CSI: Clustered Segment Indexing for Efficient Approximate Searching on the Secondary Structure of Protein Sequences*

Minkoo Seo, Sanghyun Park, and Jung-Im Won

Department of Computer Science, Yonsei University, Korea
{mkseo, sanghyun, jiwon}@cs.yonsei.ac.kr

Abstract. Approximate searching on the primary structure (i.e., amino acid arrangement) of protein sequences is an essential part in predicting the functions and evolutionary histories of proteins. However, because proteins distant in an evolutionary history do not conserve amino acid residue arrangements, approximate searching on proteins' secondary structure is quite important in finding out distant homology. In this paper, we propose an indexing scheme for efficient approximate searching on the secondary structure of protein sequences which can be easily implemented in RDBMS. Exploiting the concept of *clustering* and *lookahead*, the proposed indexing scheme processes three types of secondary structure queries (i.e., exact match, range match, and wildcard match) very quickly. To evaluate the performance of the proposed method, we conducted extensive experiments using a set of actual protein sequences. According to the experimental results, the proposed method was proved to be faster than the existing indexing methods up to 6.3 times in exact match, 3.3 times in range match, and 1.5 times in wildcard match, respectively.

Keywords: Indexing method, Secondary structure of proteins, Approximate searching.

1 Introduction

It is well known to biologists that the amino acid arrangements of proteins determine their structures and functions. Therefore, it is possible to predict the functions, roles, structures, and categories of newly discovered proteins by searching for the proteins whose amino acid arrangements are similar to those of newly discovered proteins [1, 18].

However, the amino acid arrangement of one protein is rarely preserved in another protein if the two proteins are distant in an evolutionary history [5, 15]. Therefore, approximate searching on protein structures, rather than on amino

* This work was supported by the Korea Research Foundation Grant. (KRF-2004-003-D00302)

acid arrangements, is more important in finding out distant homology. Among structure searching algorithms, comparing structural arrangements based on the secondary structure elements is gaining more popularity in conjunction with database approaches [5, 14].

The secondary structures are expressed using the three characters: E (beta sheets), H (alpha helices), and L (turns or loops). These characters tend to occur contiguously rather than interspersedly [3, 11]. For example, '$HHLLLLLEEE$' is more likely to occur than '$HLLELLEELH$'.

Exploiting this property, Hammel et al. [11] proposed a segment-based indexing method. The method combines consecutive characters of a same type into a single segment and then builds a B^+-tree on two attributes of segments: (1) Type which denotes the type of consecutive characters, and (2) Len which denotes the number of consecutive characters. For example, '$HHLLLLLEEE$' is segmented into '$HH/LLLLL/EEE$' and expressed as $(H, 2)(L, 5)(E, 3)$.

Although the segmentation enables an efficient searching on the secondary structures, it has innate limitations. First, the pair of (Type, Len) does not have uniform distribution. According to our preliminary experimentation with 80,000 proteins, 87% of E segments have a length between 3 and 6, 62% of H segments have a length between 5 and 14, and 41% of L segments are of length between 3 and 6. Therefore, if every segment in a query is close to one of these *hot spots*, each index search will produce lots of intermediate results and thus the overall search performance may be worse than the full table scan. Secondly but more importantly, the number of distinct (Type, Len) pairs is not large enough to provide good selectivity. Our investigation on 80,000 proteins indicates that the total number of distinct (Type, Len) pairs is about 300 but the total number of segments to be indexed is more than 3 millions. Therefore, the average number of segments with the same (Type, Len) pair is more than 10,000.

In this paper, we propose CSI (Clustered Segment Indexing), an efficient indexing scheme for approximate searching on the secondary structure of protein sequences. The proposed indexing scheme exploits the concept of *clustering* and *lookahead* to overcome the aforementioned limitations. A pre-determined number of neighboring segments are grouped into a cluster which is then represented by three attributes: (1) CluStr which denotes the type string of the cluster obtained by concatenating the Type attributes of the underlying segments, (2) CluLen which denotes the length of the cluster obtained by summing up the Len attributes of the underlying segments, and (3) CluLA which denotes the lookahead of the cluster obtained by concatenating the Type attributes of the segments *following* the cluster. It is obvious that the triple (CluStr, CluLen, CluLA) for clusters is more discriminative than the pair (Type, Len) for segments. Therefore, by using a cluster as an indexing and query processing unit, it becomes possible to reduce both the number of intermediate results and the overall query processing time.

2 Related Work

BLAST [2] is the most widely used tool for approximate searching on DNA and protein sequences. BLAST is based on the sequential scan method basically, but it makes use of heuristic algorithms to reduce the number of sequences to be aligned against a query. However, BLAST still has two main drawbacks [18]: (1) entire data set should be loaded into a main memory for fast searching, and (2) since it is based on sequential access, its execution time is directly proportional to the number of sequences in the database. Due to these drawbacks, index-based approaches for approximate searching are demanding.

Suffix trees [16] have been recognized as the best index structure for string or sequence searching, but they have been notorious for large space requirement. Recently, algorithms for building a suffix tree from a data set larger than a main memory were proposed [13]. However, the internal structure of suffix trees is not suitable for pagination and therefore it is not easy to incorporate suffix trees into database systems [16, 17].

RAMdb [7] is an indexing system for the primary structures of protein sequences and was proved, by experiments, to be faster than heuristic approaches up to 800 times. However its search performance deteriorates when the length of a query is not close to that of the interval used for indexing. In addition, RAMdb is an indexing system mainly for the primary structures of protein sequences and therefore it is not easy to apply the proposed idea directly to the secondary structures of protein sequences.

Hammel et al. [11] proposed the segment-based indexing method. The method combines the consecutive characters of the same type into a single segment, and then builds a B^+-tree index on the number and type of consecutive characters. As mentioned in the previous section, however, this segment-based approach does not support good selectivity, thus resulting in an innate limitation of search performance.

VAST [10] and DALI [12] support three dimensional structure-based similarity search algorithms. VAST is motivated by the fact that the number of secondary structure elements (SSEs) is much smaller than the number of C_α and C_β atoms [15]. Hence, VAST performs substructure alignments in three steps: (1) rapid identification of SSE pair alignments, (2) clustering identified SSEs into groups, and (3) scoring the best substructure alignment. DALI, on the other hand, compares C_α atoms using distance matrices. For each protein, a distance matrix which resembles a dot matrix is populated. Each dot in the matrix represents the distance between C_α atoms along the polypeptide chain and between C_α atoms within the protein structure [15]. Therefore, by comparing matrices, DALI can find the proteins whose three dimensional structures are similar to that of a given query. CSI is different from VAST and DALI in that it searches for similar proteins by comparing types and lengths, rather than three dimensional coordinates, of their secondary structure elements.

3 Clustered Segment Table

A segment in a protein is defined as consecutive characters of the same secondary structure type, and can be expressed by its Type and Len attributes. A segment itself has a limitation in selectivity. Therefore, we group a pre-determined number of neighboring segments into a cluster and express it using more discriminative attributes. The procedure to construct clustered segment tables is as follows:

1. Convert each protein sequence S into a series of segments. Let N_S be the number of segments obtained from S.
2. For each k from 0 to $min(\lfloor log_2(N_s)\rfloor, MaxK)$, do the following:
   (a) Using the sliding window of size 2^k, generate a set of clusters, each of which is composed of 2^k neighboring segments.
   (b) Store each cluster into the clustered segment table named CST_k.

In the above procedure, $MaxK$ is a system parameter used to control the total number of clustered segment tables being constructed. Let $(T_1, L_1)(T_2, L_2)...$ (T_{2^k}, L_{2^k}) be 2^k neighboring segments where T_i is Type and L_i is Len of the i^{th} segment. Then the cluster is represented concisely as (CluStr $= T_1 \cdot T_2 \cdot ... \cdot T_{2^k}$, CluLen $= L_1 + L_2 + ... + L_{2^k}$).

There may be a series of segments following a cluster. The Type attributes of such segments can be concatenated, producing the lookahead, CluLA, of the cluster. The maximum length of CluLA is controlled by a system parameter $MaxCluLA$ for space efficiency. The overall schema for each clustered segment table is shown in Table 1. For example, as shown in Table 2, two clustered

Table 1. Schema of each clustered segment table

Field Name	Description
ID	The identifier of the protein from which a cluster is made.
Loc	The beginning position of the cluster.
CluStr	The type string of the cluster obtained by concatenating the Type attributes of the underlying segments.
CluLen	The length of the cluster obtained by summing up the Len attributes of the underlying segments.
CluLA	The type string obtained by concatenating the Type attributes of the segments following the cluster.

Table 2. Clustered segment tables, CST_0 and CST_1, from S_1 = '$EEEHHLLEEE$'

CST_0

ID	Loc	CluStr	CluLen	CluLA
S_1	0	E	3	HL
S_1	3	H	2	LE
S_1	5	L	2	E
S_1	7	E	3	

CST_1

ID	Loc	CluStr	CluLen	CluLA
S_1	0	EH	5	LE
S_1	3	HL	4	E
S_1	5	LE	5	

segment tables, CST_0 and CST_1, are constructed from S_1 = '$EEEHHLLEEE$' when $MaxK = 1$ and $MaxCluLA = 2$.

After populating all the tuples of CST_k, the tuples are sorted according to CluStr, CluLen, and CluLA for the sake of locality. As the final step, we build a B$^+$-tree on CluStr and CluLen attributes for each CST_k. It is also worth mentioning that the duplication of information in CST_k will bring about more storage consumption than the segment table. Hence, we store each character using 2 bits like: $L = 00_2$, $H = 10_2$, and $E = 11_2$.

4 Query Processing

4.1 Overall Query Processing Algorithm

Suppose that $MaxK + 1$ tables, CST_0, CST_1, ..., and CST_{MaxK}, were created along with their associated B$^+$-tree indices. The overall query processing algorithm which uses these tables and associated indices to process a query is as follows:

1. Convert a query Q into a series of segments. Let N_Q denote the number of segments obtained from Q.
2. Determine the target table CST_k by computing the expression $k = min(\lfloor log_2 (N_Q) \rfloor, MaxK)$.
3. Decompose the segmented query into n $(=\lceil N_Q/2^k \rceil)$ non-overlapping subqueries, each of which has 2^k segments in it. The last two subqueries may overlap each other when N_Q is not a multiple of 2^k.
4. For each subquery q_i (i=1,2,...,n), do the following:
   (a) Compute its CluStr, CluLen, and CluLA values. Let qCluStr, qCluLen, and qCluLA denote these three values, respectively.
   (b) Search the table CST_k for the tuples whose CluStr, CluLen, and CluLA match with qCluStr, qCluLen, and qCluLA, respectively. The B$^+$-tree index for CST_k is used at this step.
5. Perform the sort-merge on n sets of intermediate results using their ID and Loc as joining attributes.
6. Perform the post-processing to detect and discard false matches.

4.2 Exact Match Query

Exact match queries are expressed as $Q =< T_1(L_1)\ T_2(L_2)\ \ldots\ T_{N_Q}(L_{N_Q}) >$ where T_i $(\in \{E, H, L\})$ and L_i represent the type and length of the i^{th} segment of Q, respectively. Suppose that we already chose the target table CST_k and decomposed the query into n subqueries, each of which consists of 2^k segments. The algorithm for processing exact match queries is shown in Algorithm 1.

The result of subquery q_i is stored in N_i. If the number of tuples in N_i is less than a predefined threshold ϵ, then we believe that irrelevant answers have been filtered out sufficiently. Therefore, if this happens, we directly go to the merging step (Line 5) without considering the remaining subqueries.

Algorithm 1: ProcessExactMatchQuery

```
Input  : Query $Q$, Clustered segment table $CST_k$, Threshold $\epsilon$
Output: Set of answers
1 for (each subquery $q_i$ from $Q$) do
2     Let $qCluStr$, $qCluLen$, and $qCluLA$ be $CluStr$, $CluLen$ and $CluLA$ of $q_i$,
      respectively;
3     $N_i$ := ExecuteQuery("select * from $CST_k$ where CluStr = qCluStr
                          and CluLen = qCluLen and CluLA = qCluLA");
4     if ($count(N_i) < \epsilon$) then
          break;
5 Merge all $N_i$ into $N$ using $ID$ and $Loc$ as joining attributes;
6 answers := PostProcessing($N$);
7 return answers;
```

4.3 Range Match Query

Range match queries are expressed as $Q =< T_1(Lb_1\ Ub_1)\ T_2(Lb_2\ Ub_2) \ldots T_{N_Q}(Lb_{N_Q}\ Ub_{N_Q}) >$ where Lb_i and Ub_i represent the minimum and maximum length of the i^{th} segment of Q, respectively. In a range match query, the search condition of CluLen for each subquery q_i has the form of 'CluLen between qLb and qUb' where qLb is the sum of the minimum lengths and qUb is the sum of the maximum lengths of the underlying segments of q_i. Therefore, when the difference of qLb and qUb is large, the cost for processing q_i becomes high due to an enlarged search space for CluLen.

To overcome the problem of an enlarged search space of a subquery q_i, we propose the *selective clustering method* (SCM) where q_i is decomposed into a set of *secondary* subqueries and then a secondary subquery with the smallest *estimated* search space is chosen and executed in replacement of q_i. In detail, when a subquery q_i has 2^k segments, its secondary subqueries are generated from $2^{k'}$ underlying segments of q_i for each k' in $[0, k]$.

For example, let us consider the subquery $q_1 = (qCluStr = EH,\ qCluLen = [3+3, 5+6],\ qCluLA = L)$ from the query $Q = < E(3\ 5)H(3\ 6)L(3\ 7) >$. Let $q_{i,j}$ denote the j^{th} secondary subquery of the subquery q_i. Then, in SCM, three secondary subqueries are generated from q_1. When $k' = 0$, we obtain two secondary subqueries, $q_{1,1} = (qCluStr = E,\ qCluLen = [3, 5],\ qCluLA = HL)$ and $q_{1,2} = (qCluStr = H,\ qCluLen = [3, 6],\ qCluLA = L)$, each of which has 2^0 segment in it. Similarly, when $k' = 1$, we obtain one secondary subquery, $q_{1,3} = (qCluStr = EH,\ qCluLen = [6, 11],\ qCluLA = L)$, which has 2^1 segments in it. Among these secondary subqueries, we choose the most selective one by estimating the number of tuples to be retrieved by each secondary subquery, and execute it in replacement of q_1.

Algorithm 2: Estimate_SizeOf_ResultSet

```
Input  : Secondary subquery $q_{i,j}$
Output: Predicted number of tuples in the result set
1 Let $qCluStr$ be $CluStr$ of $q_{i,j}$;
2 Let [qLb, qUb] of $q_{i,j}$ be the range of $qCluLen$;
3 Suppose that $q_{i,j}$ is composed of $2^{qK}$ segments;
4 $N_1$ := ExecuteQuery("select sum(#Clusters) from CluStrHistogram where
  hK=qK");
5 $N_2$ := ExecuteQuery("select #Clusters from CluStrHistogram where hK=qK
                          and hCluStr=qCluStr");
6 $N_3$ := ExecuteQuery("select #Clusters from CluLenHistogram where hK=qK
                          and hCluLen between qLb and qUb");
7 return $N_3 \times N_2/N_1$;
```

In the estimation of selectivities, we use two histograms, one for CluLen and the other for CluStr. The algorithm for estimating the number of tuples to be retrieved by each secondary subquery is presented in Algorithm 2.

4.4 Wildcard Match Query

Wildcard match queries are specified as $Q =< T_1(Lb_1\ Ub_1)\ T_2(Lb_2\ Ub_2) \ldots T_{N_Q}(Lb_{N_Q}\ Ub_{N_Q}) >$ where T_i takes a value from $\{E, H, L, ?\}$. The meanings of Lb_i and Ub_i are same as before. Note that T_i may take '?' to express that the i^{th} segment can be of any secondary structure type. To accommodate this wildcard type, we just use 'CluStr like qCluStr' predicate in Algorithm 1 (Line 3) instead of 'CluStr=qCluStr' predicate.

We also apply SCM to this type of query because it has the problem of an enlarged search space for both CluStr and CluLen. Since qCluStr may contain '?', we use 'hCluStr like qCluStr' predicate in Algorithm 2 (Line 5) instead of 'hCluStr=qCluStr'.

Table 3. Schemas of CluLen and CluStr histograms

CluLen Histogram

Field Name	Description
hK	k value of a cluster
hCluLen	CluLen value of the cluster
#Clusters	Number of clusters whose k value is the same as hK and CluLen value is the same as hCluLen

CluStr Histogram

Field Name	Description
hK	k value of a cluster
hCluStr	CluStr value of the cluster
#Clusters	Number of clusters whose k value is the same as hK and CluStr value is the same as hCluStr

5 Performance Evaluation

In obtaining the secondary structures of proteins, we applied PREDATOR [8, 9] to the amino acid arrangements of proteins downloaded from PIR [19]. To verify the effectiveness of the proposed method CSI, we compared its performance with those of the three segmentation-based methods, MISS(1), MISS(2), and SSS. MISS(n) chooses the most selective n segments from a query and treats each of them as a subquery. It then executes each subquery using a B$^+$-tree on the segment table. SSS chooses the most selective segment from a query and executes it by performing a full table scan on the segment table.

5.1 Parameter Setting

To determine the values of two system parameters, $MaxK$ and $MaxCluLA$, we first performed the preliminary experiments with a data set of 80,000 protein sequences.

MaxK. The *selectivity*, a ratio of the number of tuples retrieved by a search to the total number of tuples stored in a table, was used as a measure for determining the optimal value of $MaxK$. Figure 1 shows the selectivity of a pair (CluStr, CluLen) for each clustered segment table CST_k. It is clear from the figure that the selectivity becomes better as k increases but it is saturated after k exceeds 3. Therefore, we set 3 as the optimal value of $MaxK$.

MaxCluLA. The selectivity becomes better as the value of $MaxCluLA$ increases. To measure the degree of improvement in selectivity (DIS), we use the following formula:

$$DIS = \frac{selectivity\ of\ (CluStr, CluLen)\ -\ selectivity\ of\ (CluStr, CluLen, CluLA)}{selectivity\ of\ (CluStr, CluLen)}$$

Figure 2 shows the degree of improvement in selectivity for each value of $MaxCluLA$. According to the result, the degree of improvement grows as the value of $MaxCluLA$ increases but the growth is almost saturated after the value of $MaxCluLA$ exceeds 8. Therefore, we set 8 as the optimal value of $MaxCluLA$.

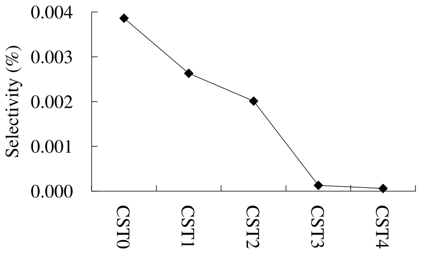

Fig. 1. Selectivity of (CluStr, CluLen) for each clustered segment table CST_k

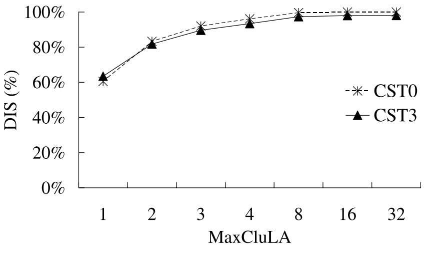

Fig. 2. Degree of improvement in selectivity (DIS) for each value of $MaxCluLA$

5.2 Query Processing Time

Query Processing Time with Various Numbers of Segments in a Query. While changing the number of segments, N_Q, in a query, we measured the query processing times of four methods, CSI, MISS(1), MISS(2), and SSS. For this experiment, we used a data set of 80,000 protein sequences from which queries were randomly extracted.

For the simplicity of experimentations, we let only the segment in the middle of range match and wildcard match queries have a range in its length specification. Considering the distribution of segment lengths, we set the size of the

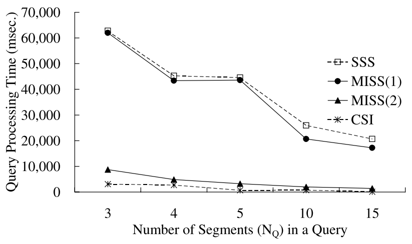

Fig. 3. Exact match query processing times with increasing N_Q

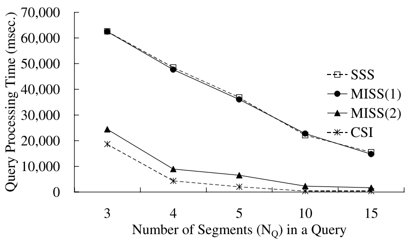

Fig. 4. Range match query processing times with increasing N_Q

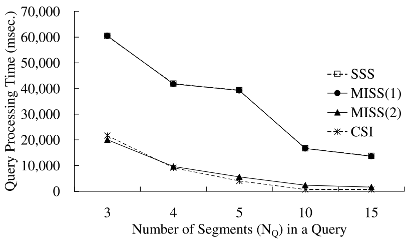

Fig. 5. Wildcard match query processing times with increasing N_Q

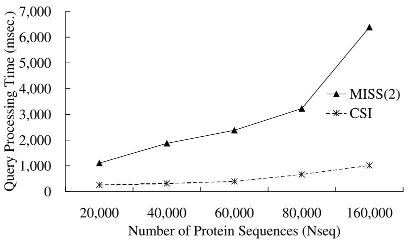

Fig. 6. Exact match query processing times with increasing N_{seq}

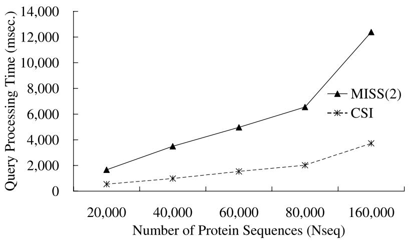

Fig. 7. Range match query processing times with increasing N_{seq}

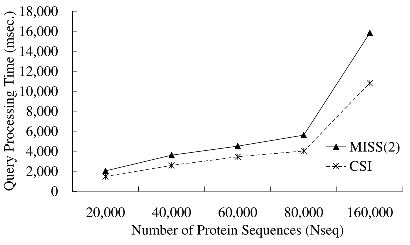

Fig. 8. Wildcard match query processing times with increasing N_{seq}

range as 30. In case of wildcard match queries, only the segment in the middle had the wildcard character '?'. Figure 3, 4, and 5 shows the query processing times of four methods for exact match, range match, and wildcard match queries, respectively.

According to the experimental results, query processing times of all methods decrease as N_Q increase. This is because more segments with high selectivity are contained in queries as N_Q increases. If N_Q is large, CSI gets extra benefit by choosing a larger k value when deciding a clustered segment table to be searched. As a result, CSI was 1.7∼13.0 times, 1.3∼6.0 times, and 1.0∼3.4 times faster than the best one of the other methods in exact match, range match, and wildcard match, respectively.

Query Processing Time with Various Data Set Sizes. While increasing the number of protein sequences, N_{seq}, from 20,000 to 160,000, we measured the query processing times of CSS and MISS(2). SSS and MISS(1) were not included in this experiment because they proved to be less efficient than MISS(2) in most cases. The number of segments in a query was 5, and ranges and wildcard characters were given only to the third segment. According to the experimental results shown in Figure 6, 7, and 8 the query processing times of both CSI and MISS(2) were proportional to the data set size, and CSI was 4.3∼6.3 times, 3.0∼3.3 times, and 1.4∼1.5 times faster than MISS(2) in exact match, range match, and wildcard match, respectively.

6 Conclusion

Approximate searching on protein structures, rather than on amino acid arrangements, are essential in finding out distant homology. In this paper, we proposed CSI, an efficient indexing scheme for approximate searching on the secondary structure of protein sequences. The proposed indexing scheme exploits the concept of clustering and lookahead to improve the selectivity of indexing attributes. Algorithms for exact match, range match, and wildcard match queries were also proposed and evaluated. The experimental results revealed that CSI is faster than MISS(2) up to 6.3 times, 3.3 times, and 1.5 times in exact match, range match, and wildcard match queries, respectively.

References

1. B. Alberts, D. Bray, J. Lweis, M. Raff, K. Roberts, and J. D. Watson, *Molecular Biology of the Cell, 3rd ed.*, Garland Publishing Inc., 1994.
2. S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman, "Gapped BLAST and PSI-BLAST: A New Generation of Protein Database Search Programs", *Nucleic Acids Research*, 25(17), 1997.
3. Z. Aung, W. Fu, and K.-L. Tan, "An Efficient Index-based Protein Structure Database Searching Method", *Proc. IEEE DASFAA Conf.*, 2003.

4. A. D. Baxevanis and B. F. F. Ouellette. *BIOINFORMATICS: A Practical Guide to the Analysis of Genes and Proteins, 2nd ed.* WILEY INTERSCIENCE, 2001.
5. O. Camoglu, T. Kahveci, and A. K. Singh, "Towards Index-based Similarity Search for Protein Structure Databases", *Proc. IEEE Computer Society Bioinformatics Conf.*, pp. 148-158, 2003.
6. I. Eidhammer and I. Jonassen, "Protein Structure Comparison and Structure Patterns - An Algorithmic Approach", *ISMB tutorial*, 2001.
7. C. Fondrat and P. Dessen, "A Rapid Access Motif Database(RAMdb) with a Searching Algorithm for the Retrieval Patterns in Nucleic Acids or Protein Databanks", *Computer Applications in the Bioscience*, 11(3), pp. 273-279, 1995.
8. D. Frishman and P. Argos, "Seventy-five Accuracy in Protein Secondary Structure Prediction", *Proteins*, 27(3), pp. 329-335, 1997.
9. D. Frishman and P. Argos, "Incorporation of Long-Distance Interactions into a Secondary Structure Prediction Algorithm", *Protein Engineering*, 9(2), pp. 133-142, 1996.
10. J. F. Gibrat, T. Madel, and S. H. Bryant, "Surprising Similarities in Structure Comparison", *Current Opinion in Structural Biology*, 6(3), pp. 377-385, 1996.
11. L. Hammel and J. M. Patel, "Searching on the Secondary Structure of Protein Sequence", In *Proc. VLDB Conf.*, 2002.
12. L. Holm and C. Sander, "Protein Structure Comparison by Alignment of Distance Matrices", *J. Molecular Biology*, 233(1), pp. 123-138, 1993.
13. E. Hunt, M. P. Atkinson, and R. W. Irving, "Database Indexing for Large DNA and Protein Sequence Collections", *VLDB Journal*, 11(3), pp. 256-271, 2002.
14. P. Koehl, "Protein Structure Similarities", *Current Opinion in Structural Biology*, 11(3), pp. 348-353, 2001.
15. D. W. Mount, *Bioinformatics.* Cold Spring Harbor Laboratory Press, 2000.
16. G. A. Stephen, *String Searching Algorithms*, World Scientific Publishing, 1994.
17. H. Wang, C.-S. Perng, W. Fan, S. Park, and P. S. Yu, "Indexing Weighted Sequences in Large Databases", *Proc. IEEE ICDE Conf.*, pp 63-74, 2003.
18. H. E. Williams, "Genomic Information Retrieval", *Proc. Australasian Database Conf.*, pp. 27-35, 2003.
19. C. H. Wu, L.-S. L. Yeh, H. Huang, L. Arminski, J. Castro-Alvear, Y. Chen, Z. Hu, P. Kourtesis, R. S. Ledley, B. E. Suzek, C. R. Vinayaka, J. Zhang, and W. C. Barker, "The Protein Information Resource", *Nucleic Acids Research*, 31(1), pp. 345–347, 2003.