Geometric Suffix Tree: A New Index Structure for Protein 3-D Structures

Tetsuo Shibuya

Human Genome Center, Institute of Medical Science, University of Tokyo
4-6-1 Shirokanedai, Minato-ku, Tokyo 108-8639, Japan
tshibuya@hgc.jp

Abstract. Protein structure analysis is one of the most important research issues in the post-genomic era, and faster and more accurate query data structures for such 3-D structures are highly desired for research on proteins. This paper proposes a new data structure for indexing protein 3-D structures. For strings, there are many efficient indexing structures such as suffix trees, but it has been considered very difficult to design such sophisticated data structures against 3-D structures like proteins. Our index structure is based on the suffix trees and is called the geometric suffix tree. By using the geometric suffix tree for a set of protein structures, we can search for all of their substructures whose RMSDs (root mean square deviations) or URMSDs (unit-vector root mean square deviations) to a given query 3-D structure are not larger than a given bound. Though there are $O(N^2)$ substructures, our data structure requires only $O(N)$ space where N is the sum of lengths of the set of proteins. We propose an $O(N^2)$ construction algorithm for it, while a naive algorithm would require $O(N^3)$ time to construct it. Moreover we propose an efficient search algorithm. We also show computational experiments to demonstrate the practicality of our data structure. The experiments show that the construction time of the geometric suffix tree is practically almost linear to the size of the database, when applied to a protein structure database.

1 Introduction

Analyzing 3-D structures of proteins is very important in molecular biology and more and more protein structures are solved today with the aid of state-of-the-art technologies such as nuclear magnetic resonance (NMR) techniques, as seen in the increasing number of PDB [4] entries: 35,813 on March 28, 2006. It is said that structurally similar proteins tend to have similar functions even if their amino acid sequences are not similar to each other. Thus it is very important to find proteins with similar structures (even in part) from the growing database to analyze protein functions.

Structure similarity search methods for protein structure databases can be classified into two types. One is by comparing each database entry with the query. There are many comparison algorithms for protein structures [10], and the results could be very accurate, but it will require enormous amount of time

M. Lewenstein and G. Valiente (Eds.): CPM 2006, LNCS 4009, pp. 84–93, 2006.

to apply against very large databases. The other approach is by indexing with some important features of structures [1, 3, 6, 5, 8, 12]. In ordinary, these methods can search queries more efficiently, but with less accuracy than the pairwise comparison-based methods. The accuracy of comparison of two protein structures is often measured by RMSD (root mean square deviation) [2, 9, 17] or sometimes by URSMD (unit-vector root mean square deviation) [7, 15]; see section 2.1 for more details. But it has been considered too difficult to design indexing structures that strictly consider the RMSD or the URMSD.

In this paper, we propose a new data structure called the geometric suffix tree that succeeds in finding all the substructures whose RMSD or URMSD to a query is not larger than some given bound. As the name implies, our data structure is very similar to the famous suffix tree for character strings: The edges in the ordinary suffix tree represent substrings of texts, while the edges in the geometric suffix tree represent 3-D substructures of protein 3-D structures. The geometric suffix tree can be stored in $O(N)$ space where N is the sum of the lengths of the proteins in the database. We propose an $O(N^2)$ construction algorithm for it, though it takes $O(N^3)$ time if we construct the data structure naively. Furthermore, the experiments will show that the construction time of the geometric suffix tree is almost linear to the size of the database in practice, when applied to a protein structure database. Moreover, we propose an efficient search algorithm for substructure queries. This data structure is also useful for finding structural motifs, clustering substructures, and so on.

Organization of this paper is as follows. In section 2, we explain related work as preliminaries. In section 3, we describe definitions of two data structures: the geometric trie and the geometric suffix tree, where the geometric trie is the basis for the geometric suffix tree. In sections 4 and 5, we explain algorithms for constructing the data structure and algorithms for searching queries. In section 6, we demonstrate experimental results. In section 7, we conclude our results.

2 Related Work

2.1 RMSD and URMSD

A protein is a chain of amino acids. Each amino acid has one unique carbon atom named C_α, and we often use the coordinates of the C_α atom as the representative position of the amino acid. The set of C_α atom positions of all the amino acids in a protein is called the backbone of the protein, and is often used to ease protein structure analysis in previous work. The backbone is topologically linear, but it forms a geometrically very complex structure in the 3-D space. In this paper, we consider the backbone as the target to index.

The most popular and basic measure to determine geometric similarity between two sets of points like the positions of backbone atoms is the RMSD (root mean square deviation) [2, 9, 17], if we know which atom in one structure corresponds to which atom in the other. The measure describes the similarity of two structures when one of the point sets is rotated and translated reasonably. Let the two sets of points to be compared be $P = \{\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_n\}$ and

$Q = \{\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_n\}$, where $\boldsymbol{p}_i$ and $\boldsymbol{q}_j$ are coordinates in the 3-D space, and we consider $\boldsymbol{p}_i$ corresponds to $\boldsymbol{q}_i$ for each i. The RMSD is the minimum value of $\{(\sum_{i=1}^{n} \|\boldsymbol{p}_i - (R \cdot \boldsymbol{q}_i + \boldsymbol{v})\|^2)/n\}^{1/2}$ over possible rotation matrices R and translation vectors $\boldsymbol{v}$, where $\|\cdot\|$ denotes the norm. Let $\hat{R}(P,Q)$ and $\hat{\boldsymbol{v}}(P,Q)$ be R and $\boldsymbol{v}$ that minimizes the value. We call $\sum_{i=1}^{n} \|\boldsymbol{p}_i - (\hat{R}(P,Q) \cdot \boldsymbol{q}_i + \hat{\boldsymbol{v}}(P,Q))\|^2$ the MSSD (minimum sum squared distance) of P and Q.

It is known that $\hat{\boldsymbol{v}}(P,Q) = \sum_{i=1}^{n} (\boldsymbol{p}_i - \hat{R}(P,Q) \cdot \boldsymbol{q}_i)/n$, *i.e.*, the distance is minimized when the centroids of the two point sets are translated to the same point. Hence, if both of the point sets are translated so that their centroids are located at the origin of the coordinates, the RMSD/MSSD problem is reduced to a problem of finding R that minimizes $f(R) = \sum_{i=1}^{n} \|\boldsymbol{p}_i - R \cdot \boldsymbol{q}_i\|^2$. We can find $\hat{R}(P,Q)$ in linear time by using singular value decomposition (SVD) [2, 17] as follows. Let $H = \sum_{i=1}^{n} \boldsymbol{p}_i \cdot \boldsymbol{q}_i^t$. Then $f(R)$ can be described as $\sum_{i=1}^{n} (\boldsymbol{p}_i^t \boldsymbol{p}_i + \boldsymbol{q}_i^t \boldsymbol{q}_i) - trace(R \cdot H)$, and $trace(RH)$ is maximized when $R = VU^T$, where $U \Lambda V$ is the SVD of H. Hence $\hat{R}(P,Q)$ can be obtained in constant time from H (see [13] for SVD algorithms). Note that there are rare degenerate cases where $det(VU^T) = -1$, which means that VU^T is a reflection matrix. We ignore the degenerate cases in this paper. In this way, we can compute the RMSD/MSSD values in $O(n)$ time.

The URMSD (unit-vector root mean square deviation) [7, 15] is a variation of the RMSD. The RMSD is sometimes influenced badly by very distant pairs of points, and the URMSD is designed to avoid such influence. It is the minimum value of $\{(\sum_{i=1}^{n-1} \|\boldsymbol{p}'_i - R \cdot \boldsymbol{q}'_i\|^2)/(n-1)\}^{1/2}$ over possible rotation matrices R, where $\boldsymbol{p}'_i = (\boldsymbol{p}_{i+1} - \boldsymbol{p}_i)/\|\boldsymbol{p}_{i+1} - \boldsymbol{p}_i\|$ and $\boldsymbol{q}'_i = (\boldsymbol{q}_{i+1} - \boldsymbol{q}_i)/\|\boldsymbol{q}_{i+1} - \boldsymbol{q}_i\|$. Let $\check{R}(P,Q)$ be R that minimizes the value. We call $\sum_{i=1}^{n-1} \|\boldsymbol{p}'_i - \check{R}(P,Q) \cdot \boldsymbol{q}'_i\|^2$ the UMSSD (unit-vector minimum sum squared distance). The URMSD/UMSSD can be computed with the same strategy in $O(n)$ time, *i.e.*, by computing the SVD of $H' = \sum_{i=1}^{n} \boldsymbol{p}'_i \cdot (\boldsymbol{q}'_i)^t$.

2.2 Suffix Trees

The suffix tree [11, 14, 16, 19, 20] of a string $S \in \Sigma^n$ is the compacted trie of all the suffixes of $S^+ = S\$$ where \$ is a character such that $\$ \notin \Sigma$. This data structure can be stored in $O(n)$ space and moreover is known to be buildable in $O(n)$ time. Each leaf represents a suffix of the string S^+, and each node represents some substring. This data structure is very useful for various problems in sequence pattern matching. Using it, we can query a substring of length m in $O(m)$ time, we can find frequently appearing substrings in a given sequence in linear time, we can find a common substring of many sequences in linear time, and so on [14].

Not much work has been done for applying this data structure to biomolecular structures. The PSIST [12] is the only index data structure for protein structures based on the suffix trees as far as we know. It converts local features of the amino acid chain (*i.e.*, some feature vectors computed from only several adjacent atoms) into some alphabets and constructs suffix trees over the converted alphabet sequences, without considering global similarity measures like the RMSD or the

URMSD at all. For RNA (secondary) structures, the s-suffix tree [18], a generalization of the suffix tree, can be used for mining some interesting RNA structures from sequence databases, but it cannot be applied to protein 3-D structures.

3 Geometric Suffix Tree Data Structure

In this section, we describe the definition of the geometric suffix tree. Before defining the geometric suffix tree, we define a data structure called the geometric trie for a set of protein structures.

Consider a set of n protein structures represented by the sequence of their C_α atom coordinates. Let W_i be the i-th structure, where the 3-D coordinates of the j-th C_α atom is denoted as $\boldsymbol{w}_j^{(i)}$, and let ℓ_i be the length of W_i (*i.e.*, number of C_α atoms). Let $W_i[j..k]$ denote $\{\boldsymbol{w}_j^{(i)}, \boldsymbol{w}_{j+1}^{(i)}, \ldots, \boldsymbol{w}_k^{(i)}\}$, which means a structure formed by the $(k-j+1)$ atoms from the j-th atom to the k-th atom in W_i. We call it a substructure of W_i. Moreover, we call $W_i[1..j]$ a prefix substructure of W_i. Conversely, $W_i[j..\ell_i]$ is called a suffix substructure. From now on, we define two versions of the geometric trie: one based on the RMSD/MSSD (which we call the RMSD Geometric trie (RGT)) and the other based on the URMSD/UMSSD (which we call the URMSD geometric trie (UGT)). The geometric trie for the set of protein structures is defined as a rooted tree data structure that has following features:

1. All the internal nodes (nodes other than the leaves) except for the root have more than one child, while the root has only one child. (It corresponds to the fact that a structure with only one atom is always the same structure.) The trie has n leaves, each of which corresponds to one protein structure, and no two leaves correspond to the same structure. Let $leaf(i)$ denote the leaf that corresponds to W_i.
2. All the internal edges (*i.e.*, edges that end at internal nodes) and some external edges (*i.e.*, edges that end at leaves) correspond to a substructure of some protein. If the corresponding substructure of edge e is $P(e) = W_i[j..k]$, we represent it with only three values: i, j, and $length(e) = k - j + 1$. Let $length(e) = 0$ if e is an external edge without a corresponding substructure. We call the value $length(e)$ the edge length of e. Let the $depth(v)$ be the sum of all the edge lengths on the path from the root to v, which we call the depth of v.
3. Add to the three values that represent its corresponding substructure, each edge with a corresponding substructure has information of a rotation matrix $R(e)$ and a translation vector $\boldsymbol{v}(e)$. $R(e)$ and $\boldsymbol{v}(e)$ must satisfy the condition in the items 4 and 5.
4. Let $S(e)$ be a 3-D structure obtained by rotating $P(e)$ with $R(e)$ and translating it with $\boldsymbol{v}(e)$ after that. We call $S(e)$ the 'edge structure' of e. Note that $S(e)$ (not $P(e)$) corresponds to the substring represented by an edge in an ordinary suffix tree for alphabet strings. The 'node structure' $S(x)$ for a node x is defined as a structure that can be obtained by concatenating 'edge

structures' of the edges on the path from the root to the node x. For any leaf $v = leaf(i)$ and its node structure $S(v)$, the MSSD (in case of RGTs, or the UMSSD in case of UGTs) between any prefix substructure of $S(v)$ and the prefix substructure of W_i of the same length must not be larger than some given fixed bound b. (Note that b is unrelated to the RMSD/URMSD bound d used in the next section for searching structures.)

5. For an edge $e = (v, w)$ with some corresponding substructure $P(e)$, the 'branching structure' $str(e)$ is defined as a structure that is obtained by adding the coordinates of the first atom of $S(e)$ (*i.e.*, $S(e)[1]$) after the coordinates sequence $S(v)$. For any internal node v with more than one outgoing edge with corresponding substructures, the MSSD (for RGTs, or the UMSSD for UGTs) between $str(e_1)$ and $str(e_2)$ must be larger than b, where e_1 and e_2 are arbitrary two of the edges.

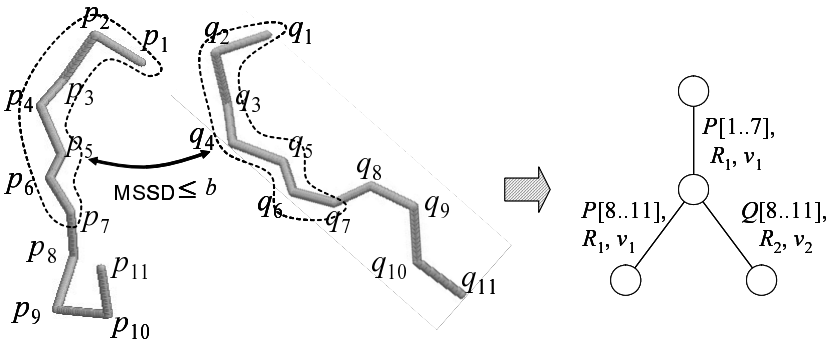

Fig. 1. A geometric trie for two protein 3-D structures

As there are only $O(n)$ nodes/edges in the trie and we need only $O(1)$ memory for each edge/node, the total memory space to store the geometric trie is only $O(n)$. Note that the data structure is not unique for a fixed set of protein structures. Figure 1 shows an example of the geometric trie constructed for two structures P and Q. In the figure, we consider the MSSD of $P[1..7]$ and $Q[1..7]$ is not larger than b, while the MSSD between $P[1..8]$ and $Q[1..8]$ is larger than b.

Now we can define the geometric suffix tree: The geometric suffix tree for a set of proteins is the geometric trie for all the suffix substructures of all the proteins in the set. It is easy to see that we need $O(N)$ space to store the geometric suffix tree, where N is the sum of the lengths of the proteins.

4 Constructing Geometric Suffix Trees

In this section, we describe how to construct the geometric tries and the geometric suffix trees. Given a set of n protein structures W_i and some given MSSD (for RGTs or UMSSD for UGTs) bound b, we can construct the geometric trie by adding structures one by one as follows:

Algorithm 1. At first, construct a tree with only the root node. For each protein structure W_i, set the root node to v and do the following.

1. From among a set of v's outgoing edges with some corresponding substructures, find an edge e such that the MSSD (for RGTs, or the UMSSD for UGTs) between $W_i[1..\mathit{depth}(v)+1]$ and $\mathit{str}(e)$ is smaller than b. If there is more than one such edge, choose an arbitrary one (or preferably the one with the smallest MSSD (or UMSSD)). If no such edge exists, go to step 2. Otherwise go to step 3.
2. Add a new outgoing edge $e' = (v, w)$ to v, and let the new leaf w correspond to W_i. Let $P(e')$ be $W_i[\mathit{depth}(v)+1..\ell_i]$. If v is the root, let $R(e')$ be the identity matrix and let $\boldsymbol{v}(e')$ be a zero vector. Otherwise, in case of RGTs, let $R(e')$ be $\hat{R}(S(v), W_i[1..\mathit{depth}(v)])$, and let $\boldsymbol{v}(e')$ be $\hat{\boldsymbol{v}}(S(v), W_i[1..\mathit{depth}(v)])$. In case of UGTs, let $R(e')$ be $\check{R}(S(v), W_i[1..\mathit{depth}(v)])$, and let $\boldsymbol{v}(e')$ be $(S(v)[\mathit{depth}(v)] - R(e') \cdot W_i[\mathit{depth}(v)])$. Notice that, in both cases, $R(e')$ and $\boldsymbol{v}(e')$ represents alignment between $S(v)$ and $W_i[1..\mathit{depth}(v)]$. Then stop.
3. Let w be the node where the edge e ends. Find the longest prefix substructures of $S(w)$ and W_i whose MSSD (for RGTs, or UMSSD for UGTs) is not larger than b, and let the length be ℓ. If $\ell < \mathit{depth}(w)$ go to step 4. If $\ell = \mathit{depth}(w)$ and $\ell < \ell_i$, set w to v and go to step 1. Otherwise, add a new outgoing edge (w, u) with no corresponding substructure, and let the new leaf u correspond to the structure W_i. Then stop.
4. Insert a new node u between v and w. Let $e_1 = (v, u)$ and let $e_2 = (u, w)$. Let $P(e_1)$ be the prefix substructure of $P(e)$ of length $(\ell - \mathit{depth}(v))$, and $P(e_2)$ be the suffix substructure of $P(e)$ of length $(\mathit{depth}(w) - \ell)$. Let $R(e_1)$ and $R(e_2)$ be the same matrix as $R(e)$, and $\boldsymbol{v}(e_1)$ and $\boldsymbol{v}(e_2)$ be the same vector as $\boldsymbol{v}(e)$. Add a new outgoing edge $e'' = (u, x)$ to u, and let the new leaf x correspond to the structure W_i. If $\ell = \ell_i$, let e'' have no corresponding substructure. Otherwise, let the corresponding substructure $P(e'')$ be $W_i[\ell+1..\ell_i]$. In case of RGTs, let $R(e'')$ be $\hat{R}(S(u), W_i[1..\ell])$ and let $\boldsymbol{v}(e'')$ be $\hat{\boldsymbol{v}}(S(u), W_i[1..\ell])$. In case of UGTs, let $R(e'')$ be $\check{R}(S(u), W_i[1..\ell])$ and let $\boldsymbol{v}(e'')$ be $(S(u)[\ell] - R(e'') \cdot W_i[\ell])$. Then stop.

With the same algorithm, we can construct the geometric suffix tree: Just consider that W_i is the i-th suffix substructure.

Recall that it takes $O(\ell)$ time to compute the MSSD or the UMSSD (and the rotation matrix and the translation vector related to it) between two structures of size ℓ. Thus, if we execute the algorithm naively, we would need $O((\ell_i + n) \cdot \ell_i)$ time to add W_i to the tree, because there are at most n branches on the path from the root node to some leaf. Accordingly, we need $O(\sum_i^n \{(\ell_i + n) \cdot \ell_i\})$ time for constructing the geometric trie. It means that the above algorithm requires $O(N^3)$ time to construct the geometric suffix tree, where N is the sum of all the structure lengths. From now on we present how to reduce it to $O(N^2)$.

We reduce the computation time by proposing an incremental MSSD/UMSSD computation technique. Recall that the MSSD of two protein structures $P[1..j]$ and $Q[1..j]$ can be obtained by computing the SVD of $H =$

$\sum_{i=1}^{j} (\boldsymbol{p}_i - \boldsymbol{c}_P) \cdot (\boldsymbol{q}_i - \boldsymbol{c}_Q)^t$ where $\boldsymbol{c}_P$ and $\boldsymbol{c}_Q$ are the centroids of P and Q. H can be computed in constant time if we are given $f_P(j) = \sum_i^j \boldsymbol{p}_i$, $f_Q(j) = \sum_i^j \boldsymbol{q}_i$, and $g(j) = \sum_i^j \boldsymbol{p}_i \cdot \boldsymbol{q}_i^t$, as $H = g(j) - \{f_P(j) \cdot (f_Q(j))^t\}/j$. Add to these values, we need $h_P(j) = \sum_{i=1}^{j} \boldsymbol{p}_i^t \boldsymbol{p}_i$ and $h_Q(j) = \sum_{i=1}^{j} \boldsymbol{q}_i^t \boldsymbol{q}_i$ to compute the MSSD or RMSD values in constant time. Notice that all of these can be computed incrementally in constant time from $f_P(j-1)$, $f_Q(j-1)$, $g(j-1)$, $h_P(j-1)$, and $h_Q(j-1)$. It means that we can add the structure W_i to the tree in $O(\ell_i+n)$ time, and accordingly we can construct the RGT in $O(N+n^2)$ time. In conclusion, we can construct the RMSD-based geometric suffix tree in $O(N^2)$ time.

Similarly, we can compute the UMSSD of two protein structures $P[1..j]$ and $Q[1..j]$ in constant time if we are given $g'(j) = \sum_{i=1}^{j} \boldsymbol{p}'_i \cdot (\boldsymbol{q}'_i)^t$, $h'_P(j) = \sum_{i=1}^{j} (\boldsymbol{p}'_i)^t \boldsymbol{p}'_i$, and $h'_Q(j) = \sum_{i=1}^{j} (\boldsymbol{q}'_i)^t \boldsymbol{q}'_i$. We can easily see that these can also be computed from $g'(j-1)$, $h'_P(j-1)$ and $h'_Q(j-1)$ in constant time. Therefore we conclude that the UGTs and the URMSD-based geometric suffix trees can be constructed in the same time bound as the RGTs and the RMSD-based geometric suffix trees: We can construct the UGTs in $O(N+n^2)$ time, and the URMSD-based geometric suffix trees in $O(N^2)$ time.

5 Geometric Suffix Tree Applications

There are two important features on the RMSD/MSSD (or URMSD/UMSSD) measures. One is that the MSSD (or UMSSD) of two structures P and Q (of the same length) is always larger than or equal to that of P' and Q', where P' and Q' are any same-length prefix substructures of P and Q. The other is that there is a triangle inequality $c \leq a + b$ where a is the RMSD (URMSD) between P and Q, b is that between Q and R, and c is that between R and P, for any set of three structures P, Q, and R of same lengths.

Using these features, all maximal substructures whose RMSD (or URMSD) to a query $Q[1..m]$ is within some bound d can be computed efficiently as follows. Let 'representative structure' mean any prefix substructure of the 'node structure' of any node in the geometric suffix tree. First, we find all the maximal representative substructures whose RMSD (or URMSD) to the query Q is within $\sqrt{b/m}+d$ by just doing a depth-first or breadth-first search from the root, where b is the MSSD (or UMSSD) bound used for constructing the geometric suffix tree. Let E be the set of edges to which the collected representative substructures correspond. After that, find all the leaves that are descendants of the edges in E. As the suffixes that correspond to the collected leaves are candidates of the answer substructures (and there are no candidates elsewhere), check their RMSDs (or URMSDs) one by one.

Ordinary suffix trees have tremendous number of applications in string pattern matching [14]. Like them, applications of the geometric suffix trees are not limited to the database search. A long representative structure whose corresponding edge has many descendants is a repeated structure in a protein structure, which could have some meaning. By constructing the geometric suffix tree for several functionally-related protein structures, we could find structural motifs. We could

further use this fundamental data structure for designing more complicated combinatorial pattern matching algorithms on protein structures, such as structural alignment algorithms, clustering/classification algorithms and functional prediction algorithms.

6 Experimental Results

In this section, we demonstrate the performance of the geometric suffix trees through experiments on a Sun Fire 15K super computer with 288 GB memory and 96 UltraSPARC III Cu CPUs running at 1.2GHz. Note that we used only one CPU for each experiment. As a data for experiments, we used a set of 228 myoglobin or myoglobin-related PDB data files containing 275 protein structures. The total number of amino acids in the protein set is 41,719.

Table 1 shows the computation time for constructing the RMSD-based geometric suffix trees against databases of different sizes, setting 400Å^2 to the MSSD bound. In the experiment (1), we used all the 275 proteins to index. In the experiments (2)-(5), we used different subsets of them. The '#sequence(#a.a.)' column shows the numbers of sequences and amino acids contained in the protein sets. The 'Time' column shows the computation time, while the 'GST Size' column shows the numbers of nodes in the constructed geometric suffix trees. According to the table, the computation time is almost linear to the size of the databases, though the theoretical time bound is $O(N^2)$. It is reasonable as there should be some reasonable upper bound on protein lengths.

Next, we examined the query speed on the RMSD-based geometric suffix trees with different MSSD bounds. Table 2 shows the results, where '$b = \ldots$' denotes the MSSD bound in Å^2. We used two protein substructures of the same length as queries: In experiment (a), we used as a query a substructure from

Table 1. Time for constructing the geometric suffix trees ($b = 400\text{Å}^2$)

Database	#sequence	(#a.a.)	Time (sec)	GST Size
(1) Entire database	275	(41,719)	53.15	57,241
(2) Subset A	198	(30,061)	36.37	41778
(3) Subset B	111	(16,983)	17.68	25,942
(4) Subset C	54	(8,267)	7.91	13,050
(5) Subset D	20	(3,069)	2.89	4,855

Table 2. Query time (sec) on the geometric suffix trees with various MSSD bounds

Queries		$b = 1$	$b = 100$	$b = 400$	$b = 900$	$b = 1600$	$b = 2500$	#found
(a)	$d = 1.0$Å	1.63	0.56	0.39	0.43	0.60	0.87	19
	$d = 5.0$Å	11.70	5.08	5.66	6.55	6.63	6.63	217
(b)	$d = 1.0$Å	1.63	0.73	0.48	0.33	0.19	0.21	0
	$d = 5.0$Å	16.13	7.83	7.93	8.00	7.58	7.20	0

the 20th amino acid to the 69th amino acid of a myoglobin's structure obtained from the PDB entry named 103M. In experiment (b), we used a protein that is unrelated to myoglobins: A substructure from the 20th amino acid to the 69th amino acid of a rhodopsin's structure obtained from the PDB entry named 1F88. In both experiments, we examined query time by setting two different RMSD bounds: $d = 1.0$Å and $d = 5.0$Å. In the table, the '#found' column shows the numbers of found substructures similar to the query. According to the experiments, the query is very fast when the RMSD bound for the query is small in both experiments. Note that we can observe similar phenomenon on ordinary suffix trees: It is known that the inexact matching on suffix trees is (not) efficient when there is (not) a small edit distance limit.

7 Concluding Remarks

We proposed a new data structure called the geometric suffix tree for indexing the protein 3-D structures. The data structure can be stored in $O(N)$ space where N is the database size, and we presented an $O(N^2)$ construction algorithm for it. Moreover, we showed through experiments that we can build the data structure in quasi-linear time in practice. We also showed that we can search for queries very efficiently with the geometric suffix tree.

It is an open problem whether we can improve the theoretical time bound for building the geometric suffix tree. We are now working on utilizing this data structure for further combinatorial matching problems and machine learning problems on protein structures. We suppose this work is just the beginning.

Acknowledgements

The author would like to thank Prof. Tatsuya Akutsu for fruitful discussions on protein comparison algorithms. All the computational experiments in this research were done on the Super Computer System, Human Genome Center, Institute of Medical Science, University of Tokyo.

References

1. T. Akutsu, K. Onizuka, and M. Ishikawa. New hashing techniques and their application to a protein database system. *Proc. Hawaii Int. Conf. System Sciences (HICSS-28)*, Vol. 5, pp. 197-206, 1995.
2. K. S. Arun, T. S. Huang, and S. D. Blostein. Least-squares fitting of two 3-D point sets. *IEEE Trans Pattern Anal. Machine Intell.*, Vol. 9, pp. 698-700, 1987.
3. Z. Aung, W. Fu and K. Tan. An efficient index-based protein structure database searching method. *Proc. Intl. Conf. on Database Systems for Advanced Applications*, pp. 311-318, 2003.
4. H. M. Berman, J. Westbrook, Z. Feng, et al. The protein data bank. *Nucl. Acids Res.*, Vol. 28, pp. 235-242, 2000.

5. O. Çamoğlu, T. Kahveci and A. Singh. Towards index-based similarity search for protein structure databases. *IEEE Computer Society Bioinformatics Conference*, pp. 148-158, 2003.
6. T. Can and Y. Wang. CTSS: a robust and efficient method for protein structure alignment based on local geometrical and biological features. *IEEE Computer Society Bioinformatics Conference*, pp. 169-179, 2003.
7. L. P. Chew, D. Huttenlocher, K. Kedem and J. Kleinberg. Fast detection of common geometric substructure in proteins. *J. Comput. Biol.*, Vol. 6, No. 3, pp. 313-325, 1999.
8. I. Choi, J. Kwon and S. Kim. Local feature frequency profile: A method to measure structural similarity in proteins. *Proc. Natl. Acad. Sci.*, Vol. 101, No. 11, pp. 3797-3802, 2004.
9. D. W. Eggert, A. Lorusso and R. B. Fisher. Estimating 3-D rigid body transformations: a comparison of four major algorithms. *Machine Vision and Applications*, Vol. 9, pp. 272-290, 1997.
10. I. Eidhammer, I. Jonassen, and W. R. Taylor. Structure Comparison and Structure Patterns. *J. Computational Biology*, Vol. 7, No. 5, pp. 685-716, 2000.
11. M. Farach. Optimal suffix tree construction with large alphabets. *Proc. 38th IEEE Symp. Foundations of Computer Science,* pp. 137-143, 1997.
12. F. Gao and M. J. Zaki. PSIST: Indexing Protein Structures using Suffix Trees. *Proc. IEEE Computational Systems Bioinformatics Conference (CSB)*, pp. 212-222, 2005.
13. G. H. Golub and C. F. Van Loan. *Matrix Computation*. 3rd eds., John Hopkins University Press, 1996.
14. D. Gusfield. *Algorithms on strings, trees, and sequences: computer science and computational biology,* Cambridge University Press, 1997.
15. K. Kedem, P. Chew and R. Elber. Unit-vector RMS (URMS) as a tool to analyze molecular dynamics trajectories. *Proteins: Struct. Funct. Genet.*, Vol. 38, pp. 1-12, 1999.
16. E. M. McCreight. A space-economical suffix tree construction algorithm. *J. ACM.*, Vol. 23, pp. 262-272, 1976.
17. J. T. Schwartz and M. Sharir. Identification of partially obscured objects in two and three dimensions by matching noisy characteristic curves. *Intl. J. of Robotics Res.*, Vol. 6, pp. 29-44, 1987.
18. T. Shibuya. Generalization of a suffix tree for RNA structural pattern matching. *Algorithmica*, Vol. 39, No. 1, pp. 1-19, 2004.
19. E. Ukkonen. On-line construction of suffix-trees. *Algorithmica,* Vol. 14, pp. 249-260, 1995.
20. P. Weiner. Linear pattern matching algorithms. *Proc. 14th Symposium on Switching and Automata Theory*, pp. 1-11, 1973.