PROTEIN STRUCTURE COMPARISON USING REPRESENTATION BY LINE SEGMENT SEQUENCES

TATSUYA AKUTSU   HIROSHI TASHIMO
Department of Computer Science, Gunma University
1-5-1 Tenjin, Kiryu, Gunma 376, Japan[a]

This paper proposes a new comparison method of tertiary protein structures. The method consists of two parts. First, a sequence of line segments which approximates each tertiary protein structure is computed. Then, an alignment of the sequences of line segments corresponding to input structures is computed. The proposed method is considered as an intermediate one between two comparison methods: a method based on topology diagram and a method based on structure alignment. Moreover it takes less CPU time than structure alignment methods because most structures are represented by means of sequences of at most 100 line segments. The effectiveness of the method is confirmed through a comparison with a previous method and an application to database searching for similar structures.

1 Introduction

Classification of tertiary protein structures is very important for better understanding of protein structures. Indeed several studies have been done.[6,9,10,12] Among them, the FSSP [6] (families of structurally similar proteins) database is well known. In most of them, protein structure alignment is a basic tool for classification. Protein structure alignment is, given two tertiary structures, to find equivalent residues (Cα atoms) between them. Many methods have been proposed for protein structure alignment. Rao and Rossmann,[11] and Pascarella and Argos [10] proposed iterative improvement methods. Vriend and Sander [14] developed a greedy method in which small fragments were assembled into larger structures. Taylor and Orengo [13] developed the SSAP algorithm using the double dynamic programming technique. Holm and Sander [6] developed the DALI algorithm using Monte Carlo optimization to compare distance matrices. Nussinov and Wolfson [8] applied geometric hashing to protein structure alignment. Šali and Overington [12] developed a stochastic method using probability density functions.

Although these proposed methods work very well in general, Holm et al. [6] point out that each one has one or more limitations. Thus, they use three algorithm to classify tertiary protein structures. Moreover, consider the following extreme case: although two proteins have similar core structures, one consists of β-strands while the other consists of α-helices. In this case, it seems that

[a] **e-mail:** akutsu@cs.gunma-u.ac.jp   tashimo@keim.cs.gunma-u.ac.jp

most structure alignment algorithms fail to find the similarity because they try to find one-to-one correspondences between Cα atoms. Although this case is not probable, a similar case might occur in the future. Thus, protein structure alignment is too detailed in such a case. Another problem is that protein structure alignment takes long CPU time in general. Thus, it will take a large amount of CPU time if applied to classification of protein structures. Since the number of tertiary structures stored in PDB[4] (Protein Data Bank) is increasing rapidly and the number of structure comparisons required to classify n protein structures is $O(n^2)$, much faster structure alignment methods are required.

On the other hand, classification by topology diagram has been used traditionally.[5] Using this, protein structures are classified into such families as: α/β sandwich, α/β doubly wound, TIM barrel, Greek key, Globin, $\cdots$. In this method, each protein structure is treated as a sequence of α-helices, β-strands and turns. Although this method is useful, it seems difficult to make the classification procedure automatic and difficult to classify structures into more detailed families.

From the above discussions, it seems that there is a case where classification by structure alignment is too detailed and classification by topology diagram is too rough, although most cases can be treated by the usual structure alignment methods. Thus, an intermediate classification method is required. In this paper, we propose such a method. In this method, each tertiary structure is represented by a sequence of line segments (see Fig. 1) and the similarity between two structures is measured by the score of the alignment between two sequences of line segments. Note that, once such representation (i.e., a sequence of line segments) is computed, the alignments can be computed quickly because most structures are represented by means of sequences of at most 100 line segments. Thus the proposed comparison method is very useful for classification of tertiary structures in which a lot of comparisons are required.

We compare the proposed comparison method with a conventional structure alignment algorithm[3] using PDB[4] data. Moreover, we apply the proposed method to database searching for similar tertiary structures, which will lead to classification of tertiary protein structures. The results of these experiments show that the proposed method is much faster than the previous one and classifies tertiary structures as well as the previous one does.

The organization of this paper is as follows. First, a method for computing a sequence of line segments from a tertiary structure is described. Next, a method for computing an alignment between two sequences of line segments is described. Then, experimental results are described. Finally, we conclude with a brief summary and future work.

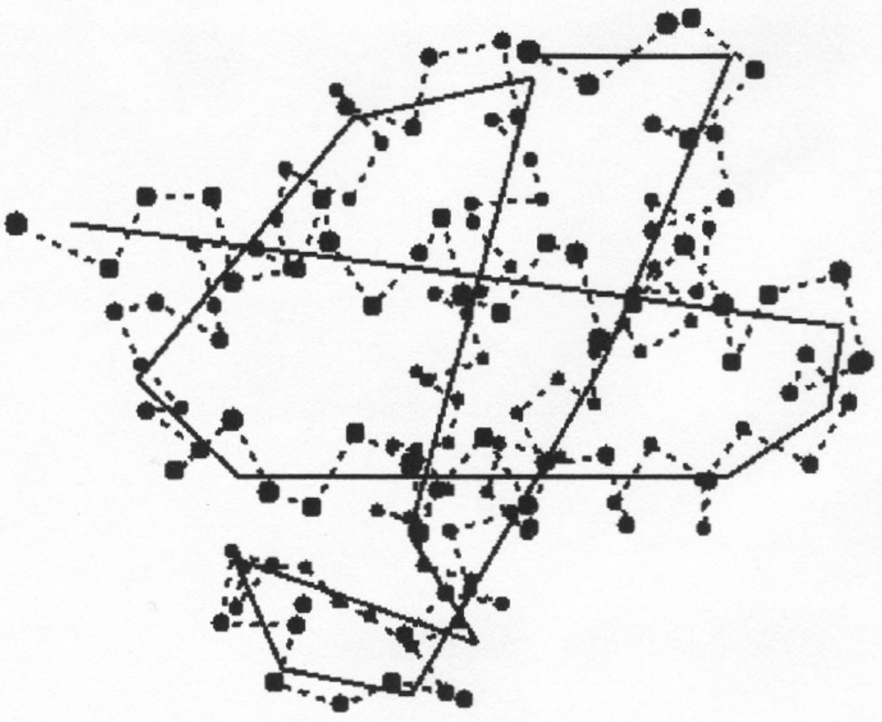


Figure 1: An example of a sequence of line segments computed from protein structure '4hhb'. Black circles denote the input Cα atoms and black lines denote the obtained line segments.

2 Representation by a Sequence of Line Segments

As mentioned in Section 1, the proposed comparison method consists of two parts: computation of a sequence of line segments from a tertiary structure, and alignment of two sequences of line segments. In this section, we describe a method to compute a sequence of line segments which approximates an outline of an input tertiary structure. First a basic version is described, and then an improved one is described.

2.1 Method

We assume that each tertiary protein structure is input as a sequence of points (i.e., a sequence of Cα atoms). This representation method is used in most structure alignment algorithms.[8,13,14] Thus we let $P = (\boldsymbol{p}_1, \cdots, \boldsymbol{p}_n)$ be an input tertiary structure, where each $\boldsymbol{p}_i$ denotes a point in the three-dimensional Euclidean space. Then we compute a sequence of line segments via the following two steps (see Fig. 2):

(1) Compute a sequence of lines $(L_1, \cdots, L_k)$ approximating an outline of a protein structure P.

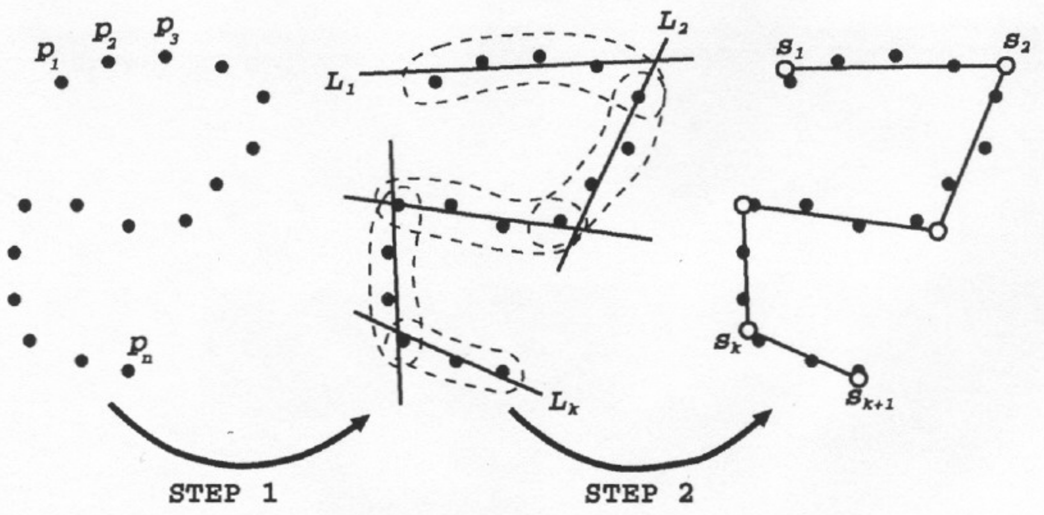


Figure 2: A method for computing a sequence of line segments.

(2) Compute a sequence of line segments $ss(P) = (\overline{s_1 s_2}, \cdots, \overline{s_k, s_{k+1}})$ from the sequence of lines obtained in step (1).

First, we consider step (1). Let $LS = (L_1, L_2, \cdots, L_k)$ be a sequence of lines, and $I = (i_1, \cdots, i_{k+1})$ be a sequence of integer numbers such that $i_1 = 1$, $i_{k+1} = n$ and $i_j < i_{j+1}$. We define a score $fit(P, LS, I)$ by

$$fit(P, LS, I) = \sqrt{\frac{1}{n+k-1}(\sum_{j=i_1}^{i_2} d(p_j, L_1)^2 + \sum_{j=i_2}^{i_3} d(p_j, L_2)^2 + \cdots + \sum_{j=i_k}^{i_{k+1}} d(p_j, L_k)^2)}$$

where $d(\boldsymbol{p}_j, L_h)$ denotes the distance between a point $\boldsymbol{p}_j$ and a line L_h. Note that $fit(P, LS, I)$ corresponds to the average distance between points and lines. In step (1), we compute a pair (LS, I) that minimizes k under the condition that $fit(P, LS, I) \leq \delta$, where $\delta > 0$ is a constant. In order to compute such pair (LS, I), we consider the following problem: given P and k, compute a pair (LS, I) that minimizes $fit(P, LS, I)$. This problem can be solved in $O(n^3)$ time, using the least-squares fitting technique and the dynamic programming technique. Here we briefly describe the algorithm.

Let $lsf(i, j)$ $(i < j)$ denotes the sum of squares of distances that is computed from an application of the least-squares fitting technique to $\boldsymbol{p}_i, \boldsymbol{p}_{i+1}, \cdots, \boldsymbol{p}_j$. That is, $lsf(i, j)$ denotes $\min \sum_{h=i}^{j} d(\boldsymbol{p}_h, L)^2$, where the minimum is taken from all lines L in three-dimensions. From P and k, we construct a directed graph $G(V, E)$ such that

$$V = \{(\boldsymbol{p}_i, h) \mid \boldsymbol{p}_i \in P, 1 \leq h < k\} \cup \{START, GOAL\},$$

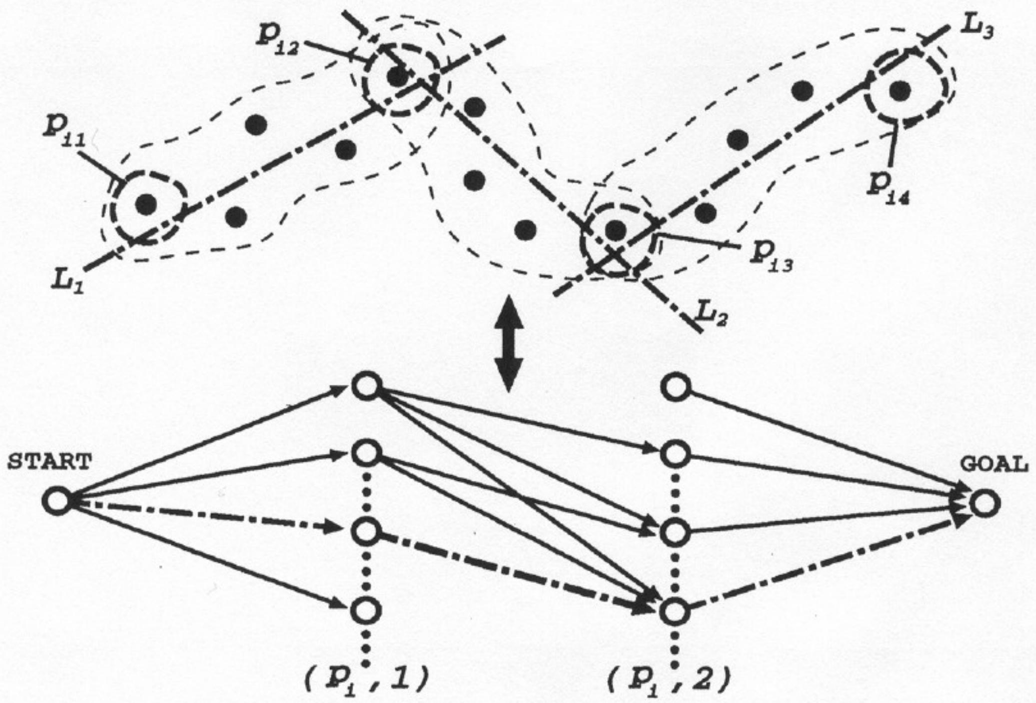


Figure 3: Transformation to the shortest path problem. A shortest path (denoted by dashed arcs) corresponds to a sequence of lines minimizing $fit(P, LS, I)$, where $k = 3$ in this example.

$$\begin{aligned} E &= \{\langle START, (\boldsymbol{p}_i, 1)\rangle\} \cup \{\langle(\boldsymbol{p}_i, k-1), GOAL\rangle\} \\ &\quad \cup \{\langle(\boldsymbol{p}_i, h), (\boldsymbol{p}_j, h+1)\rangle \mid i < j,\ 1 \leq h < k-1\}, \end{aligned}$$

where the costs of edges are given by

$$cost(\langle START, (\boldsymbol{p}_i, 1)\rangle) = lsf(1, i), \qquad cost(\langle(\boldsymbol{p}_i, k-1), GOAL\rangle) = lsf(i, n),$$
$$cost(\langle(\boldsymbol{p}_i, h), (\boldsymbol{p}_j, h+1)\rangle) = lsf(i, j).$$

Then, a minimum cost path (i.e., a shortest path) from $START$ to $GOAL$ corresponds to a pair (LS, I) that minimizes $fit(P, LS, I)$ (see Fig. 3). Thus, such a pair can be computed by solving the shortest path problem for $G(V, E)$. Since $G(V, E)$ is an acyclic graph, the shortest path problem can be solved in $O(|E|) = O(kn^2)$ time using the dynamic programming technique (the reaching algorithm [2]). Although $O(n)$ time is required for computing a cost of each edge, the total computation time for computing all the costs of edges is $O(n^3)$ because there are $O(n^2)$ distinct costs (note that $cost(\langle(\boldsymbol{p}_i, h), (\boldsymbol{p}_j, h+1)\rangle) = cost(\langle(\boldsymbol{p}_i, h'), (\boldsymbol{p}_j, h'+1)\rangle)$ for all $1 \leq h, h' < k-1$). Therefore, the pair (LS, I) that minimizes $fit(P, LS, I)$ can be computed in $O(kn^2 + n^3) = O(n^3)$ time. Note that the above problem is a variant of the k-link path problem,[1] which is well-known in computational geometry.

Using the above algorithm as a subroutine, the following procedure computes a pair (LS, I) minimizing k under the condition that $fit(P, LS, I) \leq \delta$.

```
Procedure $ComputeSequenceOfLines(P, \delta)$
  begin $k := 1$;
    repeat
      Compute $(LS, I)$ that minimizes $fit(P, LS, I)$ where $|LS| = k$;
      $k := k + 1$
    until $fit(P, LS, I) \leq \delta$;
    Output $(LS, I)$
  end
```

If this procedure is implemented as it is, it would take $\sum_{k=1}^{n} O(n^3) = O(n^4)$ time. However, this procedure can be implemented so that it works in $O(n^3)$ time by means of the following modification. Note that, adding $O(n^2)$ edges to a graph for k, we can obtain a graph for $k+1$. Moreover, a tail of each added edge is a newly created node. Thus, testing newly added $O(n^2)$ edges (in the reaching algorithm[2]), we can obtain a shortest path for $k+1$ in $O(n^2)$ time. Therefore, the total computation time is reduced to $O(n^3)$.

The choice of δ is important because δ affects the quality of the obtained sequence. Currently we use $\delta = 2.35\text{Å}$, where this value was determined by experiment.

Next we consider step (2). Step (2) is very simple although it is rather ad hoc (see Fig. 4). Note that we must create endpoints of each line segment because two lines do not necessarily have a common point in three-dimensions. For each pair of lines (L_j, L_{j+1}) $(1 \leq j < k)$, we compute a point $s_j = \frac{q+r}{2}$, where $q \in L_j$ (resp. $r \in L_{j+1}$) is a point such that $|\overline{p_{i_{j+1}} q}|$ (resp. $|\overline{p_{i_{j+1}} r}|$)

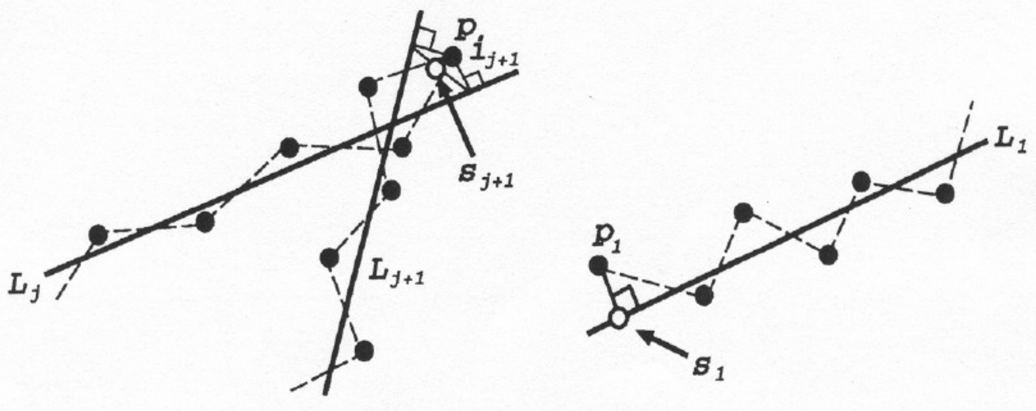


Figure 4: Computation of endpoints of line segments in Step (2).

is minimum. Moreover, we compute a point $\boldsymbol{s}_1 \in L_1$ (resp. $\boldsymbol{s}_{k+1} \in L_k$) such that $|\overline{\boldsymbol{s}_1\boldsymbol{p}_1}|$ (resp. $|\overline{\boldsymbol{s}_{k+1}\boldsymbol{p}_n}|$) is minimum. Finally, we obtain a sequence of line segments $ss(P) = (\overline{\boldsymbol{s}_1\boldsymbol{s}_2}, \overline{\boldsymbol{s}_2\boldsymbol{s}_3}, \cdots, \overline{\boldsymbol{s}_k\boldsymbol{s}_{k+1}})$. Note that, in this paper, $\overline{\boldsymbol{xy}}$ denotes a line segment connecting points $\boldsymbol{x}$ and $\boldsymbol{y}$, and $|\overline{\boldsymbol{xy}}|$ denotes the length of a line segment $\overline{\boldsymbol{xy}}$.

2.2 Improvement

Although the above algorithm works well in most cases, there are some cases where good approximations are not computed. For example, in the case of Fig. 5, representation (A) is computed. However, in this case, representation (B) should be computed. Thus, we have improved the algorithm so that such representation as (B) can be computed. The improvement is done by removing bad edges from $G(V, E)$ in the following way. Let L be a line obtained by applying the least-squares fitting technique to $\boldsymbol{p}_i, \cdots, \boldsymbol{p}_j$, and let $\boldsymbol{v}$ be a unit vector parallel to L. Let $\boldsymbol{x} \cdot \boldsymbol{v}$ denote the inner product between vectors $\boldsymbol{x}$ and $\boldsymbol{v}$. Then, each edge $\langle(\boldsymbol{p}_i, h), (\boldsymbol{p}_j, h+1)\rangle$ such that there exists $i \le l < j$ satisfying $\boldsymbol{p}_l \cdot \boldsymbol{v} > \boldsymbol{p}_{l+1} \cdot \boldsymbol{v}$ is removed from $G(V, E)$. Moreover, each edge $\langle(\boldsymbol{p}_i, h), (\boldsymbol{p}_j, h+1)\rangle$ such that $lsf(i, j) > (j - i + 1)\delta^2$ is also removed from $G(V, E)$.

This improvement is effective not only for obtaining good approximations but also for reducing the computation time. Since bad lines are removed and most lines are bad lines, the computation time is considerably reduced although $O(n^3)$ time is still required in the worst case. Table 1 shows the CPU times for the basic version and the improved version on SUN Sparc Station-10 (a unix workstation). PDB code and the size (the number of residues) are described for each protein structure. Note that in this paper, only chain A is used in each structure. You can see that the computation time is considerably reduced

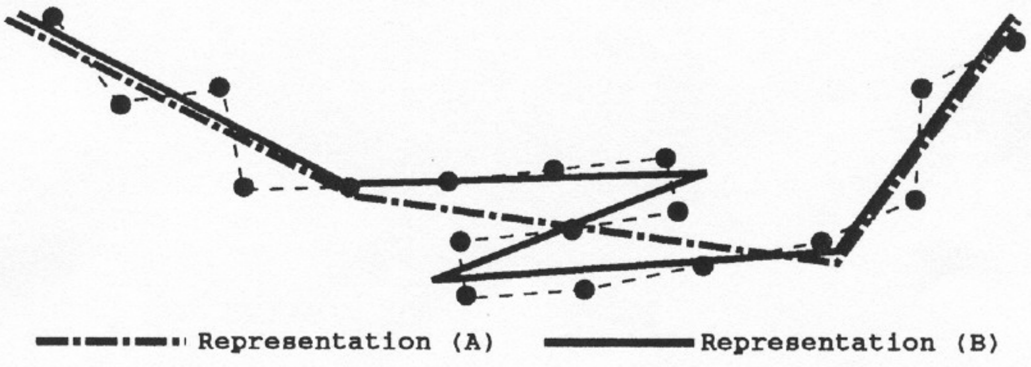


Figure 5: A bad case for the basic version.

by this improvement. Examples of sequences of line segments obtained by this improved algorithm are shown in Fig. 6.

Table 1: CPU times for computing line segment representations.

PDBcode	SIZE	BASIC(sec)	IMPROVED(sec)
1tgn	245	137.33	2.18
2trm	245	139.82	1.98
2fvb	106	125.80	2.19
2fvw	116	135.99	2.36
4hhb	141	24.26	1.01
5mbn	153	33.48	1.11

3 Comparison of Structures

We compare two tertiary structures using the line segment representation described in Section 2. The comparison is done via the following two steps.

(1) $ss(P)$ and $ss(Q)$ are transformed into strings $str(ss(P))$ and $str(ss(Q))$ respectively.

(2) The score between $str(ss(P))$ and $str(ss(Q))$ is computed applying the standard string (sequence) alignment algorithm.

In Subsections 3.1 and 3.2, we describe a basic method corresponding to the above steps. In Subsection 3.3, we describe an improved procedure using the double dynamic programming technique.[13]

3.1 Transformation into Strings

From a sequence of line segments $ss(P) = (\overline{\boldsymbol{s}_1\boldsymbol{s}_2}, \overline{\boldsymbol{s}_2\boldsymbol{s}_3}, \cdots, \overline{\boldsymbol{s}_k\boldsymbol{s}_{k+1}})$, we construct a string $str(ss(P))$ in the following way. Let $\boldsymbol{c}_i$ be the centroid of a line segment $\overline{\boldsymbol{s}_i\boldsymbol{s}_{i+1}}$. For segments $\overline{\boldsymbol{s}_i\boldsymbol{s}_{i+1}}$ and $\overline{\boldsymbol{s}_j\boldsymbol{s}_{j+1}}$, we define the following values (see Fig. 7):

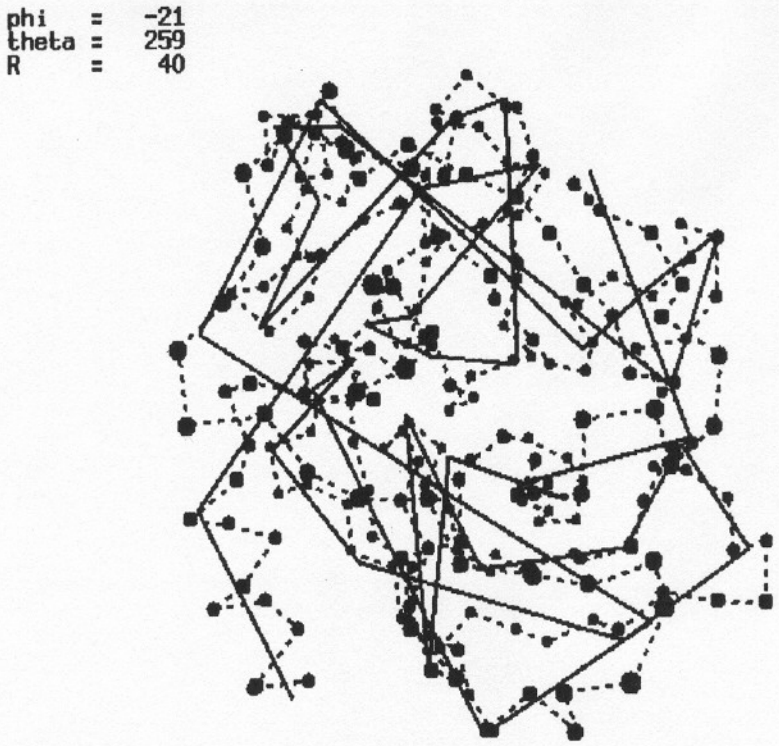
l_i**:** a length of a line segment $\overline{\boldsymbol{s}_i\boldsymbol{s}_{i+1}}$ (i.e., $l_i = |\overline{\boldsymbol{s}_i\boldsymbol{s}_{i+1}}|$),

$l_{i,j}$**:** a length between $\boldsymbol{c}_i$ and $\boldsymbol{c}_j$,

$\alpha_{i,j}$**:** an angle between $\overline{\boldsymbol{s}_i\boldsymbol{s}_{i+1}}$ and $\overline{\boldsymbol{s}_j\boldsymbol{s}_{j+1}}$,

$\beta_{i,j}$**:** an angle between $\overline{\boldsymbol{c}_i\boldsymbol{c}_j}$ and $\overline{\boldsymbol{s}_i\boldsymbol{s}_{i+1}}$,

$\gamma_{i,j}$**:** an angle between $\overline{\boldsymbol{c}_i\boldsymbol{c}_j}$ and $\overline{\boldsymbol{s}_j\boldsymbol{s}_{j+1}}$.



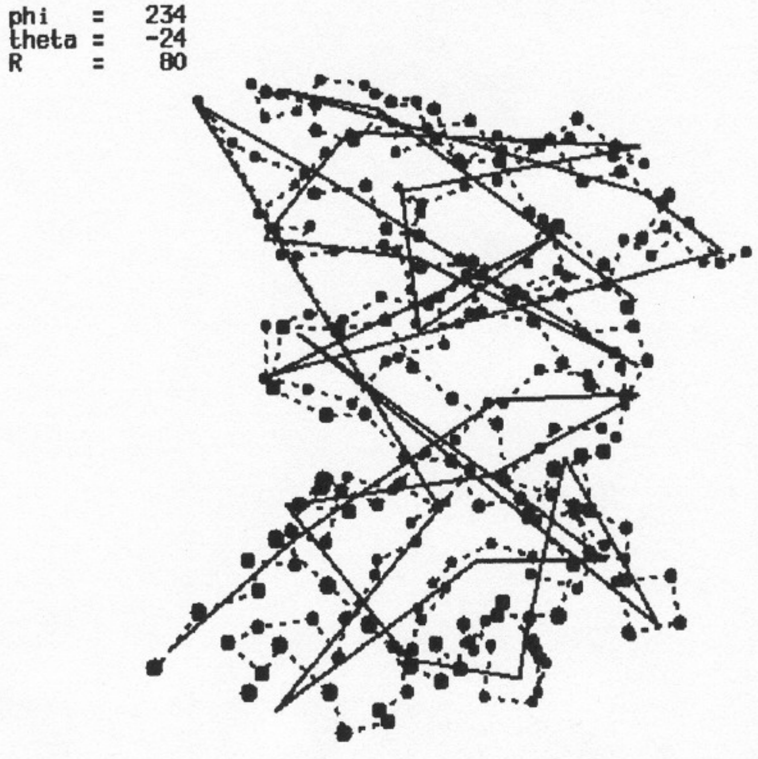


Figure 6: Sequences of line segments computed from '1tgn' and '2fvw' respectively. Sequences of $k = 37$ and $k = 30$ are obtained respectively.

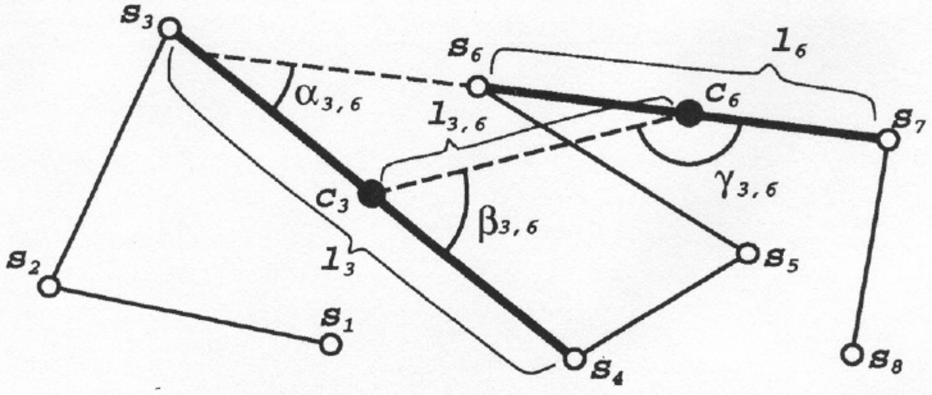


Figure 7: Definitions of l_i,$l_{i,j}$,$\alpha_{i,j}$,$\beta_{i,j}$ and $\gamma_{i,j}$ used in $str(ss(P))$.

For each pair of segments $\overline{s_i s_{i+1}}$ and $\overline{s_j s_{j+1}}$, we define $ch(i,j)$ as a sequence of six real numbers by $ch(i,j) = \langle l_i, l_j, l_{i,j}, \alpha_{i,j}, \beta_{i,j}, \gamma_{i,j} \rangle$. Note that $ch(i,j)$ corresponds to a character in a string. For each segment $\overline{s_i s_{i+1}}$, we define $str(i)$ as a sequence of such characters by

$$str(i) = (ch(i,i+1), ch(i,i+2), \cdots, ch(i,i+D)) ,$$

where D is some constant. Then $str(ss(P))$ is obtained by concatenating $str(1), str(2), \cdots, str(k-D)$, where concatenation of $(t_1, \cdots, t_p)$ and $(u_1, \cdots, u_q)$ is $(t_1, \cdots, t_p, u_1, \cdots, u_q)$.

3.2 Alignment of Transformed Strings

To apply the standard alignment algorithm, we must define a score between two characters. For $ch(i,j) = \langle l_i, l_j, l_{i,j}, \alpha_{i,j}, \beta_{i,j}, \gamma_{i,j} \rangle$ in P and $ch(g,h) = \langle l_g, l_h, l_{g,h}, \alpha_{g,h}, \beta_{g,h}, \gamma_{g,h} \rangle$ in Q, we define a score ($score(ch(i,j), ch(g,h))$) between $ch(i,j)$ and $ch(g,h)$ to be

$$c_1 - c_2|l_i - l_g| - c_3|l_j - l_h| - c_4|l_{i,j} - l_{g,h}| - c_5|\alpha_{i,j} - \alpha_{g,h}| - c_6|\beta_{i,j} - \beta_{g,h}| - c_7|\gamma_{i,j} - \gamma_{g,h}|,$$

where $c_1, \cdots, c_7$ are appropriate constants ($c_1 = 100.0$, $c_2 = c_3 = 0.2$, $c_4 = 0.5$, $c_5 = c_6 = c_7 = 10.0$ are used in the current version). Then, for two protein structures P and Q, we compute an optimal alignment between $str(ss(P))$ and $str(ss(Q))$ by means of the standard alignment algorithm for two strings. Finally the score of an optimal alignment indicates the similarity between P and Q. It is expected that the score is high if P is similar to Q, and the score is low if P is not similar to Q (see Fig. 8). Note that not only local similarities but also global similarities are taken into account if large D is used.

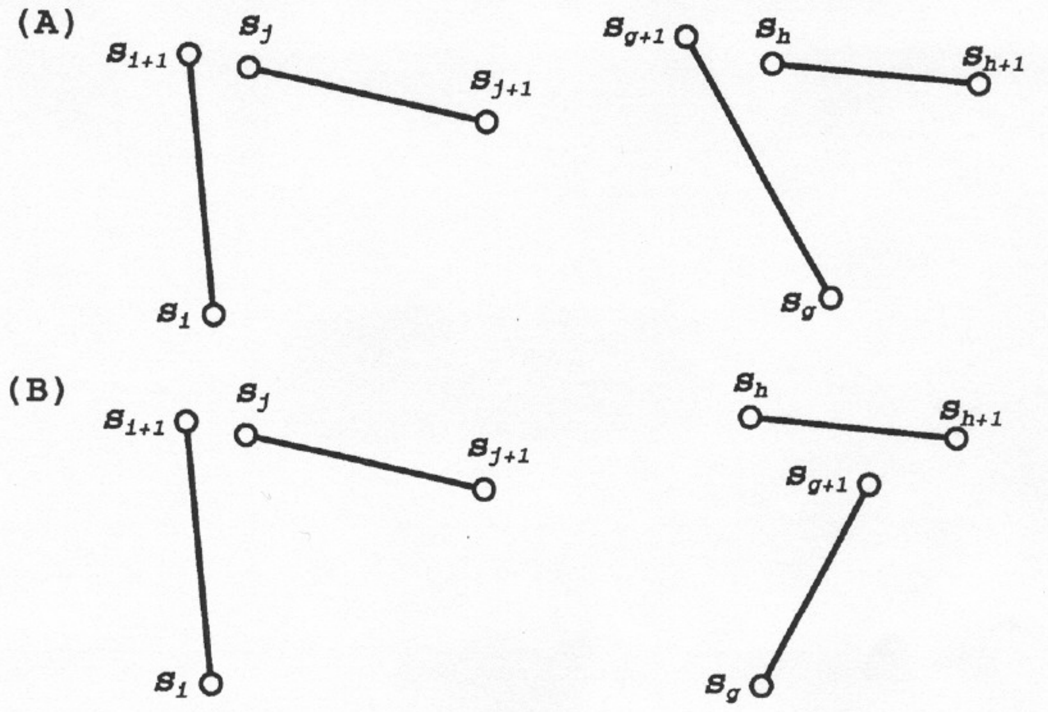


Figure 8: The score between $ch(i,j)$ and $ch(g,h)$ is high in (A), while it is low in (B).

3.3 Alignment by Double Dynamic Programming

Although the similarity between $str(ss(P))$ and $str(ss(Q))$ can be computed by the standard alignment algorithm, a correspondence (an alignment) between $ss(P)$ and $ss(Q)$ can not be computed. To compute an alignment between $ss(P)$ and $ss(Q)$, we apply the double dynamic programming method proposed by Taylor and Orengo,[13] in which the dynamic programming procedure is applied at two levels. While the dynamic programming procedure is applied to residues (or points) in their method, it is applied to line segments in our method. The procedure $DoubleDP(ss(P), ss(Q), D)$ describes an outline of the alignment algorithm for line segments, where n_1 (resp. n_2) denotes the number of line segments in $ss(P)$ (resp. $ss(Q)$).

Note that the main routine ($DoubleDP(ss(P), ss(Q), D)$) corresponds to the standard alignment algorithm, where scores are computed using a subroutine $InnerDP(i,j)$. $InnerDP(i,j)$ computes a score between two line segments $\overline{s_i s_{i+1}}$ in $ss(P)$ and $\overline{s_j s_{j+1}}$ in $ss(Q)$ applying the standard alignment algorithm to sequences $(\overline{s_i s_{i+1}}, \cdots, \overline{s_{i+D} s_{i+D+1}})$ in $ss(P)$ and $(\overline{s_j s_{j+1}}, \cdots, \overline{s_{j+D} s_{j+D+1}})$ in $ss(Q)$. Note that the score defined in Subsection 3.2 is used to compare two pairs of line segments. This alignment procedure works in $O(D^2 n^2)$ time, where n is $\max(n_1, n_2)$. In the current version, we use $D = 5$,

$GAP_1 = 35.0$, $GAP_2 = 0.0$.

```
**procedure** $DoubleDP(ss(P), ss(Q), D)$
  **begin**
    **for** $i := 0$ to $n_1$ **do** $a[i][0] := i \times GAP_1$;
    **for** $j := 0$ to $n_2$ **do** $a[0][j] := j \times GAP_1$;
    **for** $i := 1$ to $n_1 - D$ **do**
      **for** $j := 1$ to $n_2 - D$ **do**
        **begin**
          $s := InnerDP(i, j)$;
          $a[i][j] := \max(a[i-1][j] + GAP_1, a[i][j-1] + GAP_1, a[i-1][j-1] + s)$
        **end**;
      Output $a[n_1][n_2]$
    **end**
```

```
**function** $InnerDP(i, j)$
  **begin**
    **for** $i_1 := 0$ to $D$ **do** $b[i_1][0] := i \times GAP_2$;
    **for** $j_1 := 0$ to $D$ **do** $b[0][j_1] := j \times GAP_2$;
    **for** $i_1 := 1$ to $D$ **do**
      **for** $j_1 := 1$ to $D$ **do**
        **begin**
          $s := score(ch(i, i + i_1), ch(j, j + j_1))$;
          $b[i_1][j_1] := \max(b[i_1 - 1][j_1] + GAP_2,$
                      $b[i_1][j_1 - 1] + GAP_2, b[i_1 - 1][j_1 - 1] + s)$
        **end**;
    **return** $b[D][D]$
  **end**
```

4 Experimental Results

The proposed comparison method was examined on SUN Sparc Station 10 using PDB data. All methods were implemented in the C programming language. First, we compare the proposed method with an existing protein structure alignment algorithm. Next, we apply the proposed method to searching for similar structures in a database.

4.1 Comparison with a Previous Method

We have compared the proposed method (denoted by NEW) with a protein structure alignment algorithm[3] (denoted by DP), where the improved method (described in Subsections 2.2 and 3.3) was used for NEW. DP was previously proposed by us, in which input structures are divided into small fragments and then a dynamic programming technique is applied.[3] DP tries to find a set of spatially equivalent Cα atom pairs between two input structures with keeping the rms (root mean square) distance between corresponding points small. Note that DP is similar to the algorithms by Taylor and Orengo [13] and Vriend and Sander.[14]

Table 2 shows the results. PDB code and the size (the number of residues) are described for each input protein structure. The score in DP is defined by

$$100 \times \frac{\text{size of the obtained alignment (the number of atom pairs)}}{\text{size of smaller input structure}}$$

while scores in NEW are normalized so that the score between the same structures becomes 100.0. CPU times are shown for both DP and NEW.

Table 2: Comparison of the methods for tertiary structure comparison.

INPUT1		INPUT2		DP		NEW	
PDB code	SIZE	PDB code	SIZE	SCORE	TIME (sec)	SCORE	TIME (sec)
2fvb	106	2fvw	112	75.5	2.30	84.5	0.05
4hhb	141	5mbn	153	85.1	1.67	58.3	0.07
1tgn	245	2trm	245	81.6	2.75	74.2	0.43
2fvb	106	4hhb	141	9.4	0.85	30.0	0.06
2fvb	106	2trm	245	9.4	1.1	28.1	0.16
5mbn	153	2fvw	112	8.9	2.14	34.1	0.06
5mbn	153	2trm	245	13.1	7.31	34.1	0.15
1tgn	245	2fvw	112	8.9	3.53	25.6	0.15
1tgn	245	4hhb	141	7.1	2.66	40.0	0.19

You can see that the scores of the first three pairs are higher than those of the other pairs in both methods. Indeed, looking at each structures, we can see that two structures are similar to each other in the first three pairs, and two structures are not similar to each other in the other pairs. Thus it is confirmed that NEW finds the similarity between tertiary structures as well as DP does.

Next, we consider the CPU time. Note that the time for computing a sequence of line segments is not included in NEW. Such computation must be done only when a new tertiary structure is registered into a database. Moreover, it takes a few seconds per structure. Thus we can ignore the time for computing a sequence of line segments. From Table 2, it is seen that NEW is much faster than DP.

4.2 Result on Database Search

We have applied the presented method to database searching for similar structures. We use 811 structures of PDB since we only have an old version of PDB. Note that the following preprocessing is done before the experiment: for all structures, sequences of line segments are computed and stored as files along with tertiary structure data. Although this preprocessing takes a few hours, it may be done only once.

Table 3 shows the result of the experiment. Each item in PDBcode denotes an input protein structure. For each input structure, protein structures that have the scores more than a threshold value are listed in MATCHED, where we use 50.0 as a threshold value, which is determined by experiment. The number of matched structures is shown in NUM. For each structure, CPU time for the comparison with 810 structures is described.

Note that most structures in MATCHED have tertiary structures (folding patterns) similar to an input structure. Indeed, DP computed high scores for each pair of structures in MATCHED in most cases. Moreover, the obtained result seems to be similar to (a part of) the classification result by Šali and Overington[12] although we can not compare the results directly since different data are used.

Next, note that CPU time for each input structure is less than 7 minutes. If we apply DP to searching for similar structures, it would take a few hours.

5 Conclusion

We have proposed a new method for comparing tertiary protein structures. In this method, a sequence of line segments which approximates each tertiary protein structure is computed and then an alignment of the obtained line segments is computed. Comparison with a previous method (DP) shows that the proposed method is much faster than the previous one and classifies tertiary structures as well as the previous one does.

However, we have not yet shown an advantage of the proposed method over the other structure alignment algorithms, especially from a viewpoint of the quality of classification. Thus, we are planning to apply the proposed method to classification of tertiary structures using the latest version of PDB and compare the results with such database as FSSP.[6] This is the most important future work.

Table 3: Result on searching for similar structures.

PDB code	TIME (sec)	NUM	MATCHED (PDBcode/SCORE)
1tim	273.8	6	1ypi/68.1 2ypi/68.9 3tim/66.8 4tim/68.5 5tim/68.6 6tim/67.8
4fab	173.1	12	1fai/77.5 1fc1/58.4 1fdl/76.8 1igf/86.0 1mcp/79.4 2f19/79.5 2fb4/61.5 2fbj/77.9 2ig2/61.4 2igf/85.2 2mcp/81.5 3hfm/81.9
2hhb	96.7	45	1eca/60.8 1coh/99.3 1fdh/79.1 1hbs/58.6 2sdh/60.1 1lh1/53.4 1mba/59.9 1pmb/65.3 ···
2prk	313.1	15	1mee/72.7 1cse/70.5 1s01/69.7 1s02/69.8 1sbc/71.9 1sbt/64.7 1sic/62.0 1st2/70.0 1tec/70.0 2sbt/58.8 2sec/71.7 2sni/67.9 2st1/72.5 2tec/75.5 3tec/75.5
3ldh	385.2	9	1ldb/67.8 1ldm/83.7 1llc/73.1 1ldb/65.3 1ldx/66.0 5ldh/66.5 6ldh/73.3 8ldh/72.3 4mbh/63.9
4cpv	102.1	14	1cdp/97.9 1pal/65.9 2pal/80.0 3cln/54.3 3pal/80.0 4pal/85.1 5cpv/82.1 1cc5/53.0 3cyt/55.6 451c/56.8 5cyt/55.9 1omd/69.5 4tnc/51.0 2cln/52.9
1trm	252.5	59	1chg/67.3 1gct/84.9 2est/79.0 2kai/74.3 2pka/84.0 3rp2/84.0 1sgt/67.2 1tab/84.4 ···
1hla	242.9	3	1hsa/73.2 2hla/70.7 3hla/74.7

Another future work is to improve the proposed method. For example, using an existing program to identify α-helices and β-strands in tertiary structures,[7] better fittings (better sequences of line segments) might be obtained because biological and chemical structures of proteins are taken into account in such a case. For another example, using special kinds of curves instead of line segments, better fittings might be obtained. Such methods should be studied since better representation may lead to better comparison methods.

Acknowledgments

This work was partially supported by the Grant-in-Aid for Scientific Research on Priority Areas, "Genome Informatics", of the Ministry of Education, Science and Culture of Japan.

References

1. A. Agarwal, B. Schieber and T. Tokuyama, "Finding a minimum weight K-link path in graphs with Monge property and applications," *Proc. ACM Symposium on Computational Geometry*, 189–197 (1993).
2. R. K. Ahuja, T. L. Magnanti and J. B. Orlin, *Network Flows - Theory, Algorithms, and Applications*, (Prentice Hall, NJ, 1993).
3. T. Akutsu, "Efficient and robust three-dimensional pattern matching algorithms using hashing and dynamic programming techniques," *Proc. 27th Hawaii Int. Conf. on System Sciences* **5**, 225–234 (1994).
4. F. C. Bernstein *et al.*, "The Protein Data Bank: A computer-based archival file for macromolecular structures," *J. Molecular Biology* **112**, 535–542 (1976).
5. C. Branden and J. Tooze, *Introduction to Protein Structure*, (Garland Publishing, 1991).
6. L. Holm, C. Onzounis, C. Sander, G. Tuparev and G. Vriend, "A database of protein structure families with common folding motifs," *Protein Science* **1**, 1691–1698 (1992).
7. W. Kabsch and C. Sander, "Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded and geometrical features," *Biopolymers* **22**, 2577–2637 (1983).
8. R. Nussinov and H. J. Wolfson, "Efficient detection of three-dimensional structural motifs in biological macromolecules by computer vision techniques," *Proc. Natl. Acad. Sci. (USA)* **88**, 10495–10499 (1991).
9. C. A. Orengo, D. T. Jones and J. M. Thornton, "Protein superfamilies and domain superfolds," *Nature* **372**, 631–634 (1994).
10. S. Pascarella and P. Argos, "A data bank merging related protein structures and sequences," *Protein Engineering* **5**, 121–137 (1992).
11. S. T. Rao and M. G. Rossmann, "Comparison of super-secondary structures in proteins," *J. Molecular Biology* **76**, 241–256 (1973).
12. A. Šali and J. P. Overington, "Derivation of rules for comparative protein modeling from a database of protein structure alignments," *Protein Science* **3**, 1582–1596 (1994).
13. W. R. Taylor and C. A. Orengo, "Protein structure alignment," *J. Molecular Biology* **208**, 1–22 (1989).
14. G. Vriend and C. Sander, "Detection of common three-dimensional substructures in proteins," *PROTEINS: Structure, Function, and Genetics* **11**, 52–58 (1991).