Protein Structure Searching using Suffix Arrays

Tarek F. Gharib¹  A. Salah²  I. M. El Henawy²  Abdel-Badeeh M. Salem¹
¹Faculty of Computer and Information Sciences, Ain Shams University, Cairo 11566, Egypt
E-mail: tgharib@asunet.shams.edu.eg
²Faculty of Computer and Information Systems, Zagazig University, Zagazig, Egypt

Abstract - Searching for similarities of proteins using Structured-based query, has a vital role in many applications like drug discovery and drug design, disease diagnosis and treatment and protein classification. Indexing the protein structure is one approach of searching protein structure for similarities. In this paper we proposed a method to enhance the memory space for storing the indexed data without affecting other performance criteria. Our technique starts by extracting the local feature vectors of proteins structures. Normalization is applied to these vectors components. Finally we use the generalized suffix array to index these vectors. Suffix array is used to return the maximal structural similarities as a result for a structured query. The experimental results, which based on the structural classification of protein (SCOP) dataset, show that our method outperforms existing similar methods in memory usage. Our results show an enhancement in the memory usage with factor exceeds 50%.

Keywords: protein structures, indexing, suffix array

1 Introduction

The rapid growth of the Protein Databank (PDB) current holdings, > 40000 at the last quarter of 2007, 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. Most of these tools search the protein structure rather than protein sequence, which is a sequence of amino acid molecules, because of the relation between protein shape and its function, in other words, proteins that have similar functionality have similar structure besides it might have not the same primary structure [3]. But if a set of proteins have the same primary structure then they will have the same functionality. So the importance of the 3-D shape of protein comes from that the function of protein depends on its shape rather than its sequence (primary structure). Proteins Primary structure is a sequence of letters that states the amino acid in this protein. Protein Secondary structure is a 3D description of the proteins as a sequence of local segment of 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.

Searching for similarities in database is a problem approached by several ways. First it was approached by sequence alignment [9], but because of the link between protein structure and its functionality this raised the need for structural alignment. That means we can search for partial structure similarities between proteins.

Several approaches were proposed to solve this problem. Pair-wise structural alignment algorithms can perform the alignment at the secondary structure elements SSEs level or intra and inter-molecular atomic level [4]. However, pair-wise alignment is not feasible for large databases with more than few thousands of proteins; PSI belongs to this class [6]. Database searching using information retrieval techniques [7] and indexing the protein structure using suffix tree [5] are examples of approaches that don’t follow the pair-wise alignment.

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 [6].

Protein Structure Indexing using Suffix Trees (PSIST) convert the 3D structure to a sequence by extracting feature vector for each protein in the database. The feature vector includes the distance between each two residues and the angle between their plans. Each protein is described as a list of vectors. Each vector is converted to a unique symbol, that map the list of vectors to a sequence (String) that can be fit in a suffix tree which is an indexing structure that speedup the searching[5].

In this paper, we present a proposed method for indexing the protein structure. The method starts by extracting feature vectors from the protein structure so that these feature vectors are invariant to translation and rotation. Each feature vector represents one residue, its components are the two torsion angles phi and psi and the distance between the Cα atom of this residue and the Cα atom of the pervious atom. To reduce the vector space we apply the normalization. After normalizing the feature vectors we convert feature vectors to a sequence of symbols.
Generalized suffix array is used to index these sequences [10].

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.

The rest of the paper is organized as follows. Section 2 discusses the index construction on the protein structure dataset. Section 3 discusses the searching algorithm. Section 4 presents the experimental results. We end with the conclusion section.

2 Constructing the protein index

In this section we provide details about our method to index and search the protein structures. We start by providing an overview of the protein torsion angles and how to calculate them as a part of local feature extraction. Then we provide an overview of the generalized suffix array and its construction.

2.1 Extracting the local feature vector

Protein structure is a chain of residues connected by peptide bonds. The 3D structure of protein can be described using the bonds length, bonds angles and torsion angles. Both bond lengths and bonds angles are fixed at physically reasonable values, but only the torsion angles are allowed to vary. There are three bonds N-C, C-Cα and Ca-N and there are three angles N-Cα-C, N-Cαα and Ca-NαC. Torsion angle is formed by three bonds, four atoms, and considered the key to protein folding. For example if we have 4 successive atoms Ci-1, Ni, Cai and Ci, since any plane is defined by three points we can determine two planes, one includes Ci-1, Ni, Cai atoms and the other includes Ni, Cai and Ci atoms. The angle between these two planes is the torsion angle about the bond Ni-Cai, which is the common bond between them.

Protein backbone has three types of bonds N-C, C-Cα and Ca-N with three torsion angles α, ψ and φ respectively. Torsion angle α value is 0° or 180° so it has no affect in the protein structure. Torsion angles ψ and φ can vary in values, so they have a great influence in the protein structure. They are the torsion angles of the Caα-Ci bond and Na-Cai bond respectively.

Each protein is described as a set of feature vectors; each one contains the torsion angles ψ and φ for the two bonds going from atom Caα and the Euclidian distance between the two atoms Caα and Caα+1. These features are used because they are invariant to rotation and translation.

The Euclidian distance can be calculated from the values of the three coordinates provided for each atom in the PDB file. The torsion angles are calculated by calculating the normal vector for the two planes and then calculating the angle between these two normal vectors. For example to calculate the φ torsion angle we firstly find the normal vector of the plane that contains the three atoms Ci-1, Ni, and Caα, then we find the normal vector of the plane that contains the three atoms Ni, Caα, and Ciα, finally we calculate the angle between these two normal vectors using the following equation:

$$\cos \theta = \frac{\| \vec{n}_1 \|^2 + \| \vec{n}_2 \|^2 - \| \vec{n}_1 - \vec{n}_2 \|^2}{2 \| \vec{n}_1 \| \| \vec{n}_2 \|}$$ (1)

Dihedral angles range is [-π, π], so after calculating them using the above equations we have to check for sign by calculating torsion angle sign and then adjust it if it is needed. It is calculated by the inner product of the normal vector contains the first three atoms and the vector its points are the third atom and the last atom (Caα and Cai atoms respectively in case of φ, Cai and Ciα+1 respectively in case of ψ). The sign of the torsion angle follow the sign of the resulted vector from the inner product process.

$$\text{sign}(\phi) = \text{sign}(\text{normal vector}(C_{i-1}, N_i, C_{ai}) \cdot (C_i - C_{ai}))$$ (2)

$$\text{sign}(\psi) = \text{sign}(\text{normal vector}(N_i, C_{ai}, C_i) \cdot (N_{i+1} - C_{ai}))$$ (3)

Using our method the protein is described as a vector of vectors, each sub vector Vi contains three components <Vi,1, Vi,2, Vi,3>. Vi,1 is the φ torsion angle, Vi,2 is the ψ torsion angle and Vi,3 is the distance between Ci and Caα+1, i ≤ j ≤ n, and n is number of amino acids in the protein. The protein will be described as a set of Vi vectors, from V1 to Vn. Finally, we have a set of vectors Pj one for each protein in the dataset, each Pj holds n Vi vectors, where 1 ≤ j ≤ m, and m is the number of proteins in dataset.

2.2 Normalization

The components of feature vector have a great degree of freedom, since φ and ψ range is [-π, π] and the average length range of the virtual bond between any two successive Ca atoms is [0°, 4.023Å°] (Å° is the measurement unit of distance in angstroms).

We apply normalization for both angles and distance to reduce the vector space. Starting by distance it is normalized by the following equation d = floor ((d*a) / (4.023)), that makes the distance range is [0, b-1], where d is the calculated distance. The equation of torsion angles normalization is floor ((a*b) / (360)), that makes the distance range is [0, b-1], where "a" is a torsion angle in degrees.
2.3 Index structure construction

Generalized suffix array (GSA) is an indexing structure that can be constructed in linear time. It is an array that holds all possible suffixes of a set of sequences (strings) sorted in lexicographical (alphabetical) order. Indexing one sequence can be done using the suffix array, but generalized suffix array is used to index more than one sequence. To use it we start by converting each protein $P_j$ to a sequence rather than a set of vectors that is done by mapping each vector $V_i$ in the protein $P_j$ to a unique key, then we concatenate these $n$ keys of the $n$ vectors to form the protein sequence. Finally, we use the generalized suffix array to index these sequences of $P_j$.

The key idea is that after converting each protein to a sequence all the suffixes of this protein is added to the GSA in its order. If there is a suffix shared between more than one protein, we add the protein number to this element, that means each element in the suffix array maintain a list of protein numbers in which it appears. Also it stores the start index of the suffix in each of these proteins.

For example given two sequences $sequence\, 1 = 'abrac'$ and $sequence\, 2 = 'xza'$. To construct a generalized suffix array index for them, we start by finding all possible suffixes for the both sequence then we sort these suffixes lexicographically as in figure 2.

<table>
<thead>
<tr>
<th>Suffix</th>
<th>Index and sequence number</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3, 2</td>
</tr>
<tr>
<td>abrac</td>
<td>1, 1</td>
</tr>
<tr>
<td>ac</td>
<td>4, 1</td>
</tr>
<tr>
<td>brac</td>
<td>2, 1</td>
</tr>
<tr>
<td>c</td>
<td>5, 1</td>
</tr>
<tr>
<td>rac</td>
<td>3, 1</td>
</tr>
<tr>
<td>xza</td>
<td>1, 2</td>
</tr>
<tr>
<td>za</td>
<td>2, 2</td>
</tr>
</tbody>
</table>

Figure 2 GSA for $sequence\, 1 = 'abrac'$ and $sequence\, 2 = 'xza'$

3 Searching the suffix array

After building the GSA for database proteins, given a query protein and $\varepsilon$. First we extract its feature vector the same way we used for database proteins, and then we build another suffix array for the query protein(s). Finally we use the query suffix array to search for its elements in the database GSA.

We compare each suffix in the query suffix array with other suffixes in the proteins database. The comparison is held by mapping each symbol in the query suffix array and the corresponding symbol in the database GSA element to its feature vector then we calculate the distance between both the symbol in the proteins suffix array and the query suffix array. If the distance is less then or equal the epsilon ($\varepsilon$) then these two symbols is matched, test matching of the next two symbols. The process is repeated until the first mismatch is discovered, if the number of matched symbols is larger than or equal a certain threshold $l$ then there is a maximal match between both of these items in the query suffix array and database suffix array.

Matched proteins are stored with their matched symbols number in a list. This list is ranked using the best diagonal run as stated in FASTA [8]; calculating a score between the query protein and database proteins in the list, then rank the protein according to their score.

QSA is the query suffix array, and DBSA is the database suffix array, epsilon is the accepted distance and $l$ is threshold of the maximal matching. $E_{db}.length$ is the length of an element in DBSA. $E_{q}.length$ is the length of an element in QSA. $MList$ is a list that contains all matched proteins resulting from the search process.

```
INPUT : QSA, epsilon, l
OUTPUT : list of matched proteins MList
PROCEDURE : searchForQuery(QSA, DBSA, $\varepsilon$, l)

Foreach element $E_{q}$ in QSA do
  Foreach element $E_{db}$ in DBSA do
    matchedLength = match2Suffices($E_{db}$, $E_{q}$)
    if matchedLength >= this.threshold then
      Foreach proteinNo $P_{q}$ in $E_{q}$ do
        Foreach proteinNo $P_{db}$ in $E_{db}$ do
          Add($P_{q}$, $P_{db}$) to Mlist
      return Mlist

Figure 3 MaximalMatchesSearch algorithm
```

```
INPUT : $E_{db}$, DBSA element, $E_{q}$ QSA element
OUTPUT : No of matched symbols
PROCEDURE: match2Suffices ($E_{db}$, $E_{q}$, $\varepsilon$)

If $E_{db}.length > E_{q}.length$ then
  length = $E_{q}.length$
else
  length = $E_{db}.length$

matchedLength = 0
For i = 1 To length do
  if distance ($E_{db}$, $E_{q}$) < $\varepsilon$ then
    matchedLength = matchedLength + 1
  else
    exit for
return matchedLength

Figure 4 SuffixArraySearch algorithm
```

4 Experimental Results and Discussion

SCOP database is a classified database of proteins; it classifies proteins according to a four level hierarchical
classification, namely, family, super-family, fold and class. SCOP has many classes but we used in our experiment only four classes, like all previous approaches, which are: all α, all β, α + β and α / β. The dataset we used in test contains 200 proteins chosen randomly from the four classes. We build the protein database suffix array from these proteins. To build the query suffix array we select one protein file from these 200 proteins.

We compare our algorithm with PSIST algorithm, our main point of comparison is the memory usage. We run the PSIST with window size used to index the local feature vector \( w = 3 \), range used to normalize the local feature vector \( b = 10 \), the distance threshold \( \epsilon = 3 \) and the matched symbols threshold \( l = 20 \). We ran the algorithms on a PC with 3.0GHz CPU AND 512MB RAM.

Table 1 shows an enhancement in the memory usage which will be increased as the number of indexed proteins increased. The results stated in the table show memory saving with factor equal 51.7% for 200 indexed proteins.

<table>
<thead>
<tr>
<th>Proteins</th>
<th>30</th>
<th>50</th>
<th>100</th>
<th>150</th>
<th>200</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSIST Memory usage (MB)</td>
<td>1.53</td>
<td>1.92</td>
<td>5.39</td>
<td>7.53</td>
<td>9.51</td>
</tr>
<tr>
<td>Proposed method Memory usage(MB)</td>
<td>1.49</td>
<td>1.66</td>
<td>3.12</td>
<td>3.40</td>
<td>4.60</td>
</tr>
<tr>
<td>Memory Saving (%)</td>
<td>2.61</td>
<td>13.54</td>
<td>42.12</td>
<td>54.85</td>
<td>51.63</td>
</tr>
</tbody>
</table>

5 Conclusion

In this paper, we have presented a modified version of PSIST algorithm we modify local feature extraction algorithm and we use the GSA instead of generalized suffix tree, and that leaded us to modify the searching technique also. We have proposed a novel use of suffix array in searching for protein structural similarities. The experiment results showed that our proposed algorithm outperforms the PSIST in memory usage with factor over 50% on large dataset.

6 References