pair-wise and multiple structure alignment approaches are not suitable for searching for similarity over thousands of protein structures. Database indexing and scalable searching approaches satisfy this requirement.

The definition of the Multiple Structural Alignment problem is not a straightforward issue. Given m input molecules, should all m molecules participate in the alignment or only a subset of the m molecules? Certainly, we wish to detect the best subsets which give a good multiple structure alignments. Consider an example where among 100 input molecules there are 40 structurally similar molecules from family “A”, 50 structurally similar molecules from family “B” and additional 10 molecules, which are structurally dissimilar to any other molecule in the input. Naturally, we require from a Multiple Structural Alignment algorithm to detect simultaneously the similarity between the 40 molecules from family "A" and between the 50 molecules from family "B". Also, there might be only a sub-structure (motif, domain) that is similar between some molecules. Thus, partial similarities between the input molecules should also be reported by the Multiple Structural Alignment algorithm [20].

Obviously, the number of all possible solutions could be exponential in the number of input molecules. For example, consider proteins which contain α-helices. Each pair of α-helices could be structurally aligned (at least partially, if they are different in their lengths). Any combination of α-helices from different molecules gives us some solution to the Multiple Structural Alignment problem. The number of such combinations is exponential. Thus, it is not practical to output (even if the algorithm is capable to detect) all possible solutions [20].

Multiple Protein Structural Alignment (MltiPort) method is an example of the Multiple Structural Alignment approach. It is based on the pivoting technique, i.e. there is a pivot molecule that has to participate in all the alignments. In other words, the rest of the molecules are aligned with respect to the pivot molecule. In order not to be dependent on the choice of the pivot, we iteratively choose every molecule to be the pivot one.

The goal of the Multiple Structural Alignment with Pivot algorithm is to detect the largest structural cores between the input molecules with respect to the pivot molecule. The algorithm requires that the pivot molecule participates in the multiple alignments, but it does not require that all the input molecules from the set S' are included in the multiple alignment. Thus, the method detects the best partial multiple alignments [21].

The third approach for solving the problem is protein structure indexing according to the representation of the local features. Local features can be extracted at residue level [19] or SSEs level to approximate the structure of the protein [18, 4].

Due to their time complexity, the pair-wise and multiple structure alignment approaches are not suitable for searching for similarity over thousands of protein structures. Database indexing and scalable searching approaches meet the online searching requirement [2].

ProGreSS algorithm is an example of this approach. It extracts the features for both the structure and sequence, within a sliding window over the backbone. The algorithm extracts a number of feature vectors on sequence and structure components of each protein in the database by sliding a window. Each feature vector maps to a point in a multi-dimensional space. Thus, a protein is represented by a number of points. This multi-dimensional space consists of orthogonal dimensions for sequence and structure. Later, the algorithm partitions the space with the help of a grid and indexes these points using Minimum Bounding Rectangles (MBRs).

Given a query, the search method runs in three phases:

Phase 1 (index search): Feature vectors (i.e., points) are extracted from the query protein. For each of these query points, all the database points that are within εq and εt distance along the sequence and the structure dimensions are found using the index structure. Each such point casts a vote for the protein to which it belongs as in geometric hashing [22].

Phase 2 (statistical significance): For each database protein, a statistical significance value is computed based on the votes it obtained in Phase 1 and its length.

Phase 3 (post-processing): The top c proteins of highest significance are selected, where c is a predefined cutoff. The optimal pair-wise alignment of these c proteins to the query protein is then computed using the SW technique. Finally, the Cα atoms of the matching residues are super-positioned using the least-squares method [23], to find the optimal RMSD (Root Mean Square Distance).

It is a fast, novel protein indexing method called PSIST (which stands for Protein Structure Indexing using Suffix Trees). As the name implies, the new approach transforms the local structural information of a protein into a sequence, on which a suffix tree is built for fast matches. The algorithm first extracts local structural feature vectors using a sliding a window along the backbone. For a pair of residues, the distance between their Cα atoms and the angle between the planes formed by Cα, N and C atoms of each residue are calculated. The feature vectors for a given window include all the distances and angles between the first residue and the rest of the residues within the window. Compared with the local features from a single residue, our feature vectors contain both the translational and rotational information. After normalizing the feature vectors, the protein structure is converted to a sequence (called the structure-feature sequence or SF-sequence) of discredited symbols.

For a given query, all the maximal matches are retrieved from the suffix tree and chained into alignments using dynamic programming. The top proteins with the highest alignment scores are finally selected. The results show classification accuracy up to 97.8% and 99.4% at the super-family and class level according to the SCOP classification, and show that on average 7.49 out of 10 proteins from the same super-family are obtained among the top n matches. These results are competitive with the best previous methods [2].

Protein structure index (PSI) method prunes unpromising protein for the given protein query. It is based on extracting feature vector for each protein in database then indexing it using the R* tree. R* trees are used to prune the search space to be used by VAST structural alignment algorithm, this reduction in search space resulting in reduce the searching time [18].

The construction of index structure in this algorithm is proceeds in four steps:

1. SSE approximation.
2. Triplet construction.
3. Feature vector extraction.
4. Multidimensional index structure construction.

Let D = {a1, a2, … , ad} be the set of protein structures in the dataset.

SSE approximation is the first step in this algorithm. Let a belongs to D be a protein structure, where Sa = {s1,s2, …, sna} is the SSEs of a. let Rsi = {ri,1, ri,2, …, ri,k}be the residues which constitute si. The algorithm splits the Rsi into two equal sized sets Rsi1 and Rsi2. C1 and c2 are defined as the centers of mass of the residues in R-si1 and Rsi2. A line segment approximation to si is achieved by extending the line segment [c1, c2] by half of the Euclidean distance between c1 and c2 in both directions.

PSISA algorithm is used in searching for protein structural similarities. The algorithm follows the indexed database approach to solve the problem of searching protein structure. Generalized Suffix Array was used as the indexing structure; it is a novel usage of this data structure to approach this problem. The algorithm input is a set or all of the known structure proteins which are used to build the indexed database, Generalized Suffix Array. A query protein(s) is given to build another generalized suffix array, which is compared with indexed database to find out all similar structure proteins.

FASTA algorithm and Smith-Waterman algorithm are used as a final ranking step. This step ranks the found proteins from the above mentioned matching step between indexed database and query generalized suffix array. The proteins are listed according to how close they are similar in structure to query protein. The memory usage of PSISA algorithm outperforms all previous algorithms.

In this paper, we present a hybrid approach for indexing the protein structure. The proposed approach based on PSISA algorithm [1, 17] and PSIST algorithm [2]. The proposed approach uses the PSIST approach for extracting the feature vector. And the suffix array structure used to index data as in PSISA. The main contribution of the proposed approach is that it investigates the usage of wavelet in purpose of compressing the indexed database. Therefore, the proposed approach decreases the searching time required for queries.

Given a protein as a query, the generalized suffix array is searched to find all the proteins that have matching length greater than or equal to a certain threshold. These proteins are ranked according to the similarity to the query protein.

2 Wavelet Background

A wavelet is a mathematical function used to divide a given function or continuous-time signal into different frequency components and study each component with a resolution that matches its scale. The wavelet transform is relatively new (early 80s) and has some similarities with the Fourier transform. Wavelets differ from Fourier methods in that they allow the localization of a signal in both time and frequency.

The fundamental idea behind wavelets is to analyze according to scale. A wavelet transform is the representation of a function by wavelets. The wavelets are scaled and translated copies (known as "daughter wavelets") of a finite-length or fast-decaying oscillating waveform (known as the "mother wavelet (ψ)") [24, 25]. Wavelet transforms have advantages over traditional Fourier transforms for representing functions that have discontinuities and sharp peaks, and for accurately deconstructing and reconstructing finite, non-periodic and/or non-stationary signals. Wavelets are functions that satisfy certain mathematical requirements and are used in representing data or other functions. If we look at a signal with a large “window" we would notice gross features. Similarly, if we look at a signal with a small “window" we would notice small features. The result in wavelet analysis is to see both the forest and the trees, so to speak.

The scaling function of wavelet produces a smoother version of the data set, which is half the size of the input data set. Wavelet algorithms are recursive and the smoothed data becomes the input for the next step of the wavelet transform. The simplest example of a mother wavelet is the Haar basis, Equation (1) is the Haar wavelet scaling function:

$$a_i = \frac{s_i + s_{i+1}}{2} \quad (1)$$

where a_i is a smoothed value and data sample s_i, s_{i+1} are data samples. The Haar transform preserves the average in the smoothed values. This is not true of all wavelet transforms.

In digital signal processing terms, the wavelet function is a high pass filter. A high pass filter allows the high frequency components of a signal through while suppressing the low frequency components. For example, the differences that are captured by the Haar wavelet function represent high frequency change between an odd and an even value.

In digital signal processing terms, the scaling function is a low pass filter. A low pass filter suppresses the high frequency components of a signal and allows the low frequency components through. The Haar scaling function calculates the average of an even and an odd element, which results in a smoother, low pass signal.

The wavelet transform have two forms continuous and discrete. The continuous wavelet transform, (CWT) for a function f (t) is defined by equation (2):

$$CWT\ (f,a,b) = a^{-1/2} \int_{-\infty}^{\infty} f(t)\psi\left(\frac{t-b}{a}\right)dt \quad (2)$$

where a (the scale parameter) > 0, b (the translation parameter). The continuous wavelet transform maps a one-dimensional signal to a two-dimensional time-scale joint representation. It is calculated by continuously shifting a continuously scalable function over a signal and calculating the correlation between the two.

A wavelet decomposes a signal into several groups (vectors) of coefficients. Different coefficient vectors contain information about characteristics of the sequence at different scales. Coefficients at coarse scales capture gross and global features of the signal while coefficients at fine scales contain local details. The discrete wavelet transform is an economical way to compute the wavelet, because it is computed only on a dyadic grid of points, where the subsampling is at a different rate for different scales. The discrete wavelet transform is commonly introduced using a matrix or a computational form. In matrix form we can represent the discrete wavelet transform [24] through an orthogonal matrix.

The discrete wavelet transform is computed by successive lowpass and highpass filtering of the discrete time-domain signal as shown in fig. 1. This is called the Mallat algorithm or Mallat-tree decomposition. Its significance is in the manner it connects the continuous-time multi-resolution to discrete-time filters. In the figure, the signal is denoted by the sequence x[n], where n is an integer. The low pass filter is denoted by G0 while the high pass filter is denoted by H0. At each level, the high pass filter produces detail information; d[n], while the low pass filter associated with scaling function produces coarse approximations, a[n].

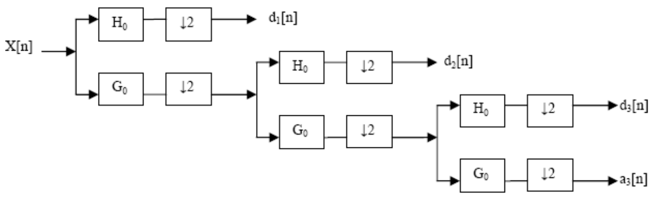

Fig. 1 three level wavelet decomposition trees

At each decomposition level, the half band filters produce signals spanning only half the frequency band. This doubles the frequency resolution as the uncertainty in frequency is reduced by half.

There are a number of basis functions that can be used as the mother wavelet for Wavelet Transformation. Since the mother wavelet produces all wavelet functions used in the transformation through translation and scaling, it determines the characteristics of the resulting Wavelet Transform. Therefore, the details of the particular application should be taken into account and the appropriate mother wavelet should be chosen in order to use the Wavelet Transform effectively.

Haar wavelet is one of the oldest and simplest wavelet. Therefore, any discussion of wavelets starts with the Haar wavelet. Daubechies wavelets are the most popular wavelets. They represent the foundations of wavelet signal processing and are used in numerous applications. These are also called Maxflat wavelets as their frequency responses have maximum flatness at frequencies 0 and π. This is a very desirable property in some applications. The Haar, Daubechies, Symlets and Coiflets are compactly supported orthogonal wavelets. These wavelets along with Meyer wavelets are capable of perfect reconstruction. The Meyer, Morlet and Mexican Hat wavelets are symmetric in shape. The wavelets are chosen based on their shape and their ability to analyze the signal in a particular application.

3 Methodology

In this paper, we present three different implementations for the Haar transform. Wavelet compression is a form of data compression well suited for image compression (sometimes also video compression and audio compression). The goal is to use this compression method for compressing the data which present proteins.

For extracting the local feature vector we used the same method used in the PSIST. Considering using the sliding window technique, we slide the window through the backbone of the protein, where the window size (w) presents the number of residues in the window. The feature vector contains 2 * (w-1) values, these values describe the relation between the first residue and other residues in the window. For the first residue in the window and each other residue we calculate two values, they are the Euclidean distances between the Cα in the two residues which presents the distance between these two residues. The second value is the angle between these two planes containing these two residues.

Each protein is presented by a 2D array of varied number of columns depends on the number of amino acids comprising protein and even number of rows depends on the window size. The three methods are based on compressing the 2D data of the array which presents proteins to half the size. The different between every each method is based in how the data are divided in four sub-arrays.

The first column in the array contains data for the first position of the sliding window; the second column in the array contains data for the second position of the sliding window and so on. The number of items in each column must be even number.

The first method compress data as follow: it takes each pair of data in each column and present it by one value which is the average of the pair values.

So data compressed to half the size. Fig. 2 depicts the data which averaged by first method.

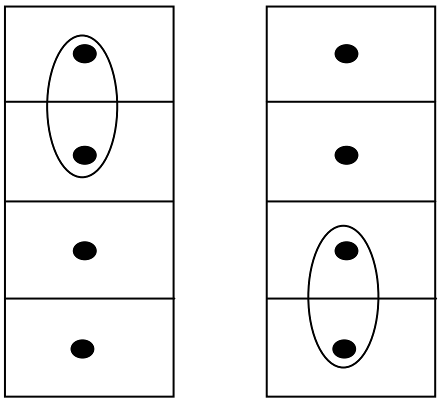

Fig. 2 first method of compression

The second method compress data as follow: it takes the first element of the first column and the corresponding value in the next column and calculates the average of the pair values. So data compressed to half the size. In other words this method use the column with odd index and the next column with even index, by taking just two column at once to compress data by compressing pairs of element in the odd column and the corresponding element in the next even index column. Fig. 3 depicts the data which averaged by the second method.

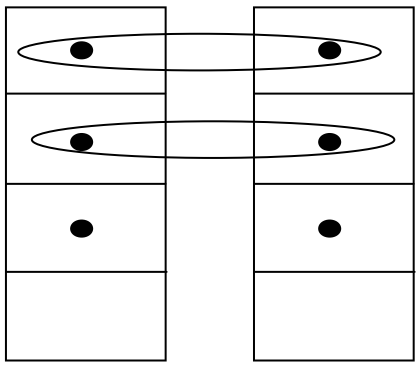

Fig. 3 second method of compression

The third method compress data as follow: it takes two columns one with even index and next column with odd index then we calculate the average of data for each four adjacent elements. Two columns with even number of elements can be divided into number of squares equals half of the number of elements in the two columns. Each square of data is presented by the average of all elements in this square, the four elements. Fig.4 depicts the data which averaged by the third method.

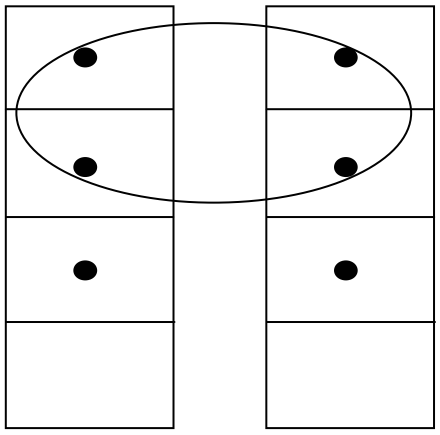

Fig. 4 third method of compression

```
INPUT: FeatureVectorList
OUTUPT: CompressedFeatureVectorList

FOR I = 0 TO FeatureVectorList.size()-1 DO

Create newFeatureVector

FOR J = 0 TO
FeatureVectorList.getVector(i).size DO

    1. Newfaturevector.add(
       FeatureVectorList.getVector(i).getComp
       onent(j) + FeatureVectorList.getVector(i
       + 1).getComponent(j)

ENDFOR
    1. Replace ith FeatureVector with
       newFeatureVector
    2. Delete (i +1)th featurevector
ENDFOR
```

Fig. 5 pseudo-code for the second method

Fig. 5 state the algorithm of the second method. We just state the second method because it gives the best results in accuracy meanwhile all of the three methods are equal in memory usage and query processing speed. The method input is “FeatureVectorList” which is a list of feature vectors. Each feature vector contains even number of values. Method starts by a loop “outer for” to read all the input feature vectors. Each new iteration starts by creating “newFeatureVector” which is a new feature vector to hold the new values. Then the inner loop starts to calculate the average of the first component of ith feature vector and the corresponding component of (i + 1)th feature vector. The average value is stored in the newFeatureVecotr. The process of calculating the average of the jth component of both ith and (i+1)th vectors is repeated till the last component in both vectors. Final to steps in the outer loop is to replace the ith feature vector with the newFeatureVector and delete the (i + 1)th feature vector. The result is replacing two feature vectors with one feature vector. The compression ratio is exactly the half size.

```
INPUT        :  qSA , l
OUTPUT       :  list of matched proteins MList
dbSA, l)
j = 1
FOR i = 1 TO n DO
   Clear querylist
   Querylist. Add ( i)
   queryFirstSymbol = qSA[i].firstsymbol
   i = i + 1
   WHILE qSA[i].firstsymbol = qSA[i-1].firstsymbol DO
      Querylist.add(i)
      i = i + 1
   ENDWHILE
   WHILE dbSA[j].firstsymbol < queryFirstSymbol DO
      j = j + 1
   ENDWHILE
   IF dbSA[j].firstsymbol > queryFirstSymbol THEN
        GOTO next FOR iteration
   ENDIF
   WHILE !querylist.empty() AND j < m DO
    FOR K = 0 TO querylist.length DO
      matchedLength = MaximalMatchesSearch (
                     dbSA[j] , qSA[querylist.get(K)] )
      IF matchedLength >= l THEN
        Foreach proteinNo $P_q$ in qSA[querylist.get(K)] Do
        Foreach proteinNo $P_{db}$ in dbSA[j] DO
            Add($P_q$, $P_{db}$, matchedLength) to Mlist
      ELSE
         IF qSA[querylist.get(K)].FirstMismatchedsymbol
            < dbSA[j]. FirstMismatchedsymbol THEN
            Querylist.reomve(K)
        ELSE
            Exit FOR
        ENDIF
     ENDIF
    ENDFOR
    j = j + 1
   ENDWHILE
ENDFOR
```

Fig. 6 pseudo-code for searching algorithm

In searching for similarity algorithm we consider that the query GSA length is n and the database GSA length is m, where m >> n. qSA is the query suffix array, dbSA is the database suffix array, and l is threshold of the maximal matching length. i is the qSA iterator and j is the dbSA iterator. MList is a list that contains all matched proteins resulting from the search process with length of the maximal matching. MaximalMatchesSearch is a function that matches two suffixes one form qSA and the other from dbSA; it starts matching form the first symbol in both suffixes and finally returns the number of successive matched symbols. It stops matching with the first mismatched symbol.

Fig. 6 states the searching algorithm; we scan the dbSA and qSA elements only one time. The searching process starts by grouping the qSA elements into groups according to the first symbol of these elements. Each group is presented by a list called query list. Working on these lists is sequential so we prepare the first list and match it to some elements from dbSA then we prepare the second list and match it too the same way till we finish all elements in the qSA. So the qSA is scanned only one time.

Having the first query list prepared from qSA as described above, we start matching all the list elements with the first element in the dbSA, supposed its index is dbSA iterator, starting with the same symbol as query list elements. Then we increase the dbSA iterator by one to decide if that new element starts with same symbol as elements of the query list or not. We stop increasing the dbSA iterator when we reach to dbSA element with first symbol logographically greater than the first symbol of query list elements. The reason for stopping the matching process that we know both the qSA, in turn query list, and dbSA are logographically ordered. For example if we prepared the query list of all qSA elements that starting with symbol 'A', so we will match list elements with all elements of dbSA appears before the first dbSA element that starting with symbol 'B'. Since the matching length between any element of the query list and the first element of dbSA starting with symbol 'B' is zero which is less than l and that is true for all dbSA elements come after this element, the one starting by symbol 'B'.

After stopping matching because of the reasons we stated above, a new query list is prepared from qSA and matching will start not from the beginning of dbSA but from the dbSA element where the previous step stopped at. For example, if matching of the query list of elements starting with symbol 'A' is finished, then we prepare a new query list that contains elements starting by symbol 'B' and the matching process will start from the first element of dbSA that starting by symbol 'B', the one where we stopped at from the previous step.

4 Experimental Results and Discussion

Many databases that hold information about the protein exist nowadays; some of them provide information about the protein sequence only. Others provide information about the secondary and tertiary structure, but the widely used databases are for the secondary structure since the functions of protein can predicted from its structure better than its sequence.

Genbank, the National Institute of Health (NIH), is built by National Center for Biotechnology Information (NCBI), SWISS-PORT, Protein Information Resource (PIR), Protein Databank (PDB), and Structural Classification Of Protein (SCOP), are some of long list of organizations that produce database for protein.

SCOP database is a classified version of PDB, scientist manually, by visual inspection, classify the data in the PDB files they first clustered the protein with similar domains then producing a set of families then families are grouped into super-families in turn it grouped into folds. SCOP now has 7 domains applying this classification on each domain.

The experiment is based on SCOP database. Since SCOP classifies data in classes of similar structure proteins. SCOP was used as reference for measuring the accuracy of the proposed algorithm for retrieving correct proteins. The algorithm was implemented using java. The experiments were performed on a PC with core duo 1.86GHz processor and 1 GB RAM on window Vista XP SP2 edition.

The experiment used a prepared dataset D_{DS} which consisted of 181 * 10 = 1810 protein structures. These protein structures were collected from different 181 superfamilies. Super-families belongs to domains which are α, all β, α + β and α / β. The dataset of query proteins, D_Q, was obtained by choosing a protein structure at random from each superfamily. The results shown below for PSISAW are the results of the second method explained in the previous section, since the second method gives the best performance.

Table 1 Running Time for query

Algorithm	Algorithm query time in second
PSISA	2.49
PSIST	1.5
PAST	0.44
PSISAW	0.6

Table 2 Searching time for different Algorithms

Algorithm	Searching time in second
PSISA	0.45
PSIST	0.68
PAST	0.44
PISSAW	0.43

Table 1 shows the query average time for PSIST algorithm which is 1.5seconds. Meanwhile it is for PSISA is 2.5 seconds that is means that PSIST is faster than PSISA by 1.55 times approximately. We notice that the average query time of PSISAW is 0.6 second. Because of the usage of the Wavelet as a compression technique, preprocessing step, with PSISA the average query time is reduced to more than half of PSIST time. PAST average time is 0.44 seconds, since PAST uses a simple feature extractor technique which does not consume a lot of time as previous algorithms. Table 2 shows the comparison of searching time without calculating the pre and post-processing steps. Table 2 shows that PSISA searching time is 0.45 second and PSIST time is 0.68, we can conclude that PSISA is faster than PSIST with 1.5 times. PSISAW searching time is 043 seconds. Finally we conclude that PSISAW is faster than other algorithms because the PSISW search a smaller suffix array than PSISA in memory size.

Table 3 the memory usage of algorithms

Algorithm	Algorithm Memory usage in MB
PSIST	111
PSISA	61
PAST	58
PSISAW	31

Table 3 shows the memory usage for PSISA, PSIST and PSISAW. The table shows that PSISA use approximately half the memory size used by PSIST. PAST algorithm memory consumption is 58 MB. The memory used by PSISAW is 31 MB. We conclude that memory usage for PSISAW reduces to the half memory size of PSISA approximately because of the uses of wavelet as a compression, preprocessing, step for PSISA algorithm.

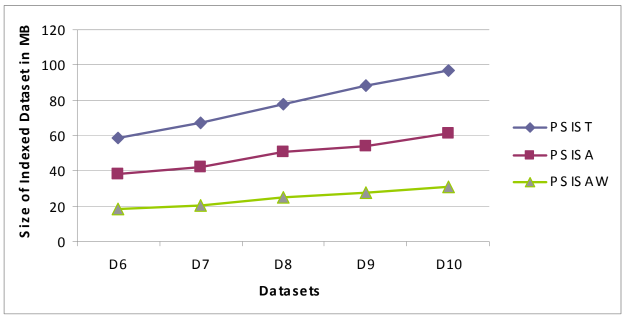

Fig. 7 Memory usage comparison

Fig. 7 depicts the memory usage for PSIST, PSISA and PSISAW for different datasets. We prepared 5 datasets differ in size for the experiment D_6, D_7, D_8, D_9 and D_{10}. All the datasets contain proteins from different 181 superfamilies from the four classes. considering D_n dataset , n presents the number of the proteins which are taken from each superfamily, so the size of each dataset is n multiplied by 181, the number of superfamilies, for example the D8 dataset contains 1448 (8 * 181) proteins.

Table 4 Accuracy of the algorithms

Algorithm	SUPERFAMILY	CLASS
PSIST	96.6%	98.3%
PSISA	96.6%	98.3%
PAST	100%	48%
PSISAW	82.5%	86%

Table 4 shows the accuracy of the proposed algorithm PSISAW comparing with PSIST, PSISA and PAST algorithms [4]. The proposed algorithm outperforms the PAST algorithm in accuracy meanwhile it gives accuracy less than PSISA and PSIST because of the uses the lossy compression technique as the preprocessing step for PSISA.

4 Conclusion

In this paper, we have presented a hybrid approach that provides the ability to retrieve similarities of proteins based on their structures. We have used the wavelet to compress the indexed data. As a result the accuracy is less than the best known accuracy by PSIST with 10% meanwhile the speed is enhanced five times. The experiment results show that our proposed algorithm outperforms the PSISA in memory usage with factor exceeds 50%.

References:

[1] Tarek F. Gharib, A. Salah and Abdel-Badeeh M. Salem "PSISA: an Algorithm for Indexing and Searching Protein Structure using Suffix Arrays" In The WSEAS International Conference on Computers, pages 775-780, 2008.

[2] Feng Gao, Mohammed J. Zaki, PSIST: Indexing Protein Structures using Suffix Trees, in IEEE Computational Systems Bioinformatics Conference, Palo Alto, CA, August 2005.

[3] A. Murzin, S. 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

[4] Hanjo Taubig, Arno Buchner and Jan Griebsch "PAST: fast structure-based searching in the PDB", Nucleic Acids Research, Vol. 34, p.p. w20-w23 Web Server issue, 2006.

[5] Amit P. Singh, Douglas L. Brutlag, Protein Structure Alignment: A Comparison of Methods, 2000.

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

[7] JACQUES COHEN, Bioinformatics—An Introduction for Computer Scientists, ACM Computing Surveys, Vol. 36, No. 2, pp. 122–158, June 2004.

[8] I. Eidhammer, I. Jonassen, and W. Taylor. Structure comparison and structure patterns. Journal of Computational Biology, 7(5):685.716, 2000.

[9] L. Holm and C. Sander. Protein structure comparison by alignment of distance matrices. J. Mol. Biol, 233:123.138, 1993.

[10] I. Shindyalov and P. Bourne. Protein structure alignment by incremental combinatorial extension(ce) of the optimal path. Protein Eng., 11(9):739.747, 1998.

[11] T. Madej, J. Gibrat, and S. Bryant. Threading a database of protein cores. Proteins, 23:356.369, 1995.

[12] K. Mizoguchi and N. Go. Comparison of spatial arrangements of secondary structural elements in proteins. Protein Engineering, 8:353.362, 1995.

[13] C. Orengo and W. Taylor. SSAP: Sequential structure alignment program for protein structure comparisons. Methods in Enzymol., 266:617.634, 1996.

[14] Y. Lamdan and H. Wolfson. Geometric hashing: a general and efficient model-based recognition scheme. Intl. Conf. On Computer Vision (ICCV), pages 238.249, 1988.

[15] O. Dror, H. Benyamini, R. Nussinov, and H. Wolfson. MASS: Multiple structural alignment by secondary structures. Bioinformatics, 19(12):95.104, 2003.

[16] R. Nussinov, N. Leibowit, and H. Wolfson. MUSTA: a general, efficient, automated method for multiple structure alignment and detection of common motifs: Application to proteins. J. Comp. Bio., 8(2):93.121, 2001.

[17] Tarek F. Gharib, A. Salah, I. M. El Henawy and Abdel-Badeeh M. Salem "Protein Structure Searching using Suffix Arrays" In The International Conference on Bioinformatics & Computational Biology (BIOCOMP), pages 688-691, 2008.

[18] O. C¸ amoglu, T. Kahveci, and A. Singh. Towards index based similarity search for protein structure databases. IEEE Computer Society Bioinformatics Conference (CSB), pages 148.158, 2003.

[19] A. Bhattacharya, T. Can, T. Kahveci, A. Singh, and Y.Wang. Progress: Simultaneous searching of protein databases by sequence and structure. Pacific Symp. Bioinformatics, pages 264.275, 2004.

[20] M. Shatsky, R. Nussinov, and H. Wolfson, Multiprot – a multiple protein structural alignment algorithm. Proteins, 56:143.156, 2004.

[21] A. D. McNaught and A. Wilkinson, Compendium of Chemical Terminology second edition, 1997.

[22] H. J.Wolfson and I. Rigoutsos. Geometric hashing: An introduction. IEEE Computational Science & Engineering, pp. 10–21, Oct-Dec 1997.

[23] K. S. Arun, T.S. Huang, and S.D. Blostein. Least-squares fitting of two 3-D point sets. IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-9(5)698–700, September 1987.

[24] Mallat,S.G., A theory for multiresolution signal decomposition:the wavelet representation. IEEE Trans. Pattern Analysis and Machine Intelligence, 11, 674–693, 1989.

[25] Daubechies,I. Ten Lectures on Wavelets. SIAM, Philadelphia, 1992.