8-7-2008

Robust and Efficient Algorithms for Protein 3-D Structure Alignment and Genome Sequence Comparison

Zhiyu Zhao
University of New Orleans

Robust and Efficient Algorithms for Protein 3-D Structure Alignment and Genome Sequence Comparison

A Dissertation

Submitted to the Graduate Faculty of the
University of New Orleans
in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy
in
Engineering and Applied Science
Computer Science

by

Zhiyu Zhao

B.E. Huazhong University of Science and Technology, 1997
M.E. Huazhong University of Science and Technology, 2000
M.S. University of New Orleans, 2006

August, 2008

Acknowledgements

I would like to give my sincere thanks to all the people who gave me the opportunity to complete this Ph.D. dissertation.

I appreciate the kind guidance and supervision of my dissertation advisers Dr. Bin Fu and Dr. Christopher M. Summa. Dr. Fu's intelligent ideas and strict attitude toward academic research have impressed me deeply. Dr. Summa always brought me interesting biologic problems and inspired me to discover the connections between biology and computation. I believe I will benefit from what I have learned from them in my whole life.

I express my deep gratitude to Dr. Mahdi Abdelguerfi who cared much about my research, my life and my family during the whole period of my Ph.D. study, and to Dr. Shengru Tu who always gave me valuable advices and kind help.

I would like to acknowledge all my co-researchers for their contributions to the research and publications which form the foundation of this dissertation. They are Dr. Bin Fu, Dr. Christopher M. Summa, Dr. Zhixiang Chen, Dr. Boting Yang, Dr. Binhai Zhu, Dr. Jinhui Xu, Dr. Robert Schweller, Francisco J. Alanis, Zaixin Lu and Sergio Garcia.

I am grateful to all the professors in my dissertation committee for their precious suggestions about my Ph.D. research and valuable comments on my dissertation. Besides my advisers, I want to thank Dr. Stephen Winters-Hilt, Dr. Huimin Chen and Dr. Dongxiao Zhu. I especially thank Dr. Seth H. Pincus who has initialized our research interest in protein structure analysis.

In addition, I would like to thank Dr. Diego Liberati who led me into the research field of Bioinformatics when I studied in the Politecnico di Milano University, Italy from 2004 to 2005, and Dr. Zhixiang Chen who greatly supported my research and life when I visited the University of Texas-Pan American in 2007 and 2008. Thank Ms. Jeanne Boudreaux, our department secretary, and Ms. Zella Huaracha, the doctoral program coordinator, for their hard work.

Finally, I appreciate the understanding and encouragement of my husband Liqiang Wang, my parents and my brother's family. Without their care throughout these years it would be impossible for me to complete this Ph.D. study.

Table of Contents

List of Figures

List of Tables

Abstract

Sequence analysis and structure analysis are two of the fundamental areas of bioinformatics research. This dissertation discusses, specifically, protein structure related problems including protein structure alignment and query, and genome sequence related problems including haplotype reconstruction and genome rearrangement. It first presents an algorithm for pairwise protein structure alignment that is tested with structures from the Protein Data Bank (PDB). In many cases it outperforms two other well-known algorithms, DaliLite and CE. The preliminary algorithm is a graph-theory based approach, which uses the concept of "stars" to reduce the complexity of clique-finding algorithms. The algorithm is then improved by introducing "double-center stars" in the graph and applying a self-learning strategy. The updated algorithm is tested with a much larger set of protein structures and shown to be an improvement in accuracy, especially in cases of weak similarity. A protein structure query algorithm is designed to search for similar structures in the PDB, using the improved alignment algorithm. It is compared with SSM and shows better performance with lower maximum and average Q-score for missing proteins. An interesting problem dealing with the calculation of the diameter of a 3-D sequence of points arose and its connection to the sublinear time computation is discussed. The diameter calculation of a 3-D sequence is approximated by a series of sublinear time deterministic, zero-error and bounded-error randomized algorithms and we have obtained a series of separations about the power of sublinear time computations. This dissertation also discusses two genome sequence related problems. A probabilistic model is proposed for reconstructing haplotypes from SNP matrices with incomplete and inconsistent errors. The experiments with simulated data show both high accuracy and speed, conforming to the theoretically provable efficiency and accuracy of the algorithm. Finally, a genome rearrangement problem is studied. The concept of non-breaking similarity is introduced. Approximating the exemplar non-breaking similarity to factor $n^{1-\epsilon}$ is proven to be NP-hard. Interestingly, for several practical cases, several polynomial time algorithms are presented.

Keywords Sequence Analysis, Structure Analysis, Protein Structure Alignment, Protein Structure Query, Haplotype Reconstruction and Genome Rearrangement.

Chapter 1

Introduction

Molecular biology and computer science are areas where truly revolutionary events and developments took place during the twentieth century. Such developments have made possible the newer discipline of bioinformatics that is now attracting many researchers from both the computational and the biological communities [35]. Technological advances in biological and biomedical research including high-throughput DNA sequencing and 3-D protein structure determination have revolutionized the development of these traditional disciplines, and the new technologies have produced an enormous amount of data with valuable information for understanding the underlying biology. Biologists now depend more and more on computers to store and analyze this data. On one hand, public repositories such as the Protein Data Bank (PDB) [1] and the GenBank [2] are valuable resources of protein structure and genomic sequence data, respectively. On the other hand, their surprising growth rate necessitates collaboration between biologists and computer scientists; biologists need efficient computer software tools to help extract useful information from the data, for instance, to make inferences about DNA sequences and to compare protein structure similarities.

Bioinformatics has grown into a large inter-disciplinary pursuit, and two of its fundamental areas studied in this dissertation are sequence analysis and structure analysis. DNA/RNA sequence alignment, genome comparison and protein sequence alignment are instances of sequence analysis, while protein structure alignment and protein folding are typical research problems of structure analysis. DNA sequence determines protein sequence, protein sequence determines protein structure, and protein structure determines protein function [64]. Sequence analysis and structure analysis are closely related, but the latter is more difficult, and the methods applied in these subfields are very different, because sequence analysis deals with one-dimensional data, while structure analysis deals with the much more complex three-dimensional data.

1.1 Protein 3-D Structure

A protein is a large organic biopolymer composed of a linear chain of amino acids. An amino acid is a molecule that contains both amine and carboxyl functional groups. In biochemistry, an amino acid typically refers to an α - amino acid in which the amino and carboxylate groups are attached to the same carbon, which is called the α - carbon (C_α) [3]. There are 20 standard α - amino acids (see Table 1.1 for details). The various α - amino acids differ in which side chains (R group) are attached to their α carbons. Figure 1.1a shows

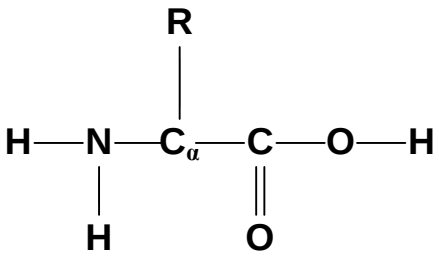
the general structure of an α - amino acid. As both the amine and carboxylic acid groups of amino acids can react to form amide bonds, one amino acid molecule can react with another and become connected through an amide linkage. This condensation reaction yields a newly formed peptide bond and a molecule of water, and the dehydrated amino acids are called amino acid residues [3]. See Figure 1.1b for the condensation of amino acids. When amino acids are linked one by one with the condensation reaction, they form a long polypeptide chain and it is through this process that natural proteins are synthesized *in vivo*. A protein molecule is made up of one or more such polypeptide chains.

Table 1.1: 20 standard α - amino acids

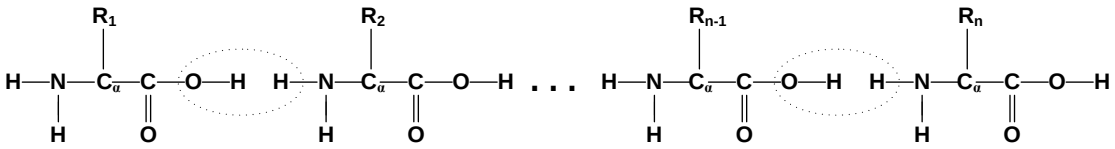
Amino Acid	3-Letter Abbreviation	1-Letter Abbreviation
Alanine	Ala	A
Arginine	Arg	R
Asparagine	Asn	N
Aspartic acid	Asp	D
Cysteine	Cys	C
Glutamic acid	Glu	E
Glutamine	Gln	Q
Glycine	Gly	G
Histidine	His	H
Isoleucine	Ile	I
Leucine	Leu	L
Lysine	Lys	K
Methionine	Met	M
Phenylalanine	Phe	F
Proline	Pro	P
Serine	Ser	S
Threonine	Thr	T
Tryptophan	Trp	W
Tyrosine	Tyr	Y
Valine	Val	V

The sequence of a naturally occuring protein (i.e. the linear string representing the identities of the individual amino acids) was shown by Anfinsen [11] to be sufficient to specify its unique 3-dimensional structrue. This is due to the differing shapes and physico-chemical properties of the individual amino acids in the sequence. It is this structure that ultimately determines the function of the protein, by arranging a complex set of atoms in a particular 3-D configuration to enable enzymatic catalysis, recognition of ligands or other protein binding partners, structural functions within cells, etc. The overall 3-D shape of a protein is often represented by a "backbone representation", which is simply a trace (or spline) connecting the centroids of the C_α atoms, with each C_α indicating the position of an amino acid residue. Figure 1.1c displays the 3-D structure of the backbone of protein GB1 (PDB ID: 2PLP), in which the C_α atoms are represented as balls connected by sticks.

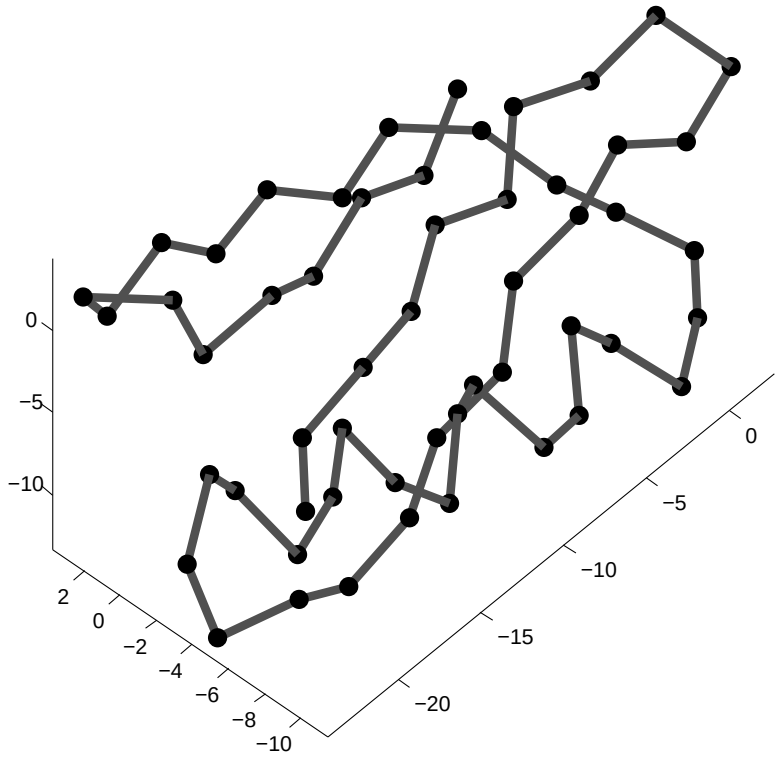
Protein 3-D structures are strongly related to their biological functions [80]. In DNA



(a) The general structure of an α - amino acid



(b) The condensation of α - amino acids



(c) A backbone representation of a protein chain with C_α atoms highlighted

Figure 1.1: 3-D structure of a protein molecule

the molecules comprising the alphabet are chemically similar, and the structure of DNA is approximately uniform. In contrast, proteins show a great variety of 3-D conformations. These are necessary to support their very diverse structural and functional roles [64]. The functions of proteins depend on their adopting the native 3-D structure. For example, the native structure of an enzyme may have a cavity on its surface that binds a small molecule and juxtaposes it to catalytic residues [64]. Protein structures reveal more evolutionary information than protein sequences do, since the structure of a protein changes more slowly in evolution than does its sequence [35]. Also, researchers frequently find that proteins with low sequential similarity are structurally homogenous. Therefore it is particularly important to discover the structural similarity / dissimilarity among different proteins. The research of protein 3-D structure similarity is very helpful for many biological applications such as predicting the functions of unknown proteins from known similar protein structures, identifying protein families with common evolutionary origins, understanding the variations among different classes of proteins, and so on.

Protein structures can be determined *via* experimental techniques such as X-ray crystallography and NMR (Nuclear Magnetic Resonance). The number of proteins discovered by biologists has increased dramatically over the last 30 years. There are three peaks in the PDB growth history [66]. The first peak occurred between 1972 and 1976 corresponding to the initial explosion of crystallography. The second peak is most likely due to the spread of Digital Equipment Company's VAX 780 virtual memory computers introduced in 1978. The third peak occurred between 1991 and 1997 corresponding to the availability of intense beams of X-rays from synchrotrons coupled with the use of crystals cooled in liquid nitrogen. Also, the development of a new field, structural genomics, contributes greatly to the growth in novel structural data since 2000. According to the statistics of the RCSB Protein Data Bank [1], as of March 2008, over 49,000 protein structures have been reported, and in these structures there are over 100,000 protein chains. In [66] a weighted count method was introduced to predict the growth rate of novel structures in the PDB as well as the size of Structural Classification of Proteins (SCOP) [4, 75] categories. The rapid growth of the Protein Data Bank (see Figure 2a of [66] for an illustration of the PDB growth rate from 1970's to the year 2005) necessitates the development of efficient protein structure comparison algorithms and automatic software tools.

1.2 Protein Structure Alignment

Protein structure alignment attempts to compare the structural similarity between proteins. A widely accepted method of protein 3-D structure comparison is to align the C_α atoms in the protein backbones. With this method, the problem of protein 3-D structure alignment can be solved by first determining the longest common sub-chains between different protein backbone chains, followed by a proper geometrical transformation which translates and rotates one protein backbone chain to align it as close as possible to the other one. An alignment is characterized by (1) how many positions are matched, (2) where these positions are, and (3) how well they are matched. (1) and (2) are available once an alignment is determined. For (3), a transformation based alignment algorithm usually calculates (C_α) $RMSD$, namely, the root mean square distance between aligned C_α atoms. Assume protein

backbone chains $S = p_1 \cdots p_m$ and $S' = q_1 \cdots q_n$, where each p_i $(1 \leq i \leq m)$ or q_j $(1 \leq j \leq n)$ is a C_α atom which is represented by a point in R^3. Assume that N_{mat} pairs of C_α atoms are aligned *via* a transformation F and the aligned positions are $p_{i_1} \cdots p_{i_{N_{mat}}}$ in S and $q_{j_1} \cdots q_{j_{N_{mat}}}$ in S', respectively. The (C_α) $RMSD$ between S and transformed S' is defined in Equation 1.1. In this dissertation, unless otherwise specified, an $RMSD$ always refers to a (C_α) $RMSD$. In the case of rigid body transformation, F is determined by a rotation matrix R and a translation vector T, and $F(p) = R \cdot p + T$ for any point p in R^3.

$$RMSD = \sqrt{\frac{1}{N_{mat}} \sum_{k=1}^{N_{mat}} \left(p_{i_k} - F(q_{j_k})\right)^2} \tag{1.1}$$

Figure 1.2 shows a pairwise structure alignment between chain E of protein 1ATP and protein 1PHK, where N_{mat} is the number of aligned C_α pairs, $RMSD$ is the root mean square distance between the aligned pairs, and the rigid body transformation used to align the two chains is the translation vector T and the rotation matrix R.

The alignment problem is non-trivial – in fact, the problem of finding the optimal global alignment between protein structures has been shown to be NP-hard [44, 63]. Therefore, there have been a number of protein structure alignment algorithms presented in the past years (e.g. [26, 36, 39, 54, 55, 58, 59, 65, 72, 79, 85, 86, 88, 89, 97–99, 101, 103]); among them: DALI (distance matrix based method) [54], SSM (secondary structure matching) [59], CE (the combinatorial extension method) [85] and FATCAT (protein structure alignment based on flexible transformation) [98] are commonly used. The distance matrix method, also called DALI, was proposed by Holm and Sander [54]. The distance matrix of each protein backbone contains all the distances between every two C_α atoms in its backbone chain. The 3-D structure comparison problem is converted into a problem of finding the best overlap of these 2-D matrices. The combinatorial extension (CE) method was proposed by Shindyalov and Bourne [85]. With the CE method, locally similar sub-chain pairs of fixed length are combined and extended to form an optimal global alignment between a pair of protein backbone chains. The secondary structure matching (SSM) method proposed by Krissinel and Henrick [59], first builds a graph based on the secondary structures of proteins and then iteratively align the backbone C_α atoms. Singh and Brutlag [86] introduced a method that represents the secondary structure elements as vectors and computes a local alignment of them *via* dynamic programming. Chew et al [26] proposed an approach that represents the backbone by a chain of unit-vectors and translates those vectors into a set of points on the unit-sphere. Yona and Kedem [99] used a hybrid method that combines unit vector spaces and root-mean square. A method called TOPFIT, introduced by Ilyin, Abyzov and Leslin [55], is based on the identification of the common volume subgraph of Delaunay triangulation. Some other methods include a method based on geometric hashing by Fischer, Nussiinov and Wolfson [39], a method to minimize the soap-bubble surface area between the backbones proposed by Falicov and Cohen [36], and the double-dynamic programming method of Orengo and Taylor [88, 89]. A method in which the backbone is represented by the angles of consecutive C_α atoms was presented by Ye, Janardan and Liu [97]. It generates the code for each C_α atom along the backbone by the angles with other two neighbor C_α atoms and the code is independent of the relative orientation of the two proteins. In [58]

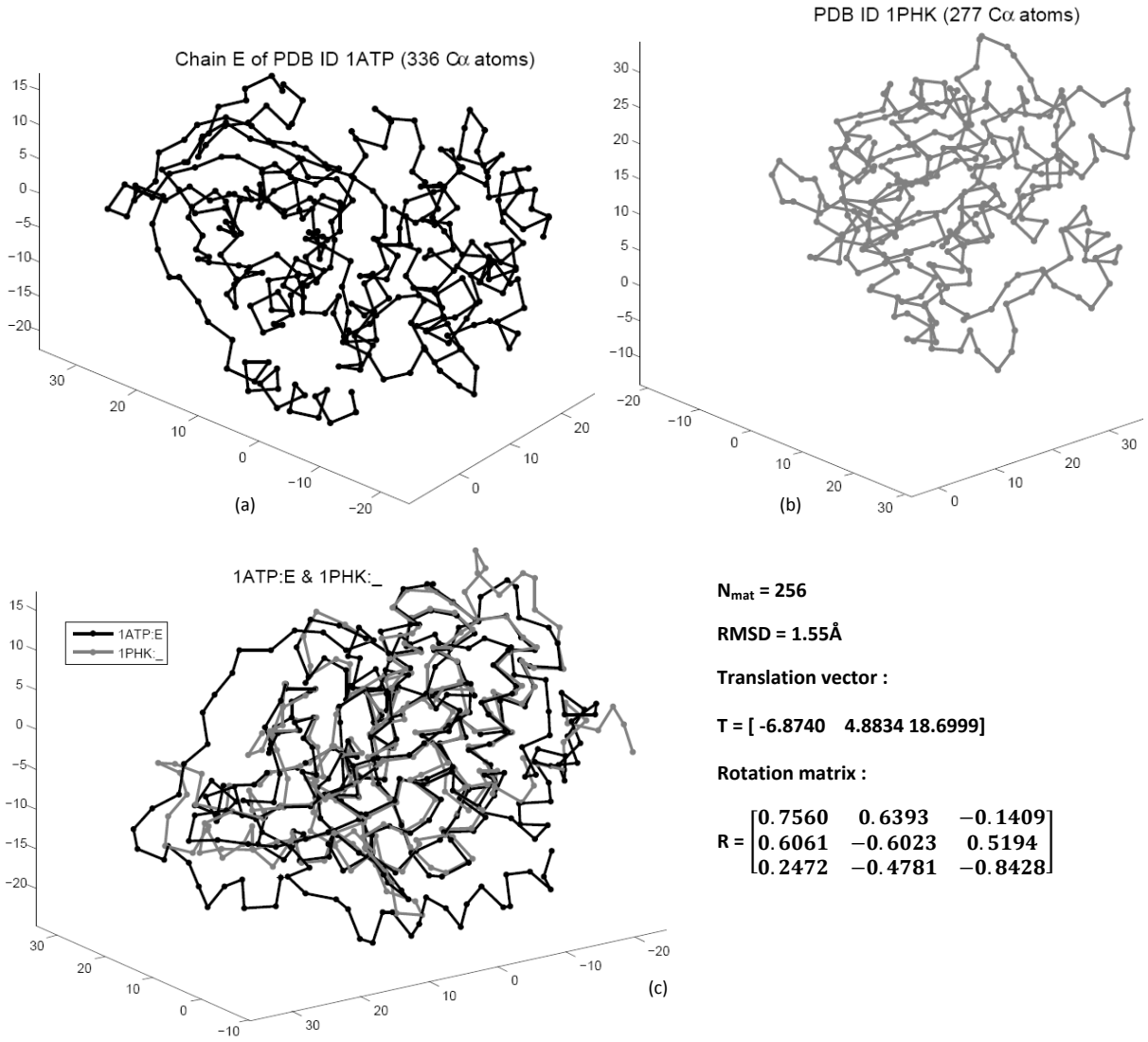


Figure 1.2: An example of pairwise protein structure alignment

Two protein chains are shown in their backbone view with dotted positions of C_α atoms. A rigid body transformation is applied to chain 2 (PDB ID: 1PHK) to align it with chain 1 (chain E of PDB ID: 1ATP). The transformation is determined by a translation vector T and a rotation matrix R. N_{mat} is the number of aligned C_α pairs and $RMSD$ is the root mean square distance between the aligned pairs.

Kolodny, Linial and Levitt studied the protein structure alignment problem as a family of optimization problems and developed an approximate polynomial time algorithm to solve them. Most protein structure alignment algorithms use a rigid body transformation, such as CE, DALI, SSM and algorithms discussed in this dissertation, while Ye and Godzik [98] proposed the FATCAT algorithm which aligns proteins *via* a flexible transformation. Although it has been studied for over 30 years, the protein structure alignment problem is far from being well resolved. New approaches and improvements to existing approaches are often proposed. Moreover, many questions are under active discussions. One of them is whether a computationally optimal solution to the problem is biologically significant. Even so, little consensus is given on how to score the significance of alignment results and how to evaluate the performance of algorithms.

1.3 An RMSD Flexible Alignment Algorithm

It is practically meaningful to design an algorithm that superimposes protein backbone chains with given accuracy and at the same time aligns as many as possible C_α atoms. In Chapter 2 of this dissertation an accuracy-adjustable pairwise backbone alignment algorithm is discussed. Part of the chapter is based on the published paper [103]. The algorithm first searches for a set of local alignments. Each local alignment consists of a series of consecutive C_α atom pairs in the backbones of two proteins. It then organizes the local alignments into a graph and attempts to find a global alignment which is an optimal group of local alignments sharing a common rigid body transformation. Based on the existing research, it is relatively easy to match two proteins in the local region, but it is hard to find the global alignment that combines multiple non-overlapped aligned local areas. The main technical contribution of this preliminary algorithm is the derivation of a global transformation that unifies local alignments. The algorithm can output an alignment result according to a user-defined maximum $RMSD$ value which allows a certain amount of flexibility in the stringency of the global alignment. Preliminary experiments are performed on 30 pairs of protein chains. The performance of the algorithm is evaluated based on the global alignment length i.e. N_{mat}, the number of aligned C_α atom pairs, and the global alignment accuracy i.e. the $RMSD$ distance between the aligned pairs. Experiments show that in many cases this algorithm has a higher N_{mat} than two well known existing algorithms (DaliLite, the pairwise version of DALI [54], and CE [85]) with an equal or lesser $RMSD$.

1.4 Improving the Preliminary Algorithm

The above protein structure alignment algorithm organizes local alignments into a graph with each local alignment being a vertex. The connectivity between vertices is determined by the consistency relationship between local alignments. Two local alignments are said to be consistent if they share a common rigid body transformation. Grouping mutually consistent local alignments is equivalent to finding cliques in a graph, which is an NP-complete problem. We present an alternative formulation of the alignment problem where the consistency check can be solved efficiently. In the preliminary alignment algorithm, this problem is simplified as looking for "stars" rather than cliques in a graph. A star is a set of

vertices including a center and all the other vertices that are connected to the center vertex. Since any clique must be included in some star, for our particular problem this simplification will not miss useful vertices, and it reduces the computational complexity to $O(n^2)$, where n is the number of vertices in the graph. The "star" approach has one center for each of its stars. This single-center star method is not flawless and shows some instability for aligning large proteins. To increase the stability of the single-center star method, in Chapter 3 a double-center star method is introduced to group the local alignments. Another crucial new technical development of Chapter 3 is the learning strategy based on feedback.The redesigned algorithm [104] is named SLIPSA, which stands for Self Learning and Improving Protein Structure Alignment. SLIPSA is self learning in that it has a feedback loop which sends the current alignment result back to its input in order to learn a better result in the next stage. SLIPSA accepts any reasonable upper-bound $RMSD$ value as one of the inputs, and outputs an alignment result with a $RMSD$ never greater than that value.

1.5 Finding Similar Structures in the Protein Data Bank

Protein structural similarity can be used to infer evolutionary relationships between proteins or to classify protein structures into more generalized groups; therefore a good protein structure alignment algorithm is very helpful for protein biologists. However, a good alignment algorithm itself is insufficient for the effective discovering of structural relationships between tens of thousands of proteins. It is hard to imagine that one could manually examine the structural similarity between 100,000 $\times$ 100,000 pairs of proteins chains; the structural relationship between proteins has to be discovered automatically. The field of protein structure query aims to find similar structures in the protein data bank according to a given query structure. Because of the requirements of both a fast and stable filter and a fast and accurate structure alignment tool, this area has posed an even greater research challenge than protein structure alignment. In Chapter 4 an efficient protein structure query algorithm and tool [71] is developed to find similar protein structures in the protein data bank for any given structure. With the combination of a series of fast and stable filters and a structure alignment algorithm particularly optimized for the query purpose, some exciting results have been observed when compared with SSM, the one we believe is among the best structure query web sites. An algorithm comparison model is also proposed to compare our protein structure query tool with others.

1.6 Diameter of Protein Backbone and Sublinear Time Computations

Protein backbones can be considered sequences of 3-D points in the Euclidean space. In our early research of protein structure alignment, we have attempted to derive a rough alignment between protein backbone chains based on computation of their diameters and matching some crucial points including the two points which determine the diameter of a chain. The diameter of a sequence of 3-D points is the largest distance between two points in the sequence. Clearly, the accurate diameter can be computed in quadratic time. This time complexity can be reduced to sublinear when an approximate diameter is acceptable.

Chapter 5 presents a series of deterministic and randomized algorithms [42] to approximate the diameter of a sequence of points in a metric space in sublinear time. Furthermore, based on the approximate diameter problem, a class of theoretical results are presented to separate the power of sublinear time computations. It is not uncommon to see that study in a certain scientific research area sometimes leads to interesting discoveries in another area. In our study of protein structure alignment, we have found such a case which brings us into a computational theory related area.

The computation for the approximate diameter is performed under deterministic, zero-error randomized, and bounded-error randomized models. We obtain a class of separations about the sublinear time computations using the various versions of the approximate diameter problem based on the restriction about the format of input data. We derive tight sublinear time separations for each of the three models such that computation with $O(n^r)$ time is more powerful than that with $O(n^{r-\epsilon})$ time, where r and ϵ are arbitrary parameters in $(0, 1)$ and $(0, r)$ respectively. We show that the bounded-error randomized sublinear time algorithms in time $O(n^r)$ cannot be done by a zero-error randomized sublinear time algorithm in $o(n)$ time or queries, where r is an arbitrary parameter in $(0, 1)$. We also show that zero-error randomized sublinear time algorithms in time $O(n^r)$ cannot be simulated by a deterministic sublinear time algorithm in $o(n)$ time or queries, where r is an arbitrary parameter in $(0, 1)$. We identify the parameters to control the diameter length and the permutation of the input points, and show how the sublinear time model and time complexity for computing the approximate diameter depend on those parameters.

1.7 Genomes, SNPs and Haplotypes

The genome of an organism is a complete DNA sequence of one set of chromosomes. DNA sequences can be derived from the biological raw material through the DNA sequencing process. A DNA sequence or genetic sequence is a succession of letters representing the primary structure of a real or hypothetical DNA molecule or strand, with the capacity to carry information. The possible letters are A, C, G, and T, representing the four nucleotide subunits of a DNA strand - adenine, cytosine, guanine, thymine. Therefore, a genome can be considered a string over the alphabet of nucleotides $\{A, C, G, T\}$. Since the cost of DNA sequencing continues to drop, many genomes, including the human genome, have been sequenced by various genome projects. Genomes of different organisms vary greatly in their sizes. For example, as the first sequenced RNA-genome, the bacteriophage MS2 has only 3569 DNA base pairs, whereas the amoeba dubia has 670 billion base pairs, which makes it the largest genome known so far. The human genome has approximately 3 billion DNA base pairs. A genome can also be represented as a sequence of genes, where each gene, corresponding to a unit of inheritance, is a long stretch of DNA. Genes are usually labeled with whole numbers, and when the polarity of genes are taken into account, the labels can be signed. With this representation, a genome is a sequence of signed numbers.

A single nucleotide polymorphism (SNP) is a DNA sequence variation occurring when a single nucleotide in the genome differs between members of a species or between paired chromosomes in an individual. For example, two sequenced DNA fragments from different individuals, AAGCCTA to AAGCTTA, contain a difference in a single nucleotide. In this

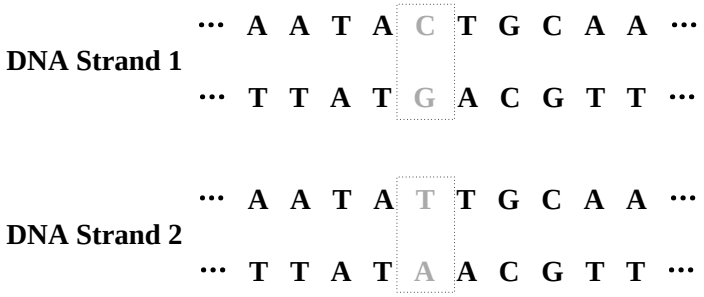


Figure 1.3: A C/T polymorphism between two DNA strands

case we say that there are two alleles: C and T. Almost all common SNPs have only two alleles. See Figure 1.3 for a C/T polymorphism between two double strand DNAs and Figure 1.4 for SNPs in the single strand DNA sequences of a set of individuals. The letters in gray indicates the positions of SNPs. The methods used to identify SNPs are the topic of SNP genotyping. Genotyping provides a measurement of the genetic variation between members of a species, and SNPs are the most common type of genetic variation. Variations in the DNA sequences of humans can affect how humans develop diseases and respond to pathogens, chemicals, drugs, vaccines, and other agents. SNPs can also provide a genetic fingerprint for use in identity testing. Additionally, the study of SNPs is important in crop and livestock breeding programs.

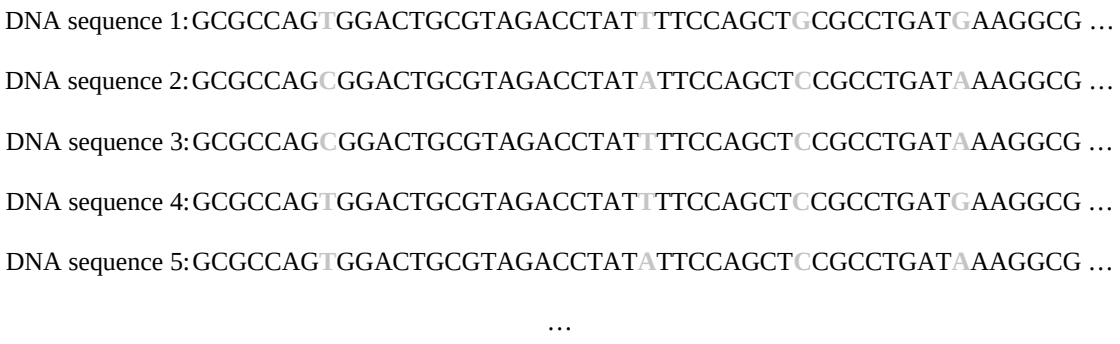


Figure 1.4: SNPs in the single strand DNA sequences of individuals

It is accepted that the genomes between any two humans are over 99% identical [53, 90]. The remaining sites which exhibit substantial variation (in at least 5% of the population) are SNPs. The values of a set of SNPs on a particular chromosome copy define a *haplotype*. See Figure 1.5 for an illustration of two haplotypes of an individual. The gray letters in the chromosomes indicate the positions of SNPs and form the two haplotypes. The gray letters in the haplotypes indicate the variations between them. While a haplotype is a string over the four nucleotide bases, typical SNP sites only vary between two values. Therefore, we can logically represent a haplotype as a binary string over the alphabet $\{A, B\}$. For

example, the two haplotypes in Figure 1.5 can be represented as (AAAAA ...) and (BAAAB ...), respectively. The assignment of As and Bs can be arbitrary as long as it reflects the variations between haplotypes. The haplotype of an individual chromosome can be thought of as a "genetic fingerprint" specifying the bulk of the genetic variability among distinct members of the same species. Determining haplotypes is thus a key step in the analysis of genetic variation.

Chromosome 1: CAACACGAAGGAAAGACGGGACCCAGGCCGACGTCCTATTAAAAGATAAT ...

Chromosome 2: CAACACCAAGGAAAGACGGGACCCAGGCCGACGTCCTATTAAAAGACAAT ...

Haplotype 1: GACTT ...

Haplotype 2: CACTC ...

Figure 1.5: Two haplotypes of an individual

In this dissertation, two problems that we have attempted to solve are the reconstruction of haplotypes from SNP fragments [23] and the comparison of genomes with gene repetitions based on the non-breaking similarity [24].

1.8 Reconstructing Haplotypes from SNP Fragments

In recent years, the haplotyping problem has been extensively studied, see, e.g., [9, 13, 28, 29, 43, 49, 50, 61, 62, 62, 67, 69, 81, 93, 94, 96, 100, 102]. There are several versions of the haplotyping problem. Of the most interest is the haplotyping of diploid organisms, such as humans, which have two chromosomes, and thus two haplotypes. A common method to obtain an individual's two haplotypes is to first obtain a collection of sequencing data from the individual's chromosomes. This data consists of a collection of fragments, each partially specifying the sequence of one of the two base chromosomes. This data typically contains two types of errors. The data is *incomplete* in that each fragment will fail to assign a base to some collection of positions, and *inconsistent* in that some positions may be sequenced with an incorrect value. Further, in practice it is not known which chromosome is being sequenced by which fragments. The general haplotype reconstruction problem is then to take this input set of sequenced fragments and derive the two haplotypes which yielded the given fragments. This problem, which we consider, is the singular haplotype reconstruction problem and, like other versions of the problem, has also been extensively studied, see, e.g., [13, 28, 69, 93]. Figure 1.6 shows two haplotypes and a matrix of SNP fragments with incomplete errors marked as hyphens and inconsistent errors as letters in gray. Each row in the matrix is a copy of a haplotype with reading errors, and its original haplotype number is displayed at the end of each row. Note that the two original haplotypes are unknown and our goal is to reconstruct them from the SNP matrix.

While much work has been done on these problems, the incompleteness and inconsistency of data fragments makes most versions of these haplotyping problems NP-hard or

Haplotype 1: ABABABABBABBBABABBBABABBBABBBABBAAABBBAAAAABAABABA

Haplotype 2: BBBBABABAABBBBAABBBAABBABABAABBABBAAABBBAAABABAABABA

Dissimilarity between haplotypes: 20%

SNP matrix: incomplete error rate 10%; inconsistent error rate: 4%

```
-BBBABABAABBBBBABBBAABBABAABBA-AAAABBBAAABABAABABA   2
ABA---BBBABAAABABBBABABBBA-BBABB-AAB-B-AA-A-AABABA   1
BBBBABABAABBBB-ABBAA-BBABABABAB-AAA-BBAAA-ABAB-ABA   2
-B-BABABBABBBABABB-ABABBB-BB-ABBAAABBBAAAAABAABABA   1
ABABA--BBABBBABABBB-BABBBABABAB-A-ABBBAAAAABAABABA   1
ABA-ABABBAB-BABABBBAB-B-BBBBBABBAAABBBAA-AABBABABA   1
BBBBABABAABBBBAABBBBABBBBAABBABBAAABBBAAABAAAAAABA   2
A-AB-BABBABBBA-ABBBBBABBB-BBBABBAAABBBAAAAA-AA-A-A   1
BBBBABABAABBBBAABBBA-BBABAABBAB-AA-ABBAAABA-AABAAA   2
BBBBA-ABAABBBBA-BBB-A-BABAABB-BB-AABB-AAABAAAABABA   2
```

Figure 1.6: A SNP matrix with incomplete and inconsistent errors

even hard to approximate (e.g., [13, 28, 69]). Many elegant and powerful methods such as those in [68] cannot be used to deal with incompleteness and inconsistency at the same time. Other methods have proposed heuristics [9, 43], but do not provide provably good results. In Chapter 6 a probabilistic approach is developed to overcome some of the difficulties caused by the incompleteness and inconsistency occurred in the input fragments. The experimental results yield provably good algorithms for our model. Further, we have tested both the accuracy and the speed of the algorithm empirically. The experiments yield results that conform to the theoretical performance efficiency. When compared to other algorithms with similar problem size, incompleteness, and inconsistency conditions, this algorithm obtains high accuracy and speed. The web demonstration program of haplotype reconstruction is available at `http://fpsa.cs.uno.edu/haprec/haprec.html`.

1.9 Non-breaking Similarity of Genomes with Gene Repetitions

In bio-molecular sequences (DNA, RNA, or amino acid sequences), high sequential similarity usually implies significant functional or structural similarity. In the genome com-

parison and rearrangement area, a standard problem is to compute the number of genetic operations to convert a source genome to a target genome. This problem is important in evolutionary molecular biology. To compare the difference between genomes, the breakpoint distance is used as one of the most famous distance measures [95]. The implicit idea of breakpoints can be traced back to as early as in 1936 by Sturtevant and Dobzhansky [87]. For two genomes X and Y without repetitions and based on the same alphabet, the breakpoint distance $b(X, Y)$ between them is defined as the number of pairs of genes that are adjacent in one genome but not in another. For signed genomes, the notion of adjacency requires that the gene polarity be conserved, so that if genome X contains two genes ordered as ab, then these two genes are adjacent in Y only if they occur as ab or as $-b-a$. For any X and Y, $b(X, Y) = b(Y, X)$. For example, the breakpoint distance between sequences $X = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)$ and $Y = (1, -3, -2, 4, 8, 5, -7, -6, 9, 10)$ is 6, since the pairs of genes that are adjacent in X but not in Y are (1,2), (3,4), (4,5), (5,6), (7,8) and (8, 9), or the pairs that are adjacent in Y but not in X are (1,-3), (-2,-4), (4,8), (8,5), (5,-7) and (-6,9).

In Chapter 7, a new similarity measure called non-breaking similarity is defined. Intuitively, this is the complement of the traditional breakpoint distance measure. Compared with the problem of computing exemplar breakpoint distance, which is a minimization problem, for the exemplar non-breaking similarity (ENbS) problem we need to maximize the number of non-breaking points. Unfortunately, we show that the Independent Set problem in graph theory can be reduced to the problem of computing the ENbS, and this reduction implies that ENbS is W[1]-complete (and ENbS does not have a factor-n polynomial-time approximation). This reduction works even when one of the two genomes is exemplar. However, we also show that for several practically interesting cases, there are polynomial time algorithms. This is done by parameterizing some quantities in the input genomes, followed by some traditional algorithmic techniques.

Summary This dissertation attempts to address some bioinformatics problems in the areas of both protein structure analysis and genome sequence analysis. The primary effort is put forth on protein structure related problems such as Chapter 2 [103] and Chapter 3 [104]: developing efficient algorithms for pairwise protein structure alignment, and Chapter 4 [71]: searching for similar structures in the Protein Data Bank. Besides these basic problems, in Chapter 5 [42] an interesting diameter calculation problem emerging with the development of the protein structure alignment algorithm is discussed. A set of sublinear time approximate algorithms for the diameter of a 3-D sequence of points and a class of separations about the power of sublinear time computations are presented. In addition, two genome sequence related problems are studied, including Chapter 6 [23]: algorithms for reconstructing haplotypes from SNP fragments and Chapter 7 [24]: theoretical results and algorithms for comparing genomes on the basis of non-breaking similarities. Finally, Chapter 8 concludes the contributions of this dissertation and points out possible future work for the discussed problems.

Chapter 2

A Preliminary Protein Structure Alignment Algorithm

There are different definitions for the protein structure alignment problem. In this chapter[1] we formally define a pairwise protein structure alignment problem that is studied in this dissertation, and then propose a new algorithm for it.

Definition 2.1. Given two protein backbone 3-D point sequences S and S', the pairwise protein structure alignment problem is to find $S_A \subseteq S$ and $S'_A \subseteq S'$ and a rigid body transformation F to maximize $N_{mat} = |S_A| = |S'_A|$ with $RMSD \leq RMSD_{max}$, where $S = p_1 \cdots p_m$, $S' = q_1 \cdots q_n$, $S_A = p_{i_1} \cdots p_{i_{N_{mat}}}$, $S'_A = q_{j_1} \cdots q_{j_{N_{mat}}}$, and $RMSD_{max}$ is a user-defined upper-bond of $RMSD$. See Equation 1.1 for the definition of $RMSD$.

There are two phases in this algorithm. The first phase aligns all the possible local regions between two protein backbones. Each local region of a protein consists of a series of consecutive C_α atoms in a backbone chain of the protein. The second phase derives a global rigid body transformation that combines some aligned local regions to achieve a global alignment and to make it as large as possible. The main technical contribution of this work is a new algorithm for the second phase, finding an optimal transformation for the global alignment. Also, the alignment accuracy measured by $RMSD$ is adjustable by the user.

2.1 Concepts for 3-D Sequences

To be distinguished from the generally used term *protein sequence*, here a *3-D sequence* (or briefly, *sequence*) is used to describe an ordered chain of points in the 3-D Euclidean space R^3. For two points p, q in R^3, $\text{dist}(p, q)$ is the Euclidean distance between them.

Definition 2.2. Assume $\epsilon \geq 0$. For two sequences $S = p_1 \cdots p_n$ and $S' = q_1 \cdots q_n$ of points in R^3, they are *ϵ-match* if there exists a rigid body transformation F such that $\text{dist}(p_i, F(q_i)) \leq \epsilon$ for $i = 1, \cdots, n$.

For a sequence $S = p_1 \cdots p_n$, $S_{i,l} = p_i \cdots p_{i+l-1}$ is a *subsequence* of S, where $1 \leq i \leq n$ and i is the starting point of the subsequence; $1 \leq l \leq n - i + 1$ and l is the length of the subsequence.

[1] Published work [103] used with permission of the CSREA Press.

For two sequences $S = p_1 \cdots p_n$ and $S' = q_1 \cdots q_m$, a triple $L = (i, j, l)$, where i is the starting point of $S_{i,l}$, j is the starting point of $S'_{j,l}$, and l is the length of $S_{i,l}$ and $S'_{j,l}$, is called a *strict local ϵ-match* between S and S', if there exists a rigid body transformation F_L *via* which $S_{i,l}$ and $S'_{j,l}$ are ϵ-match. A strict local ϵ-match is also called a *local ϵ-match* or briefly an *ϵ-match*.

Lemma 2.3. *If $L = (i, j, l)$ is a local ϵ-match between S and S', then for any $0 \leq u, v \leq l-1$, $|\text{dist}(p_{i+u}, p_{i+v}) - \text{dist}(q_{j+u}, q_{j+v})| \leq 2\epsilon$.*

Proof: Assume that F is a transformation *via* which L is an ϵ-match. In R^3 we have $\text{dist}(p_{i+u}, p_{i+v}) \leq \text{dist}(p_{i+u}, F(q_{j+u})) + \text{dist}(F(q_{j+u}), F(q_{j+v})) + \text{dist}(F(q_{j+v}), p_{i+v})$, because the straight line distance between p_{i+u} and p_{i+v} is the shortest one among all the routes from p_{i+u} to p_{i+v}. Since L is an ϵ-match, $\text{dist}(p_{i+u}, F(q_{j+u})), \text{dist}(F(q_{j+v}), p_{i+v}) \leq \epsilon$ according to Definition 2.2. Therefore $\text{dist}(p_{i+u}, p_{i+v}) \leq \epsilon + \text{dist}(F(q_{j+u}), F(q_{j+v})) + \epsilon = 2\epsilon + dist(q_{j+u}, q_{j+v})$ because F is a rigid body transformation which makes $\text{dist}(F(q_{j+u}), F(q_{j+v})) = \text{dist}(q_{j+u}, q_{j+v})$. Therefore $\text{dist}(p_{i+u}, p_{i+v}) - \text{dist}(q_{j+u}, q_{j+v}) \leq 2\epsilon$. Similarly, $\text{dist}(q_{j+u}, q_{j+v}) - \text{dist}(p_{i+u}, p_{i+v}) \leq 2\epsilon$. Hence $|\text{dist}(p_{i+u}, p_{i+v}) - \text{dist}(q_{j+u}, q_{j+v})| \leq 2\epsilon$ when $L = (i, j, l)$ is a local ϵ-match between S and S'. ∎

Corollary 2.4. *For two subsequences $S_{i,l}$ and $S'_{j,l}$, if $|\text{dist}(p_i, p_{i+l-1}) - \text{dist}(q_j, q_{j+l-1})| > 2\epsilon$, then $L = (i, j, l)$ is not a local ϵ-match between S and S'.*

Proof: Assuming L is a local ϵ-match. Let $u = 0$ and $v = l - 1$, then according to Lemma 2.3, there must be $|\text{dist}(p_i, p_{i+l-1}) - \text{dist}(q_j, q_{j+l-1})| \leq 2\epsilon$. A contradiction. ∎

Definition 2.5. For two subsequences $S_{i,l}$ and $S'_{j,l}$, if for any $0 \leq u, v \leq l - 1$ and $u \neq v$, $|\text{dist}(p_{i+u}, p_{i+v}) - \text{dist}(q_{j+u}, q_{j+v})| \leq 2\epsilon$, then $L' = (i, j, l)$ is called a *relaxed local ϵ-match* between S and S'. For two sequences $S = p_1 \cdots p_n$ and $S' = q_1 \cdots q_m$, if $L_1, L_2, \cdots, L_k$ are local ϵ-matches between S and S' and each one of them starts from point i in S and point j in S', then the one with the maximum local match length is called a *longest local ϵ-match* between S and S'. Similarly, we define a *longest relaxed local ϵ-match* between S and S'.

Lemma 2.6. *If $L_{\max} = (i, j, l_{\max})$ is the longest local ϵ-match between S and S' which starts from point i in S and point j in S', and $L'_{\max} = (i, j, l'_{\max})$ is the longest relaxed local ϵ-match between S and S' which starts also from point i in S and point j in S', then $l'_{\max} \geq l_{\max}$ i.e. $L_{\max}$ is included by $L'_{\max}$.*

Proof: If $L' = (i, j, l')$ is a relaxed local ϵ-match between S and S', by the definition of $L'_{\max}$, there must be $l'_{\max} \geq l'$. According to Definition 2.5 and Lemma 2.3, a local ϵ-match between S and S' must also be a relaxed local ϵ-match between S and S', so $L_{\max} = (i, j, l_{\max})$ is a relaxed local ϵ-match between S and S'. Hence we have $l'_{\max} \geq l_{\max}$. ∎

Definition 2.7. For two sequences $S = p_1 \cdots p_n$ and $S' = q_1 \cdots q_m$, if $L_1, L_2, \cdots, L_k$ are non-overlapped local ϵ-matches between S and S', and there exists a common rigid body transformation F_G *via* which any $L_g (1 \leq g \leq k)$ is ϵ-match, then $A_G = \{L_1, L_2, \cdots, L_k\}$ is called a *global ϵ-match* between S and S'.

2.2 Local Alignment

Given a distance constant $\epsilon > 0$ and two protein backbone chains represented by two sequences $S = p_1 \cdots p_n$ and $S' = q_1 \cdots q_m$ of points in 3-D Euclidean space, a local alignment algorithm should find out the longest local ϵ-matches for each pair of starting points in S and S'. A straightforward idea is to list all the pairs of subsequences of equal length in S and S', then examine each pair of them and try to find out all the longest local ϵ-matches. However, this straightforward method can be very time consuming, because it needs to find out an optimal transformation for each pair of subsequences, to transform one to align it with another one, and to calculate the distance between the aligned subsequences before it can tell whether these two subsequences are ϵ-match or not.

Practically, we may not need to find out the longest local ϵ-matches in a strict manner. In the local alignment phase we can search for all the longest relaxed local ϵ-matches which by Lemma 2.6 include all the longest strict local ϵ-matches between S and S'. The merit of such a relaxation is that, in the local alignment phase, we do not need to find out any transformation, nor to compare the distance from one subsequence to another transformed one. Figure 2.1 illustrates a local alignment $L = (i, j, l)$ which is a longest relaxed local ϵ-match between chains S and S' with starting points p_i and q_j and length l.

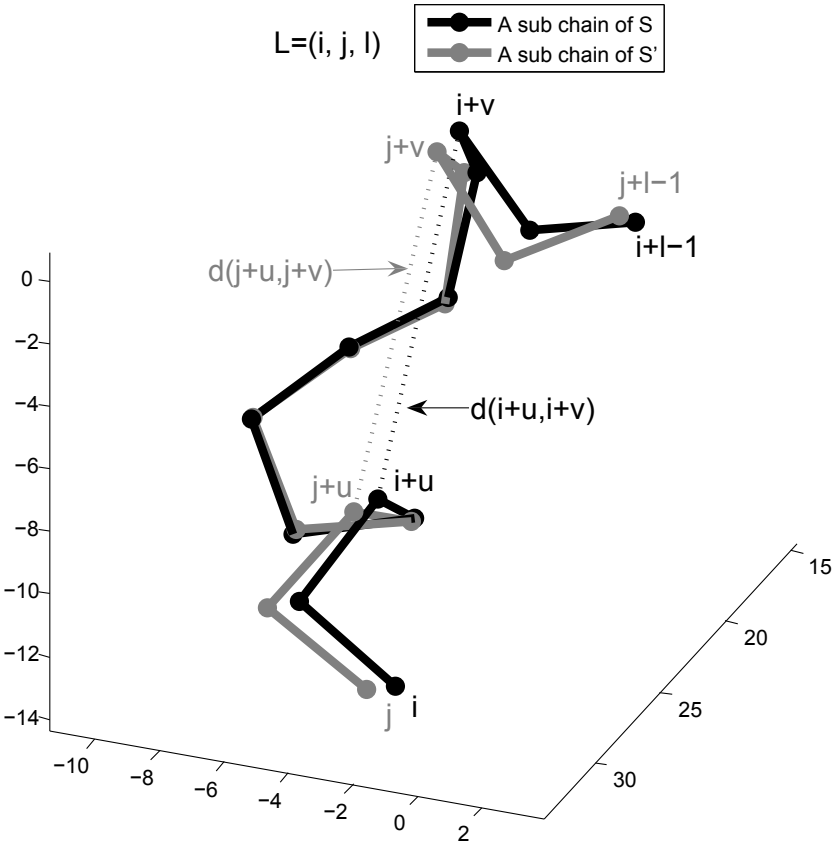


Figure 2.1: Local alignment L=(i, j, l)

There is an additional method to speed up the local alignment phase. Given any point p_i in S and any point q_j in S', we want to find out the longest relaxed local ϵ-match starting from p_i and q_j. We know that the length of a local alignment should be large enough to make sense, so if we assume that any meaningful local match $L = (i, j, l)$ should have its length $l \geq l_{\min}$, where $l_{\min}$ is a predefined minimum length of local match, then according to Lemma 2.3, we first check if $|\text{dist}(p_i, p_{i+l_{\min}-1}) - \text{dist}(q_j, q_{j+l_{\min}-1})| \leq 2\epsilon$. Only when the condition is true, the finding of the longest relaxed local match for positions i and j will start, otherwise just skip it. In our experiments this filter prevents the unnecessary calculation of those short local alignments that account for averagely 60% - 75% of all the possible local alignments.

Get-Local-Alignments$(S, S', \epsilon, l_{\min})$
Input: protein backbone chains $S = p_1 \cdots p_n$, $S' = q_1 \cdots q_m$, distance constant ϵ and minimum local alignment length $l_{\min}$, where each p or q is a 3-D point corresponding to a C_α atom in a protein backbone.
Output: $A_L = \{L_1, L_2, \cdots, L_w\}$, a set containing all the local alignments of length $\geq l_{\min}$ between S and S'. Each local alignment satisfies the relaxed ϵ-match constraint.
begin
    $A_L \leftarrow \{\}$;
    for each point $p_i (1 \leq i \leq n - l_{\min} + 1)$ in S and point $q_j (1 \leq j \leq m - l_{\min} + 1)$ in S'
        if $|\text{dist}(p_i, p_{i+l_{\min}-1}) - \text{dist}(q_j, q_{j+l_{\min}-1})| \leq 2\epsilon$
            find out the longest relaxed local ϵ-match starting from p_i and q_j;
            $l_{\max} \leftarrow$ the length of the above match;
            if $l_{\max} \geq l_{\min}$
                $A_L \leftarrow A_L \cup \{(i, j, l_{max})\}$;
            end if
        end if
    end for
    return A_L;
end

Complexity of Get-Local-Alignments: Assuming the number of C_α atoms in the two backbones are m and n. We also assume that the number of local alignments with $|\text{dist}(p_i, p_{i+l_{\min}-1}) - \text{dist}(q_j, q_{j+l_{\min}-1})| \leq 2\epsilon$ is h and the length of the u-th local alignment is l_u for $(u = 1, 2, \cdots, h)$. The computational complexity of the local alignment phase is $O(\sum_{u=1}^{h} l_u^2 + mn)$. Suppose $m = O(n)$, the worst case complexity is $O(n^4)$ since h can be $O(n^2)$ and each l_u can be $O(n)$.

2.3 Global Alignment

After local alignments are obtained, they are organized into groups. Ideally, only mutually consistent local alignments should be added to the same group. Suppose there are two local alignments $L_1 = (i_1, j_1, l_1)$ and $L_2 = (i_2, j_2, l_2)$, the point set $P = \{p_{i_1}, \cdots, p_{i_1+l_1-1}, p_{i_2}, \cdots, p_{i_2+l_2-1}\}$ is all the aligned points in the first chain, including those in L_1 and L_2, and $Q = \{q_{j_1}, \cdots, q_{j_1+l_1-1}, q_{j_2}, \cdots, q_{j_2+l_2-1}\}$ is all the aligned points in the second chain, also including those in L_1 and L_2. We say that local alignments L_1 and

L_2 are consistent if, after applying a rigid body transformation to Q, the $RMSD$ between P and transformed Q is small enough. In other words, if we have a set of local alignments, we conclude that all these local alignments are consistent if all the local alignments share a common rigid body transformation which makes them consistent with each other. Therefore a global alignment can be defined as such a set of consistent local alignments with a common transformation and an acceptable $RMSD$.

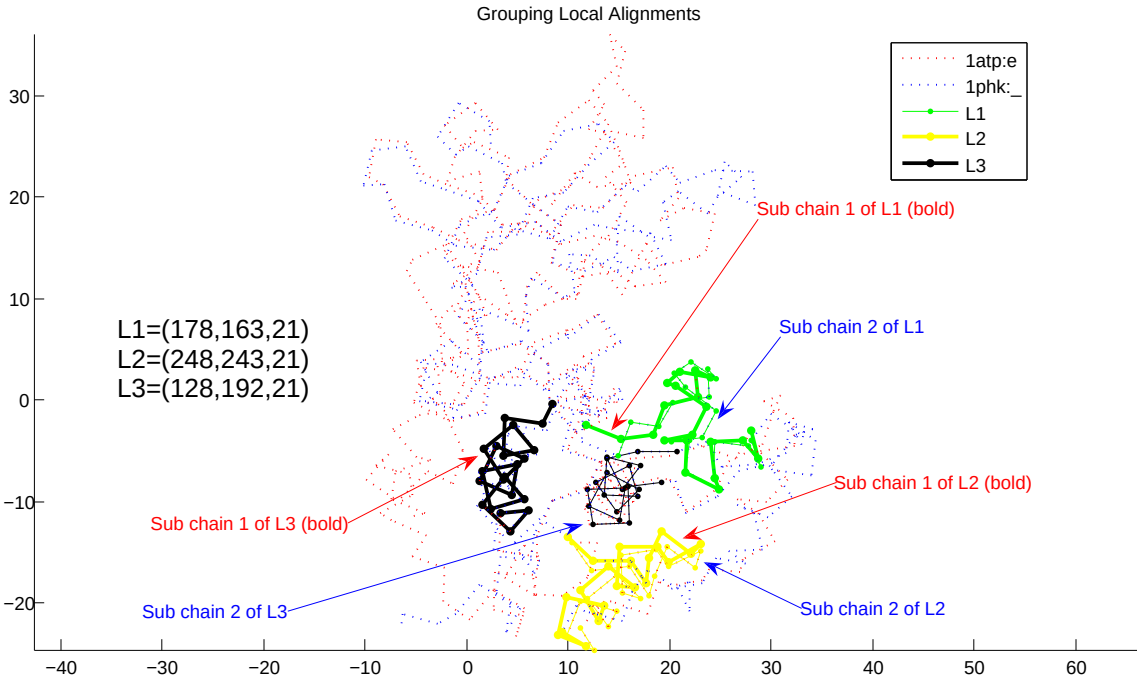


Figure 2.2: Construction of a graph

The consistency relationship between local alignments can be represented as a graph. Given $A_L = \{L_1, L_2, \cdots, L_w\}$ where each $L_u = (i_u, j_u, l_u)$ $(1 \leq u \leq w)$ is a local alignment. A graph $G = (V, E)$ is defined accordingly, where each local alignment is a vertex of the graph, $V = A_L$ is the vertex set and E is the edge set. Edge $e_{uv}, e_{vu} \in E$ if and only if L_u and L_v are consistent. Figure 2.2 illustrates the construction of such a graph, where local alignments L_1 and L_2 are consistent and L_3 is consistent with neither L_1 nor L_2, therefore edges $e_{12}, e_{21} \in E$ and $e_{13}, e_{31}, e_{23}, e_{32} \notin E$. With this representation, grouping mutually consistent local alignments is equivalent to finding cliques in a graph. Straightforwardly, a global alignment algorithm should find from graph G a clique with the largest global alignment length among all the cliques. However, finding cliques in a graph is NP-complete. To simplify the process, we find "stars" instead of cliques in graph G. A star is a set of vertices including a center and all the other vertices that are connected to the center vertex. Since any clique must be included in some star, for our particular problem this simplification will not miss useful vertices. Figure 2.3 shows a graph, two cliques and an example star. There are 6 stars in the graph since $|V|$=6. They are $Star_1 = Star_2 = Star_6 = \{L_1, L_2, L_5, L_6\}$, $Star_3 = Star_4 = \{L_3, L_4, L_5\}$ and $Star_5 = \{L_1, L_2, L_3, L_4, L_5, L_6\}$. A set of all the unique

stars is $Stars = \{Star_1, Star_3, Star_5\}$. Note that each star is finally a set of local alignments and each local alignment is a set of C_α pairs.

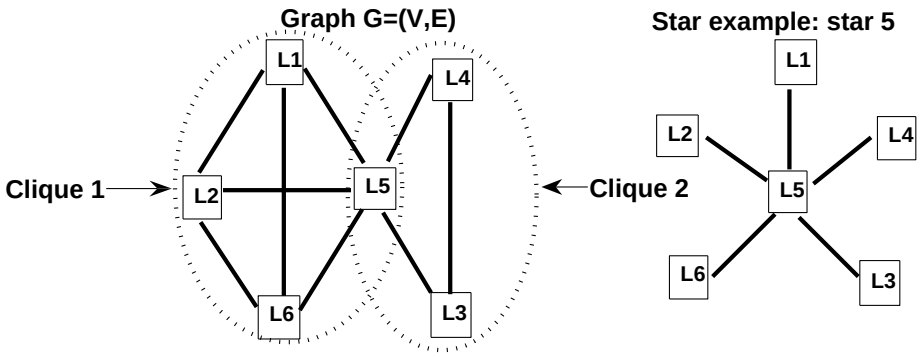


Figure 2.3: An example star

For each unique star, a corresponding global alignment candidate is calculated by deleting badly aligned C_α pairs involved in that star. Then all the candidates are compared and the optimal one is chosen. An example global alignment between protein chains 1ATP:E and 1PHK is shown in Figure 1.2, where N_{mat} is the number of aligned C_α pairs, $RMSD$ is the root mean square distance between the aligned pairs, and the rigid body transformation used to align the two chains is T (the translation vector) and R (the rotation matrix). There are many algorithms for finding a rigid body transformation between two sets of 3-D points [34]. In this dissertation a least square estimation method [92] is applied for our protein alignment experiments.

Group-Local-Alignments(A_L, $RMSD_{\max}$)
Input: $A_L = \{L_1, L_2, \cdots, L_w\}$ and $RMSD_{\max}$.
Output: $Stars$ (a set of all the different stars found in graph G).
begin
    $E \leftarrow \{\}$;
    $Stars \leftarrow \{\}$;
    for (every two local alignments L_u and L_v in A_L)
        if (there exists a common transformation F for L_u and L_v with
          $RMSD \leq RMSD_{\max}$)
            $E \leftarrow E \cup \{e_{uv}, e_{vu}\}$;
        end if
    end for
    for ($u \leftarrow 1$ to w)
        $Star_u \leftarrow \{\}$;
        for ($v \leftarrow 1$ to w)
            if ($e_{uv} \in E$)
                $Star_u \leftarrow Star_u \cup \{L_v\}$;
            end if
        end for

if ($Star_u \notin Stars$)
$Stars \leftarrow Stars \cup \{Star_u\}$;
end if
end for
return $Stars$;
end

In order to obtain an optimal rigid body transformation, it has been proven that the geometric centers of two point sets must be superimposed. This can be done by moving both point sets to the origin of a common coordinate system [35]. Then a rotation is found to minimize the $RMSD$ between them. Specifically, the following procedure describes the method that we used to calculate an optimal transformation.

Get-Optimal-Transformation(S_1, S_2)
Input: S_1 and S_2 are two 3-D point sets of size n. Each is represented by a $n \times 3$ matrix.
Output: A rigid body transformation F represented by a translation vector T and a rotation matrix R.
begin
$c_1 \leftarrow$ the centroid of S_1; $c_2 \leftarrow$ the centroid of S_2;
Move S_1 so that point $(0, 0, 0)$ is the centroid of the new S_1;
Move S_2 so that point $(0, 0, 0)$ is the centroid of the new S_2;
Apply a singular value decomposition to matrix $S_1' S_2$. Suppose that matrices U, D and V satisfy $UDV = S_1' S_2$.
$R \leftarrow UV'$; $T \leftarrow c_1 - c_2 R'$;
return R and T;
end

Complexity of Group-Local-Alignments: We assume that the number of local alignments is w and the length of the u-th local alignment is $l_u (\geq l_{\min})$ for $(u = 1, 2, \cdots, w)$. Assume that $C(t)$ is the computational complexity for finding the rigid body transformation between t pairs of 3-D points by applying the least-square estimation method [92]. The complexity for the procedure Group-Local-Alignments is $O(\sum_{u=1}^{w} \sum_{v=1}^{w} C(l_u + l_v))$. Suppose $m = O(n)$, in the worst case the complexity is $O(n^4 C(n))$ because w can be $O(n^2)$ and each l_u or l_v can be $O(n)$.

After grouping consistent local alignments, we develop our global alignment algorithm which compares all the groups of local alignments and outputs the one with the largest global alignment length. To speed up the searching of a global alignment with a branch and bound method, we first sort the stars by a descending order of the number of point pairs involved in each star.

Get-Global-Alignment($Stars$, ϵ, $RMSD_{\max}$)
Input: $Stars$, ϵ, and $RMSD_{\max}$.
Output: $A_G = \{L_1, L_2, \cdots, L_k\}$ (the largest set of consistent local alignments which share a common transformation F_G with the global $RMSD \leq RMSD_{\max}$).
begin
sort $Stars$ by a descending order of the number of 3-D point pairs involved in each star;
$l_{max} \leftarrow 0$
for (every $star_i$ in $Stars$)

$l_i \leftarrow$ the number of 3-D point pairs involved in $star_i$;
if $(l_i > l_{max})$
calculate a transformation F_G for all the 3-D point pairs involved in $star_i$;
while ($RMSD > RMSD_{\max}$ and point pair (p, q) has the maximum dist$(p, F_G(q))$)
remove point pair (p, q) from the point sets;
adjust the local alignment from which point pair (p, q) is removed;
$l_i \leftarrow l_i - 1$;
recalculate F_G and $RMSD$;
end while
if $(l_i > l_{max})$
$A_G \leftarrow$ the current set of local alignments in $star_i$;
$l_{max} \leftarrow l_i$;
else
return A_G;
end if
end if
end for
end

When point pair (p, q) is removed due to its large dist$(p, F_G(q))$, the corresponding local alignment that contains (p, q) will be broken into two parts if (p, q) is in the middle of the local alignment. In this case the local alignment should be split into two smaller ones. If the alignment is not broken, then (p, q) is either the head or the tail pair in the local alignment. If (p, q) is the head pair, then the local alignment should have its head pair moved to the next position, and the length of the local alignment is reduced by one. If (p, q) is the tail pair, then the only thing that we need to do is to reduce the alignment length by one. In any of these cases the local alignment set should be updated accordingly.

Complexity of Get-Global-Alignment: Assuming the number of C_α atoms in the two backbones are m and n. Assuming $C(t)$ is the computational complexity for finding the rigid body transformation between t pairs of 3-D points by applying the least-square estimation method [92]. As before, we assume that the number of local alignments is w and the length of the u-th local alignment L_u is $l_u (\geq l_{\min})$ for $(u = 1, 2, \cdots, w)$. The graph has w nodes and each node is the center of one star. Assume that the star with node u as center has s_u local alignments $L_{i_1}, \cdots, L_{i_{s_u}}$. Let t_u be the number of pairs of points in $L_{i_1}, \cdots, L_{i_{s_u}}$. Thus, $t_u = l_{i_1} + \cdots + l_{i_{s_u}}$. For the star with node u as its center, it removes at most t_u pairs of points and applies the least-square estimation (for F_G) at most t_u times. The complexity of Get-Global-Alignment is $O(\sum_{u=1}^{w} \sum_{t=1}^{t_u} C(t))$. Suppose $m = O(n)$, the worst case complexity is $O(n^2 C^2(n))$ because w can be $O(n^2)$ and each t_u can be $O(n)$.

2.4 Experimental Results

The algorithm was tested on two test sets. The first set consists of 20 protein backbone chains versus chain E of the quaternary complex of cAMP dependent kinase (1ATP:E) with 336 C_α atoms. See [85] for the description of this test set. The second test set contains 10

Table 2.1: Test results of 20 protein pairs

Chain 1: 1ATP:E					
No.	Chain 2	$N_{mat}/RMSD$ (Our Method)	$N_{mat}/RMSD$ (DaliLite)	$N_{mat}/RMSD$ (Our Method)	$N_{mat}/RMSD$ (CE)
1	1APM:E	336/0.3	336/0.3	336/0.3	336/0.3
2	1CDK:A	336/0.4	336/0.4	336/0.4	336/0.4
3	1YDR:E	336/0.5	334/0.5	336/0.5	336/0.5
4	1CTP:E	323/1.7	323/1.7	318/1.5	302/1.5
5	1PHK:_	256/1.6	255/1.6	254/1.4	254/1.4
6	1KOA:_	264/2.8	261/2.8	262/2.7	257/2.7
7	1KOB:A	265/2.8	263/2.8	263/2.7	259/2.7
8	1AD5:A	242/2.6	243/2.6	241/2.5	236/2.5
9	1CKI:A	255/2.5	253/2.5	259/2.7	259/2.7
10	1CSN:_	254/2.5	253/2.5	252/2.4	248/2.4
11	1ERK:_	265/2.9	265/2.9	260/2.6	253/2.6
12	1FIN:A	259/2.4	253/2.4	254/2.2	252/2.2
13	1GOL:_	268/3.0	266/3.0	261/2.6	253/2.6
14	1JST:A	262/2.9	261/2.9	254/2.4	252/2.4
15	1IRK:_	253/3.6	254/3.6	254/3.7	257/3.7
16	1FGK:A	244/3.1	246/3.1	250/3.4	253/3.4
17	1FMK:_	243/2.8	246/2.8	243/2.8	256/2.8
18	1WFC:_	255/3.4	250/3.4	246/3.0	244/3.0
19	1KNY:A	125/4.9	99/4.9	115/4.2	111/4.2
20	1TIG:_	64/3.4	56/3.4	67/3.9	54/4.0

pairs of protein backbone chains that were cited in [38] as difficult structures for alignment. The two sets were tested using three different algorithms: DaliLite, CE, and our algorithm. The first two algorithms have publicly available websites [5, 6]. Tables 2.1 and 2.2 compare our results with DaliLite and CE for the first test set and the second test set, respectively. In each test case we set the maximum $RMSD$ of our algorithm to be less than or equal to the compared algorithm. In the tables, N_{mat} is the number of matched atom pairs in two backbone chains, and $RMSD$ is the square root of the average square of the distance between the matched atoms (after the rigid body transformation for the global alignment). In Table 2.1 our algorithm has 12 cases with higher N_{mat} and 4 cases with lower N_{mat} than DaliLite. It has 4 cases of the same N_{mat} as DaliLite. 2 of these 4 cases (No. 1 and 2) have reached the maximum N_{mat} (i.e. all the 336 atoms are matched). So, it is impossible for any algorithm to achieve a better value of N_{mat}. When compared with CE, our algorithm has 12 cases with higher N_{mat} and 3 cases with lower N_{mat}. It has 5 cases of the same N_{mat}, in which 3 cases (No. 1, 2 and 3) have reached the maximum N_{mat}. In Table 2.2 our algorithm has 6 cases with higher N_{mat} and 2 cases with lower N_{mat} than DaliLite. It has 1 case of the same N_{mat} as DaliLite. In test case 24, we were not able to compare our algorithm with DaliLite because a test protein was missing on the DaliLite website. In Table 2.2 our

algorithm has 5 cases with higher N_{mat} and 4 cases with lower N_{mat} than CE. It also has 1 case of the same N_{mat} as CE.

Table 2.2: Test results of 10 protein pairs

No.	Chain 1	Chain 2	$N_{mat}/RMSD$ (Our Method)	$N_{mat}/RMSD$ (DaliLite)	$N_{mat}/RMSD$ (Our Method)	$N_{mat}/RMSD$ (CE)
21	1FXI:A	1UBQ:_	58/2.6	60/2.6	67/3.6	64/3.8
22	1TEN:_	3HHR:B	86/1.8	86/1.9	86/1.8	87/1.9
23	3HLA:B	2RHE:_	80/3.0	75/3.0	83/3.4	84/3.4
24	2AZA:A	1PAZ:_	-	-	84/2.9	84/2.9
25	1CEW:I	1MOL:A	81/2.1	81/2.3	81/2.1	81/2.3
26	1CID:_	2RHE:_	100/3.2	97/3.2	98/2.9	97/2.9
27	1CRL:_	1EDE:_	205/3.5	211/3.5	210/3.8	219/3.8
28	2SIM:_	1NSB:A	292/3.3	292/3.3	286/3.0	275/3.0
29	1BGE:B	2GMF:A	102/3.3	94/3.3	109/3.9	107/3.9
30	1TIE:_	4FGF:_	115/3.1	114/3.1	114/2.9	116/2.9

The ϵ for each test case is adjustable. Moreover, the ϵ for the local alignment phase and the ϵ for the global alignment phase can be different. To obtain the results shown in the tables, the ϵ for local alignment is adjustable from 1.5 to 4.0, the ϵ for global alignment is not less than the given $RMSD_{max}$, and $l_{\min}$ is set to 10.

According to the test results in the tables, our algorithm shows higher N_{mat} in most of the cases when compared with DaliLite and CE. Furthermore, our algorithm is $RMSD$-flexible, therefore it can be used to obtain protein structure alignments with any reasonable $RMSD$. This makes it convenient for users to set up $RMSD$ values that make sense to them.

On the other hand, there are various reasons why our algorithm shows lower N_{mat} in a few test cases. Here are some examples. In case 15, DaliLite has a local alignment (51, 43, 11) i.e. 11 atom pairs (51 to 61 in chain 1 and 43 to 53 in chain 2) are aligned, while our algorithm has (51, 43, 12). Our algorithm obtains more locally aligned atom pairs, however this is not optimal when global alignment is considered. In case 16, DaliLite has a local alignment (51, 35, 16), while our algorithm has (51, 34, 17). It matches its star center better than (51, 35, 16), however the match is not optimal for global alignment. In case 17, DaliLite has a local alignment (283, 423, 7) , while our algorithm obtains (283, 423, 3) only, due to an ϵ that is good for global alignment but not for this particular local alignment.

There are similar reasons why our algorithm has lower N_{mat} than CE in some cases. Also, it should be noticed that, although our MATLAB program read out the same number of C_α atoms from the PDB files as DaliLite did, CE sometimes read out more. For instances, in the cases that we got lower N_{mat} than CE, it read out 306, 310, and 452 C_α atoms from chain 2 of cases 15, 16 and 17, respectively, while our program and DaliLite read out 303, 278 and 438, respectively; CE read out 90 atoms from chain 1 and 203 atoms from chain 2 of case 22, while our program and DaliLite read out 89 and 195; CE read out 172 atoms

from chain 1 and 146 atoms from chain 2 of case 30, while our program and DaliLite read out 166 and 124.

Finally, as an example, we give out two lists of the positions of the aligned C_α atoms for test case 20, 1ATP:E as chain 1 and 1TIG:_ as chain 2. For a pair of protein backbone chains, a global alignment is described as a set of non-overlapped local alignments that share a common rigid body transformation. Each local alignment is represented by a triple $L = (i, j, l)$ where i is a starting point in chain 1, j is a starting point in chain 2, and l is the number of aligned C_α atoms.

The following is the two lists of positions for entry 20 of Table 2.1. Our result has much higher N_{mat} than both DaliLite and CE. With these positions, the readers can even verify our alignment result.

$RMSD$=3.4: (1,13,20), (82,33,3), (86,36,1), (87, 38, 2), (126,45,1), (130,46,18), (148,66,3), (151, 70, 3), (158,73,4), (165,77,1), (166,79,3), (171,82,1), (174, 83, 2), (176,86,1), (335,88,1).=3.9: (1,13,20), (82,33,8), (126,44, 2), (131, 46, 17), (148,66,3), (151,70,3), (157,73,4), (165,77,5), (171,82,1), (174,83,2), (176,86,1), (335, 88, 1).

Chapter 3

A Self Learning Protein Structure Alignment Algorithm

This chapter[2] discusses an improved version of the preliminary algorithm presented in Chapter 2. The new algorithm is named SLIPSA which stands for Self-Learning and Improving Protein Structure Alignment. SLIPSA has proceeded far beyond its preliminary version in terms of maturity, stability, efficiency and availability. The SLIPSA algorithm first searches all the local alignments, after that consistent local alignments are grouped into global alignment candidates called "double-center stars" and a currently optimal global alignment is chosen from all the candidates. Then this output is sent back to its own input in order to learn from itself. We call this a feedback. Such feedback is repeated to obtain improved results, until finally an optimal alignment (i.e. a result with as many as possible aligned C_α pairs and an acceptable $RMSD$). SLIPSA can also learn from other algorithms when they are available.

3.1 Introduction of Double-center "Stars"

A "star" approach has been used in the earlier version of this algorithm [103], which has one center for each of its stars and shows some instability for aligning large proteins. We now introduce a double-center method to group the local alignments. This greatly improves the reliability of the previous algorithm. Another crucial new technical development of this chapter is a learning strategy based on feedback, which is described in Section 3.2. The combination of two new methods greatly improves speed, reliability, and accuracy of the old algorithm.

The single-center star method is not flawless. It works well when the two protein chains match well or the chain diameters are small. However, we have found that it is less stable when the chains do not match very well or the chain diameters are large. This is caused by deleting badly matched C_α pairs from each star, a method applied to obtain a global alignment candidate. When local alignments are grouped into an initial star, there may exist point pairs which do not match well. An initial transformation is calculated and the worst matched pair based on that transformation is first deleted, then the transformation is recalculated to select the second worst pair. This process is repeated. In this way the

[2]Published work [104] used with permission of World Scientific Publishing Co. and Mary Ann Liebert Inc.

well matched pairs survive and the $RMSD$ becomes smaller and smaller, until an acceptable $RMSD$ is achieved. The effect of deleting bad point pairs relies on a good initial transformation, which in turn depends on the star center selection. With a single star center, the initial transformation has great freedom to move and rotate in the point pair deletion process, thus the deletion may go along a more unpredictable way. This is more obvious when the local alignments are relatively short, which usually happens when the chains do not match very well or the chain diameters are large. Based on this observation, we consider grouping local alignments into *double-center stars*.

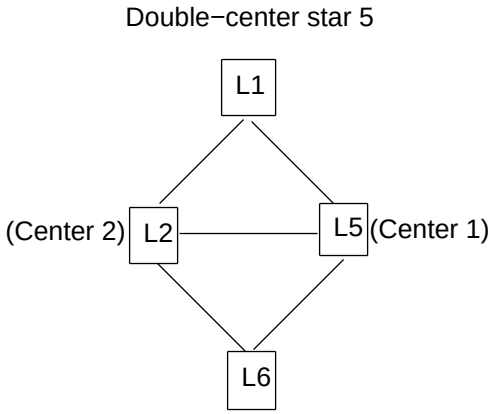


Figure 3.1: A double-center star

A double-center star is, as suggested by its name, a "star" with two centers. Each single-center star can be extended to a corresponding double-center star, while the latter is much more stable. In a single-center star, each local alignment consistent with the center is added to the star, while in a double-center star, a local alignment can be added only when it is consistent with both centers. The first center of a double-center star is exactly the one in a single-center star, and the second center is selected from that star. The selection of the second center satisfies the following conditions: (1) It is consistent with the first center; (2) It is long enough to make sense; (3) It is as far as possible from the first center. Figure 3.1 illustrates a double-center star corresponding to star 5 in Figure 2.3, supposing L_2 is the second center.

Each local alignment in a single-center star is consistent with the center, however, this does not automatically guarantee that all the local alignments in the star are mutually consistent. The consistency relationship is not necessarily transitive. In order to reduce the probability of adding inconsistent local alignments to the star, a double-center star accepts local alignments in a more prudent way. It rejects the local alignments originally surviving in the single-center star on a weak basis, therefore local alignments in the double-center star are more likely to be those very good ones. To some extent, the presence of the second center has the effect of "extending" the local alignment in the first center. With such a long "local alignment" as the center, the star will be more stable because points in it have much less freedom to move or rotate. From another aspect, with this improvement the extent to which

the initial transformation will change along with the deletion of bad point pairs is reduced significantly - the initial transformation will be closer to the final one, and thus the deletion will cause less unpredictability. Furthermore, the filtering of unpromising local alignments reduces their negative contribution to the overall transformation (as well as the number of point pairs involved in the initial star), speeding up the deletion process and resulting in a faster and better global alignment.

3.2 Development of Learning Ability

3.2.1 Self-learning

As we have mentioned, good star centers produce promising stars and have a greater probability of generating good global alignments. However, thus far the selection of star centers has been naïve: any local alignment with sufficient length can be the center of a star. The double-center method helps remove some unpromising local alignments from a star when the first center is determined, but it contributes nothing to the selection of the first center. If the first center of a star can be selected intelligently rather than by arbitrarily picking up a local alignment, then the star may yield a better global alignment. This intelligence may be difficult to achieve without any *a priori* knowledge on the global structural similarity between the two chains. However, when such knowledge is available, it is possible to improve the alignment by way of a *self-learning* strategy.

Once a currently optimal global alignment is output, we are able to know approximately where the aligned positions are. We organize the consecutively aligned point pairs into groups, and each group of consecutive point pairs is called a global alignment segment. A global alignment segment looks exactly like a local alignment, while as a part of a good global alignment, it should be a good "local alignment". Here local alignment is quoted because global alignment segments are not output of the local alignment phase, although there is no substantial difference between the two definitions. To take advantage of these good global alignment segments, we apply a feedback mechanism to teach our alignment algorithm how to improve itself. The self-learning is implemented *via* iterative utilization of its own output. When a global alignment is ready, consecutive alignment segments are extracted, then each segment is used as a new star center and local alignments consistent with the center are added to its group. This global alignment phase is repeated with a few new stars obtained from the currently optimal global alignment, until the alignment output converges (i.e. no changes are found between two iterations).

3.2.2 Learning from others

When a global alignment from another algorithm is available, the global alignment segments in that result can work as initial star centers. These centers are likely to be better than our own local alignments since they are from an optimal alignment result obtained from another algorithm. With these centers, our global alignment searching starts from a very good jumping-off point, therefore it is possible to output a result better than without learning. Learning from other algorithms may be more effective in the cases where our base algorithm performs worse than others. When it performs better than other algorithms even

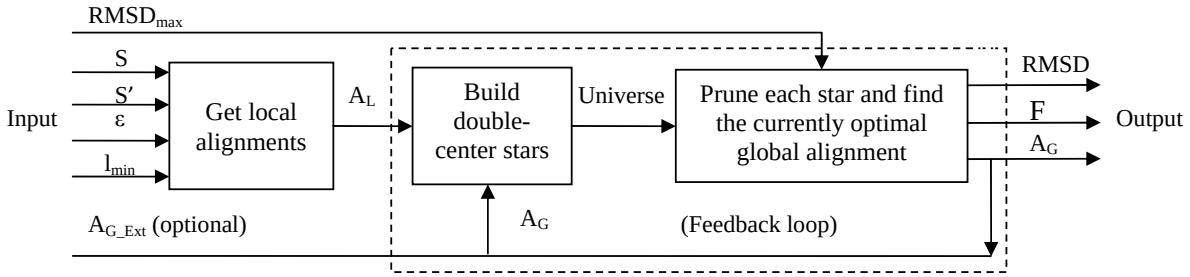


Figure 3.2: The SLIPSA framework

without learning, this learning may be less necessary, however it is never harmful, because if it results in a worse global alignment, its results can simply be disregarded. Therefore the combination of self-learning and learning-from-others will never output an alignment worse than that obtained from another algorithm. In the worst case it outputs nothing different. For this reason, our algorithm can also be used to improve the result of any other algorithm. We call this a refinement to that algorithm.

3.3 A Formal Description of SLIPSA

We give a formal description of SLIPSA. Combining the double-center star, the self-learning and the learning-from-others methods which use feedback, we greatly improve our earlier work [103] and have found interesting results when comparing SLIPSA with some other algorithms. The SLIPSA framework is shown in Figure 3.2. This system takes six parameters: protein chains S and S', $RMSD_{max}$ (a user specified maximum $RMSD$), distance constant ϵ, minimum local alignment length l_{min}, and an optional external global alignment A_{G_Ext}. Parameters S and S' are determined by the user, $RMSD_{max}$ is either determined by the user or obtained from another external algorithm, ϵ and l_{min} are selected empirically according to the user input, and A_{G_Ext} is either empty or also obtained from the external algorithm. The system outputs an optimal global alignment result consisting of A_G (a set of global alignment segments), $RMSD$ (a value not greater than $RMSD_{max}$) and F (a rigid body transformation corresponding to the final global alignment). Following sub-sections describe the details of the SLIPSA algorithm.

3.3.1 Getting local alignments

The calculation of local alignments has been reviewed in Section 2.2. The procedure used to get local alignments can be from either [103] or other related papers (e.g. [98]). The procedure body is omitted.

Get-Local-Alignments$(S, S', \epsilon, l_{\min})$
Input: protein backbone chains $S = p_1 \cdots p_n$, $S' = q_1 \cdots q_m$, distance constant ϵ and minimum local alignment length $l_{\min}$, where each p or q is a 3-D point corresponding to a C_α atom in a protein backbone.

Output: $A_L = \{L_1, L_2, \cdots, L_w\}$, a set containing all the local alignments of length $\geq l_{\min}$ between S and S'.

3.3.2 Building up stars from local alignments

The improved procedure outputs double-center stars. The input is star centers from a set of global alignment segments, or from a local alignment set when the former is empty. The non-center nodes in a star are still chosen from the local alignment set.

Build-Double-Center-Stars(A_L, A_G)
Input: $A_L = \{L_1, L_2, \cdots, L_w\}$ and $A_G = \{L_{1'}, L_{2'}, \cdots, L_{w'}\}$, where A_L is a set of local alignments and A_G is a set of global alignment segments.
Output: $Universe = \{Star_1, Star_2, \cdots, Star_k\}$, a set of all the unique double-center stars.
begin
    $Universe \leftarrow \{\}$ (the empty set);
    if ($A_G = \{\}$) **then** $A \leftarrow A_L$; **otherwise** $A \leftarrow A_G$;
    for (each local alignment L_u in A)
        find $L_{u'}$, the second center based on L_u, in A;
        $Star_u \leftarrow \{L_u, L_{u'}\}$;
        for (each local alignment L_v in A_L)
            if (L_v is consistent with both L_u and $L_{u'}$) **then** $Star_u \leftarrow Star_u \cup \{L_v\}$;
        end for
        if ($Star_u \notin Universe$) **then** $Universe \leftarrow Universe \cup \{Star_u\}$;
    end for
    return $Universe$;
end

3.3.3 Finding a global alignment from the stars

In each iteration of our algorithm, a global alignment is output and used as an input of the next iteration. We describe how to prune the set of aligned pairs in a star and obtain the global alignment which has an $RMSD$ not greater than that specified by the user. We refine a similar idea that was used in our original algorithm [103], which does not use feedback.

Prune-One-Star($Star, RMSD_{max}$)
Input: a $Star$ and $RMSD_{max}$ (a user specified maximum $RMSD$).
Output: $(A_S, RMSD_S, F_S, l_S)$, where $A_S = \{L_{1''}, L_{2''}, \cdots, L_{w''}\}$ is a set of global alignment segments which share a common transformation F_S with $RMSD_S \leq RMSD_{max}$, and l_S is the number of aligned point pairs in A_S.
begin
    $A_S \leftarrow Star$;
    $l_S \leftarrow$ the number of point pairs involved in A_S;
    calculate transformation F_S and $RMSD_S$ for all the point pairs involved in A_S;
    while ($RMSD_S > RMSD_{max}$)
        delete point pair (p, q) with the largest $d(p, F_S(q))$ in A_S;
        $l_S \leftarrow l_S - 1$;
        recalculate F_S and $RMSD_S$ for all the point pairs involved in A_S;

end while
return $(A_S, RMSD_S, F_S, l_S)$;
end

In the following function Find-Global-Alignment(), we apply the Prune-One-Star() procedure to each of the stars in the universe which is built from Build-Double-Center-Stars(). The alignment that contains the largest number of aligned pairs will be returned.

Find-Global-Alignment($Universe, RMSD_{max}$**)**
Input: $Universe = \{Star_1, Star_2, \cdots, Star_k\}$ and $RMSD_{max}$ (a user specified maximum $RMSD$).
Output: $(A_G, RMSD, F)$, where $A_G = \{L_{1'}, L_{2'}, \cdots, L_{w'}\}$ is a set of global alignment segments which share a common transformation F with $RMSD \leq RMSD_{max}$.
begin
sort $Universe$ by a descending order of the number of 3-D point pairs involved in each star;
$l_{max} \leftarrow 0$;
for (each $Star_u$ in $Universe$)
$(A_S, RMSD_S, F_S, l_S) \leftarrow$ Prune-One-Star($star_u, RMSD_{max}$);
if $(l_S > l_{max})$ **then** $A_G \leftarrow A_S$; $RMSD \leftarrow RMSD_S$; $F \leftarrow F_S$; $l_{max} \leftarrow l_S$;
end for
return $(A_G, RMSD, F)$;
end

3.3.4 The feedback procedure

This is the main procedure of SLIPSA. It calls Get-Local-Alignments in the first step, then Build-Double-Center-Stars and Find-Global-Alignment are called repeatedly. A global alignment output by the current iteration serves as the input of the next iteration. The procedure terminates when the global alignment ceases to change (i.e. converges).
SLIPSA($S, S', \epsilon, l_{min}, RMSD_{max}, A_{G\_Ext}$)
Input: S, S', ϵ, l_{min}, $RMSD_{max}$ and A_{G_Ext}, where A_{G_Ext} can be either empty or a set of global alignment segments obtained from an external algorithm.
Output: $(A_G, RMSD, F)$.
begin
$A_L \leftarrow$ Get-Local-Alignments(S, S', ϵ, l_{min});
$A_G \leftarrow A_{G\_Ext}$;
do
$A'_G \leftarrow A_G$;
$Universe \leftarrow$ Build-Double-Center-Stars(A_L, A'_G);
$(A_G, RMSD, F) \leftarrow$ Find-Global-Alignment($Universe, RMSD_{max}$);
while $(A_G \neq A'_G)$;
return $(A_G, RMSD, F)$;
end

When no external alignment is available, procedure SLIPSA is called by way of SLIPSA(S, S', ϵ, l_{min}, $RMSD_{max}$, {}). When it is available, SLIPSA can be called as SLIPSA(S, S', ϵ, l_{min}, $RMSD_{max}$, A_{G_Ext}). We call this a refinement to external alignment

A_{G_Ext}. To independently test the performance of our algorithm, none of the experiments reported in Section 3.4 uses any external alignment as our input.

3.4 Experimental Environment and Results

3.4.1 Our web alignment tool

We have developed a web alignment tool based on the SLIPSA algorithm. The website is available for public access at `http://fpsa.cs.uno.edu/` with a mirror site at `http://fpsa.cs.panam.edu/FPSA/`. It is not only a SLIPSA alignment tool but also an alignment comparison tool between SLIPSA and DaliLite, CE and SSM, some commonly used protein structure alignment algorithms with public websites [5–7].

The data used for protein alignment are PDB files downloaded from the RCSB Protein Data Bank. Now the files have been moved to the Worldwide Protein Data Bank (wwPDB) [8]. As of March 2008, there were over 49,000 protein structures with over 100,000 chains discovered.

Our website is built on an Intel dual-Xeon 3G Hz PC server with 3GB memory. The web development tools we have used include Apache HTTP server with PHP support, ActivePerl and MySQL database server. The SLIPSA algorithm is written in MATLAB. See [92] and [34] for the rigid body transformation method that we have used in SLIPSA.

The work flow of our website is shown in Figure 3.3. Besides a maximum value for $RMSD$, it accepts either PDB IDs or user uploaded PDB files as input. It is optional to compare SLIPSA with DaliLite, CE or SSM. When a comparing option is chosen, our tool automatically submits alignment request to and retrieves result from DaliLite, CE or SSM website, and performs SLIPSA alignment according to the retrieved $RMSD$ value. The website outputs the following alignment results. Except the first result, all others are optional depending on the user choices. Note that SLIPSA outputs A_G (a set of global alignment segments), $RMSD$ and F (a rigid body transformation).
(1) $(A_G, RMSD, F)_{\text{SLIPSA}}$: the SLIPSA result with a user specified $RMSD_{max}$.
(2) $(A_G, RMSD)_{DaliLite}$: the DaliLite result retrieved automatically from its website.
(3) $(A_G, RMSD, F)_{DaliLite\_Comp}$: the SLIPSA result with an $RMSD$ retrieved from DaliLite website as input. This result is used to compare SLIPSA with DaliLite.
(4) $(A_G, RMSD)_{CE}$: the CE result retrieved automatically from its website.
(5) $(A_G, RMSD, F)_{CE\_Comp}$: the result used to compare SLIPSA with CE.
(6) $(A_G, RMSD)_{SSM}$: the SSM result retrieved automatically from its website.
(7) $(A_G, RMSD, F)_{SSM\_Comp}$: the result used to compare SLIPSA with SSM.

3.4.2 Experimental results

We have collected 224 alignment cases to test the performance of our algorithm. The test cases were originally proposed by various papers for various testing purposes. Table 3.1 lists all the 224 cases. They include No. 1 - No. 20 (see Table III in [85]), No. 21 - No. 88 (see Table I in [38]), No. 89 (see Tables I and II in [85]), No. 90 - No. 92 (supplement to Table III in [85]), No. 93 (see Figure 5 in [85]), No. 94 - No. 101 (see Table IV in [85]), No. 102 - No. 111 (see Table V in [85]), No. 112 - No. 120 (supplement to Table V in

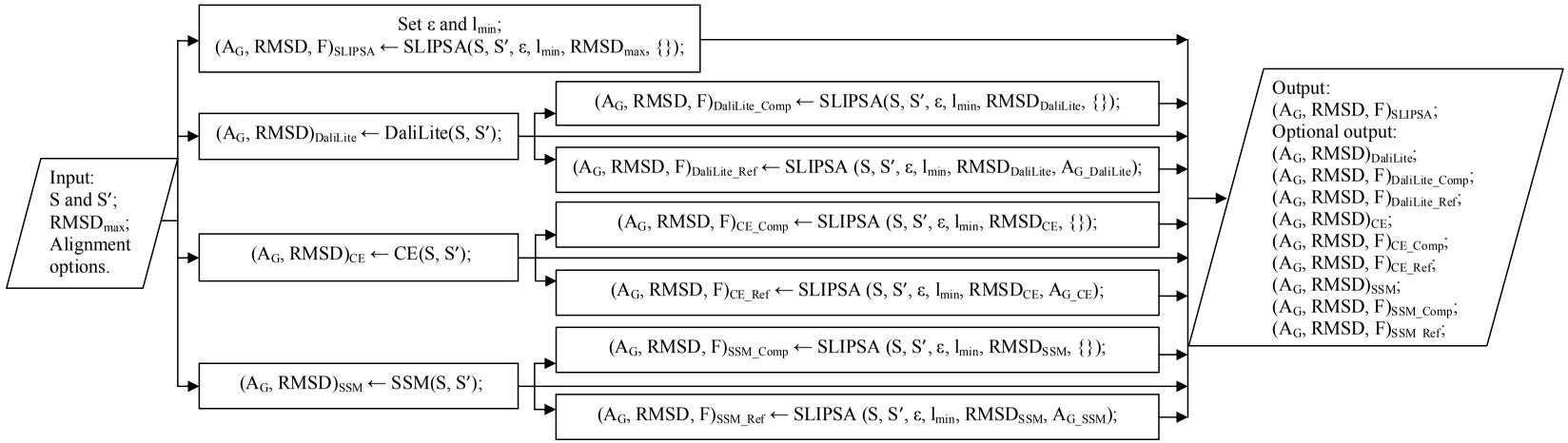


Figure 3.3: The web alignment work flow

Table 3.1: PDB chains of the 224 test cases

No.	Chain 1	Chain 2	No.	Chain 1	Chain 2	No.	Chain 1	Chain 2	No.	Chain 1	Chain 2
1	1ATP:E	1APM:E	57	2AK3:A	1GKY:_	113	1HLE:B	2ACH:B	169	1MBC:_	1PHN:A
2	1ATP:E	1CDK:A	58	1ATN:A	1ATR:_	114	1BBT:4	1TMF:4	170	1MBC:_	1CPC:A
3	1ATP:E	1YDR:E	59	1ARB:_	5PTP:_	115	1AIE:_	2FUA:_	171	1MBC:_	1LIA:A
4	1ATP:E	1CTP:E	60	2PIA:_	1FNB:_	116	1CPT:_	1FCT:_	172	1MBC:_	1CPC:B
5	1ATP:E	1PHK:_	61	3RUB:L	6XIA:_	117	1LBD:_	1PSM:_	173	1MBC:_	1QGW:C
6	1ATP:E	1KOA:_	62	2SAR:A	9RNT:_	118	4ICB:_	1CTD:A	174	1MBC:_	1LIA:B
7	1ATP:E	1KOB:A	63	3CD4:_	2RHE:_	119	2SEC:I	1EGP:A	175	1MBC:_	1COL:A
8	1ATP:E	1AD5:A	64	1AEP:_	256B:A	120	1SCE:A	1PUC:_	176	1MBC:_	2CP4:_
9	1ATP:E	1CKI:A	65	2MNR:_	4ENL:_	121	1BPI:_	1BUN:B	177	1MBC:_	1EUM:A
10	1ATP:E	1CSN:_	66	1LTS:D	1BOV:A	122	1BPI:_	5EBX:_	178	1MBC:_	1FPO:A
11	1ATP:E	1ERK:_	67	2GBP:_	2LIV:_	123	1WAJ:_	1NOY:A	179	1MBC:_	1OXA:_
12	1ATP:E	1FIN:A	68	1BBT:1	2PLV:1	124	1WAJ:_	1XWL:_	180	1MBC:_	1LE2:_
13	1ATP:E	1GOL:_	69	2MTA:C	1YCC:_	125	1ACX:_	1COB:B	181	1MBC:_	2FHA:_
14	1ATP:E	1JST:A	70	1TAH:A	1TCA:_	126	1ACX:_	1TMF:A	182	1MBC:_	1NFN:_
15	1ATP:E	1IRK:_	71	1RCB:_	2GMF:A	127	1PTS:A	1MUP:_	183	1MBC:_	1GRJ:_
16	1ATP:E	1FGK:A	72	1SAC:A	2AYH:_	128	2GBL:_	1UBQ:_	184	3TRX:_	4TRX:_
17	1ATP:E	1FMK:_	73	1DSB:A	2TRX:A	129	2GB1:_	4FXC:_	185	3TRX:_	1MDI:A
18	1ATP:E	1WFC:_	74	1STF:I	1MOL:A	130	1UBQ:_	4FXC:_	186	3TRX:_	1AIU:_
19	1ATP:E	1KNY:A	75	2AFN:A	1AOZ:A	131	1PLC:_	2RHE:_	187	3TRX:_	1ERV:_
20	1ATP:E	1TIG:_	76	1FXI:A	1UBQ:_	132	1PLC:_	1ACX:_	188	3TRX:_	1F9M:B
21	1MDC:_	1IFC:_	77	1BGE:B	2GMF:A	133	1ACX:_	1RBE:_	189	3TRX:_	1F9M:A
22	1NPX:_	3GRS:_	78	3HLA:B	2RHE:_	134	1ABA:_	1TRS:_	190	3TRX:_	1GH2:A
23	1ONC:_	7RSA:_	79	3CHY:_	2FOX:_	135	1ABA:_	1DSB:A	191	3TRX:_	1EP7:A
24	1OSA:_	4CPV:_	80	2AZA:A	1PAZ:_	136	1ABA:_	1PBF:_	192	3TRX:_	1EP7:B
25	1PFC:_	3HLA:B	81	1CEW:I	1MOL:A	137	1MJC:_	5TSS:A	193	3TRX:_	1FAA:A
26	2CMD:_	6LDH:_	82	1CID:_	2RHE:_	138	1PGB:_	5TSS:A	194	3TRX:_	1TOF:_
27	2PNA:_	1SHA:A	83	1CRL:_	1EDE:_	139	2TMV:P	256B:A	195	3TRX:_	2TIR:_
28	1BBH:A	2CCY:A	84	2SIM:_	1NSB:A	140	1TNF:A	1BMV:1	196	3TRX:_	1THX:_
29	1C2R:A	1YCC:_	85	1TEN:_	3HHR:B	141	1UBQ:_	1FRD:_	197	3TRX:_	1FB6:B
30	1CHR:A	2MNR:_	86	1TIE:_	4FGF:_	142	2RSL:C	3CHY:_	198	3TRX:_	1QUW:A
31	1DXT:B	1HBG:_	87	2SNV:_	5PTP:_	143	3CHY:_	1RCF:_	199	3TRX:_	1KTE:_
32	2FBJ:L	8FAB:B	88	1GPL:_	2TRX:A	144	1MBC:_	5MBN:_	200	3TRX:_	1JHB:_
33	1GKY:_	3ADK:_	89	1CPC:L	1COL:A	145	1MBC:_	1MBN:_	201	3TRX:_	3GRX:_
34	1HIP:_	2HIP:A	90	1KNY:A	1TIG:_	146	1MBC:_	1MYH:A	202	3TRX:_	1H75:A
35	2SAS:_	2SCP:A	91	1MAE:H	2BBK:J	147	1MBC:_	1HDS:B	203	3TRX:_	1EGO:_
36	1FCL:A	2FB4:H	92	2MHR:_	2BRD:_	148	1MBC:_	2DHB:A	204	3TRX:_	1ILO:A
37	2HPD:A	2CPP:_	93	1HCL:_	1JSU:A	149	1MBC:_	1MBA:_	205	3TRX:_	1ABA:_
38	1ABA:_	1EGO:_	94	2ASR:_	1OCC:C	150	1MBC:_	1DM1:A	206	3TRX:_	1FO5:A
39	1EAF:_	4CLA:_	95	2ASR:_	1MMO:D	151	1MBC:_	1HLM:_	207	3TRX:_	1MEK:_
40	2SGA:_	5PTP:_	96	2ASR:_	2BRD:_	152	1MBC:_	2LHB:_	208	3TRX:_	1A8Y:_
41	2HHM:A	1FBP:A	97	256B:A	1AEP:_	153	1MBC:_	2FAL:_	209	3TRX:_	1E2Y:A
42	1AAJ:_	1PAZ:_	98	256B:A	1CIY:_	154	1MBC:_	1HBG:_	210	3TRX:_	1E2Y:C
43	5FD1:_	1IQZ:A	99	256B:A	1AGS:A	155	1MBC:_	1ITH:A	211	3TRX:_	1QMV:A
44	1ISU:A	2HIP:A	100	2ASR:_	1LKI:_	156	1MBC:_	1FLP:_	212	3TRX:_	1BJX:_
45	1GAL:_	3COX:_	101	2ASR:_	1FPS:_	157	1MBC:_	1ECA:_	213	3TRX:_	1GP1:B
46	1CAU:B	1CAU:A	102	1LIS:_	1CIY:_	158	1MBC:_	2HBG:_	214	3TRX:_	1QQ2:A
47	1HOM:_	1LFB:_	103	1CFP:A	4ICB:_	159	1MBC:_	1ASH:_	215	3TRX:_	1EZK:A
48	1TLK:_	2RHE:_	104	1RPA:_	1HIW:A	160	1MBC:_	1HBI:B	216	3TRX:_	1EWX:A
49	2OMF:_	2POR:_	105	1HYP:_	1MZM:_	161	1MBC:_	1GDI:_	217	3TRX:_	1QK8:A
50	1LGA:A	2CYP:_	106	1CLC:_	1HOE:_	162	1MBC:_	1HLB:_	218	3TRX:_	1FG4:A
51	1MIO:C	2MIN:B	107	1UTG:_	1NOX:_	163	1MBC:_	1LH2:_	219	3TRX:_	1A8L:_
52	4SBV:A	2TBV:A	108	1FAR:_	1PTQ:_	164	1MBC:_	1H97:A	220	3TRX:_	1FG4:B
53	8I1B:_	4FGF:_	109	1KUM:_	1TUL:_	165	1MBC:_	1DLY:A	221	3TRX:_	1FVK:A
54	1HRH:A	1RNH:_	110	1PYI:A	1PYC:_	166	1MBC:_	1IDR:A	222	3TRX:_	1F37:A
55	1MUP:_	1RBP:_	111	1VIH:_	1PYT:A	167	1MBC:_	1DLW:A	223	3TRX:_	1F37:B
56	1CPC:L	1COL:A	112	1LYP:_	1OLG:A	168	1MBC:_	1ALL:A	224	3TRX:_	1GHH:A

[85]), No. 121 - No. 124 (see Table VII in [85]), No. 125 - No. 143 (see Table 1 in [79]), No. 144 - No. 183 (see Table 1 in [97]) and No. 184 - No. 224 (see Table 2 in [97]).

Based on this test set, we compare SLIPSA with DaliLite, CE and SSM in terms of N_{mat} (the number of aligned positions) and $RMSD$. Common protein alignment scoring methods such as Z-score, Q-score, P-score and geometric measures proposed in [57] all take N_{mat} and $RMSD$ into account. Due to the $RMSD$ flexibility of SLIPSA, it is easy to compare SLIPSA with DaliLite, CE and SSM on the basis of N_{mat} because in most cases SLIPSA outputs an equal $RMSD$. The detailed results are listed in Tables A.1 through A.3 in the appendix section A.1. In the tables, n and r stand for N_{mat} and $RMSD$, respectively, n^+ is the N_{mat} increment (%), and r^- is the $RMSD$ decrement (%). The results are sorted in a descending order of n^+. In each test case SLIPSA outputs an $RMSD$ not greater than that of DaliLite, CE, or SSM. If N_{mat} of SLIPSA is larger than N_{mat} of DaLiLite, CE, or SSM, we call it an N_{mat} increment. Similarly, if the $RMSD$ of SLIPSA is smaller than the $RMSD$ of DaLiLite, CE or SSM, we call it a $RMSD$ decrement. For the convenience of illustration, the results are sorted in a descending order of the N_{mat} increment rate. The N_{mat} increment rate is calculated by $(N_{mat\_\text{SLIPSA}} - N_{mat\_X})$ / N_{mat_X} and the $RMSD$ decrement rate is calculated by $(RMSD_X - RMSD_{\text{SLIPSA}})$ / $RMSD_X$, where X is DaliLite, CE or SSM. Figure 3.4 illustrates such increments and decrements in percentage. It should be mentioned that (1) no SSM comparison was performed in our earlier paper [103], (2) in a few cases that we could not find results on the DaliLite, CE or SSM websites, we marked the cases as "n/a" and did not compare SLIPSA with them, (3) from the time of completion of this dissertation, it is possible to see result changes on any of the alignment websites and we have observed minor changes on some of them, and (4) the SLIPSA experiments did not use any external alignment as input, although our algorithm is able to refine the alignment results retrieved from other web servers.

To verify our alignment results listed above, readers can access our website, input two protein backbones by giving their PDB IDs and chain letters, specify an $RMSD_{max}$ (default value is 3.0) and provide an email address to receive the result page. For example, when a user opens `http://fpsa.cs.uno.edu/`, types PDB ID "1ATP" and selects chain letter "E" as backbone 1, types "1APM" and selects chain "E" as backbone 2, and inputs his email address, he will receive an HTML result page showing that $N_{mat} = 336$, $RMSD = 0.3$ and the aligned positions are points 1 - 336 in protein chain 1ATP:E and points 1 - 336 in chain 1APM:E. The corresponding sequence alignment will also be shown in the page.

3.4.3 Discussion on the results

Table 3.2 shows some statistical data based on the results in Figure 3.4. For each case in which an alignment result was missing from either DaliLite, CE or SSM, we did not compare it with SLIPSA. Also, since DaliLite, CE and SSM may have different $RMSD$ values for a given test case, they were not compared mutually. In our experiments, when compared with DaliLite, CE and SSM respectively, SLIPSA outputs a larger N_{mat} in 66.67%, 61.82% and 86.70% of the cases; The maximum N_{mat} increment rate of SLIPSA is 65.33%, 64.58% and 109.09%; On average, SLIPSA increases 4.56%, 4.13%, and 7.37% of the N_{mat}; In 26.67%, 29.09% and 81.19% of the cases SLIPSA outputs a smaller $RMSD$ with the maximum $RMSD$ decrement rate being 13.21%, 11.11% and 16.56%. To sum up, in most

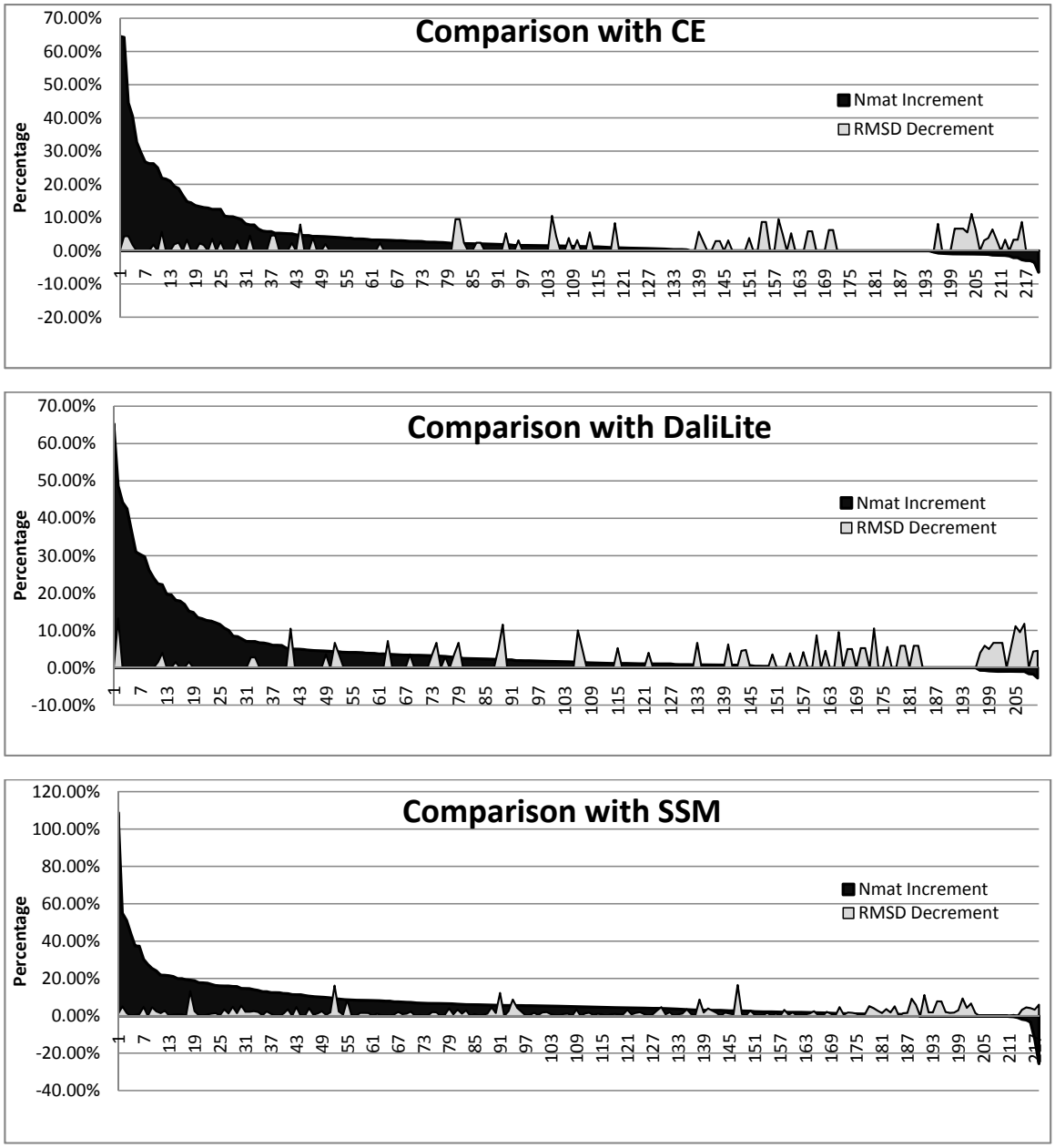


Figure 3.4: Comparing SLIPSA with DaliLite, CE and SSM

cases we see SLIPSA results with a larger or same N_{mat} and a same or smaller $RMSD$. In some cases that SLIPSA outputs a smaller N_{mat}, we also see a smaller $RMSD$.

Table 3.2: Statistics on the experimental results

	DaliLite	CE	SSM
Number of valid cases	210	220	218
Cases with larger N_{mat} by SLIPSA	149(66.67%)	136(61.82%)	189(86.70%)
Cases with smaller N_{mat} by SLIPSA	14(6.67%)	26(11.82%)	8(3.67%)
Maximum N_{mat} increment by SLIPSA	49	56	51
Maximum N_{mat} decrement by SLIPSA	2	9	12
Maximum N_{mat} increment rate by SLIPSA	65.33%	64.58%	109.09%
Maximum N_{mat} decrement rate by SLIPSA	2.74%	6.45%	25.53%
Average N_{mat} increment by SLIPSA	4.15	3.63	7.24
Average N_{mat} increment rate by SLIPSA	4.56%	4.13%	7.37%
Cases with smaller $RMSD$ by SLIPSA	56(26.67%)	64(29.09%)	177(81.19%)
Maximum $RMSD$ decrement by SLIPSA	0.7	0.4	0.52
Maximum $RMSD$ decrement rate by SLIPSA	13.21%	11.11%	16.56%
Average $RMSD$ decrement by SLIPSA	0.04	0.04	0.05
Average $RMSD$ decrement rate by SLIPSA	1.55%	1.42%	2.07%

We also attempt to compare SLIPSA with DaliLite, CE and SSM in the cases of weak similarities. To simplify the comparison process, we tentatively define a weak similarity as a large $RMSD$ between aligned chains. This definition may be incomplete, however, we have already observed some interesting results. For example, when compared with DaliLite and CE, the average N_{mat} increment rates of SLIPSA are 4.56% and 4.13% respectively, while in the cases with (C_α) $RMSD \geq 5.0$, the numbers are 26.48% and 21.62%, much higher than the overall average values. See Table 3.3 for details. In brief, SLIPSA obtains high average N_{mat} increment rate in weak similarity cases, and the larger the $RMSD$, the higher the average N_{mat} increment rate.

Table 3.3: Comparison based on weak similarity

	DaliLite		CE		SSM	
	Valid Cases	Avg. N_{mat} Inc.	Valid Cases	Avg. N_{mat} Inc.	Valid Cases	Avg. N_{mat} Inc.
$RMSD \geq 5.0$	12	26.48%	14	21.62%	0	/
$RMSD \geq 4.0$	20	23.48%	41	14.75%	9	17.50%
$RMSD \geq 3.0$	77	10.09%	102	7.64%	51	12.15%

The running time of each algorithm was recorded. The average running time of DaliLite, CE and SSM is 16.86s, 6.14s and 9.15s, respectively. When compared with them (i.e. using the $RMSD$ from the best fit from the comparison algorithms as the $RMSD$ upper-bound in SLIPSA), the average running time of SLIPSA is 105.97s, 69.89s and 81.43s, respectively. In about 50% of the cases the SLIPSA average time is below the DaliLite average, and the corresponding numbers for CE and SSM are about 25% and 28%, respectively. Possible ways to reduce the running time are discussed below.

(1) The web server was built on a slow machine. We have also tested the algorithm on an IBM ThinkPad laptop computer with Core2 Duo 1.8GHz CPUs. This machine was much slower than the mainstream web server machines, while the same results took only $\frac{1}{2}$ to $\frac{2}{3}$ of the time used on our current web server. It is possible to improve the speed to great extent by using a machine with high computational performance. (2) We used Matlab to implement the algorithm. Matlab facilitates the proof-of-concept development of complicated scientific programs, however, according to our experience it is possible to speed up algorithms at least several times if they are implemented in other languages such as

C, C++ and Java. In addition, parallel and distributed programming on high performance computational resources can also help reduce the execution time. (3) The algorithm is slower when the proteins are long and/or the $RMSD$ is large. In such cases the number of local alignments are large and the graph complexity is high. However, the algorithm can be tuned to reduce the complexity. Possible methods include reducing the dimension of data, removing unpromising local alignments as early as possible, limiting the number of times of feedback, and so on.

Chapter 4

Finding Proteins with Similar 3-D Structures in the Protein Data Bank

One application of protein structure alignment is the search for proteins with similar 3-D structures in a large protein structure databank (such as the PDB), given a protein structure as the input. The input may be a protein structure, which is described in certain format (e.g. the PDB format), or the 3-letter ID code of a structure in the protein databank. Algorithms for searching protein in the database for similar structures have been developed by multiple research groups [10,16,27,56,72,73,83,86]. In particular, the methods developed in [10, 16, 56, 72, 73, 83, 86] belong to the hierarchical method.

It is easy to see that our algorithm for the pair-wise protein backbone alignment can be applied to find similar proteins from the protein database. These algorithms are typically not fast enough to perform a brute force comparison with all proteins in the databank. One natural approach to deal with this complexity is to exclude proteins that have greatly different structures from the input structure by using some simple method, and then apply more complicated algorithms to check the similarity with small number of proteins left.

The hierarchical algorithm VAST [72] builds a bipartite graph. Each node in one side of the graph is a pair of secondary structure elements (SSEs) from the input protein, and each node in the other side of the graph is a pair of SSEs from the target protein. Connect two nodes between two sides if they can be aligned well. Their SSE alignment algorithm finds the maximal clique in the bipartite graph and extends it to C_α-atom level alignment by Gibbs sampling.

In this chapter[3], we develop a practical algorithm for the protein query problem. Our main technical contribution is that we apply a new method to check the similarity for the secondary structures between two proteins. Our method for grouping the secondary structures is different from other protein secondary structure alignment algorithms like [72]. Our approach is based on finding the star which has a center of two pairs aligned secondary structures between two proteins. Add other pairs of secondary structures to the star if there exists a common rigid body transformation between the center and the new pair. Prune the star until it has a satisfactory $RMSD$. Finding a maximal star, which can be computed in linear time, is easier than a maximal clique. This star based method was first used in our recent C_α-atom level alignment algorithm [103] and shows improvement over the existing alignment algorithms.

[3]Published work [71] used with permission of the CSREA Press.

We have found an efficient way to combine the secondary structure level alignment with the C_α-atom level alignment. The combination of two alignments are embedded into our protein query system so that it can find similar proteins in the protein databank of more than $100,000$ chains in a short time, and avoid missing similar structures.

The quality of protein query system is determined by how similar the list of output proteins is to the input protein. In order to compare the performance with other web-servers, we develop a model based on the symmetric difference between the two sets of protein structures which are outputted by two different software tools when the input is the same protein structure. It is compared with a similar tool of SSM and shows improved performance. It has been implemented as a web-server at address: `http://fpsa.cs.panam.edu/`.

4.1 Overview of Our Methods

Our algorithm has a series of filters. Given a protein 3-D structure as input, the algorithm first excludes those proteins that are greatly different in the number of C_α atoms in the protein backbone, the average distance from all the C_α atoms to the center of protein backbone, or the statistics about the secondary structures. The second layer filter does the second structure sequence alignment. The third layer filter aligns the secondary 3-D structures. The fourth layer filter uses a simplified version of our pair-wise alignment algorithm [103] to do the protein 3-D structure alignment.

4.2 Description of Algorithm

A straightforward method to find similar structures in the protein databank is to apply a pairwise protein structure alignment software to check all of the protein structures saved in the database. Since there are a large number of protein structures in the protein databank, it would be very slow to check each structure carefully. In the early stage, our algorithm has multiple phases to filter those structures that are weakly similar with the input protein structure. When there are a small number of candidate structures left, a more complicated pairwise algorithm is used in selecting the most similar protein structures.

4.2.1 Checking off-line information

We first reject those protein structures that have a greatly different number of C_α atoms in the backbone, the structures that have large differences for the average distance from the C_α-atoms to the center of C_α-backbone, and the structures that have large difference in ratio of α-helix among all secondary structures entities (SSEs). The number of C_α atoms in all the protein structures can be easily computed off-line. So, too, can the average distance to the center and the ratio of α-helix. This stage is very fast since a significant amount of off-line information describing structures in the large database have been preprocessed and is ready when a query is made. Therefore the rejection/acceptance of a structure can be decided very quickly.

Let S_0 be the input protein structure. We often use the parameter Structure-list to represent a list of protein structures which will be selected by checking similar properties with input protein structure S_0. For protein structure S, define $C(S)$ to be the C_α-chain of the backbone of S. Function Check-protein-size() checks if a target protein has a similar number of C_α atoms to the input protein S_0.

Check-protein-size(S_0, S)
Input: S_0 is the input protein structure, and S is another protein structure.
Output: $true$ or $false$.
Begin
Let n_0 be the number of C_α atoms in $C(S_0)$.
Let n be the number of C_α atoms in $C(S)$.
If ($|n_0 - n| \leq$ empirical_value$\cdot \max(n_0, n)$) **Then Return** $true$.
Else Return $false$.
End (of Check-protein-size)

For a list of points $p_1, \cdots, p_n$ in 3-D space, its center is computed by $\frac{\sum_{i=1}^{n} p_i}{n}$. We have the function Check-average-distance-to-center() to check if the average distance from all the C_α atoms to the center of $C_\alpha(S)$ is close to that of $C_\alpha(S_0)$. If two structures are similar their average distances to the center are also close.

Check-average-distance-to-center(S_0, S)
Input: S_0 is the input protein structure, and S is another protein structure.
Output: $true$ or $false$.
Begin
Let c_0 be the center of $C(S_0)$.
Let d_0 be the average distance from the C_α atoms of $C(S_0)$ to c_0.
Let c be the center of $C(S)$.
Let d be the average distance from the C_α atoms of $C(S)$ to c.
If ($|d_0 - d| \leq$ empirical_value$\cdot \max(d_0, d)$) **Then Return** $true$.
Else Return $false$.
End (of Check-average-distance-to-center)

The function Check-secondary-structure-statistics() is used to check the statistics information about the secondary structures such as the number of α-helixes and β sheets.

Check-secondary-structure-statistics(S_0, S)
Input: S_0 is the input protein structure, and S is another protein structure.
Output: $true$ or $false$.
Begin
Let a_0 be the number of α-helixes in $C(S_0)$.
Let b_0 be the number of β-sheets in $C(S_0)$.
Let a be the number of α-helixes in $C(S)$.
Let b be the number of β-sheets in $C(S)$.
If ($|a_0 - a| \leq$ empirical_value$\cdot \max(a_0, a)$ and $|b_0 - b| \leq$ empirical_value$\cdot \max(b_0, b)$)
Then Return $true$.
Else Return $false$.
End (of Check-secondary-structure-statistics)

The function Select-*via*-offline-information() filters proteins using the offline information. If the Structure-list is the list of proteins in the protein databank, there will be less than 20% proteins left after calling this function.

Select-*via*-offline-information(S_0, Structure-list)

Input: S_0 is the input protein structure, and Structure-list is the list of structures to be searched for similar proteins.

Output: a sublist of protein structures that each has a similar average distance to its center.

Begin

Let $L = \emptyset$.

For each protein structure S in L

Begin

If (Check-protein-size(S_0, S) and
Check-average-distance-to-center(S_0, S) and
Check-secondary-structure-statistics(S_0, S))

Then put S into L ($L = L \cup \{S\}$).

End (of For)

Return L.

End (of Select-*via*-offline-information)

4.2.2 Secondary sequence alignment

It has been observed that if two structures are similar, their secondary structure sequences can be well aligned at the sequence level, where each secondary structure can be represented by either α for an α helix or β for a β sheet. Using the α-β sequence alignment can reduce a large number of unrelated structures and greatly speed up the searching in the database.

The second structure sequences of all the protein structures in the databank are extracted. For each protein, its secondary structure sequence has format $s_1 s_2 \cdots s_k$ such that each s_i contains the following information:

- α-β type.
- Number of C_α atoms in the secondary structure. Define $c_\alpha(s_i)$ to be the number of C_α atoms in s_i.
- Two crucial points of the secondary structure.

Define the weight function such that $w(a, b) = \frac{\max(c_\alpha(a), c_\alpha(b))}{c_\alpha(a)+c_\alpha(b)}$ if the α-β type of a and b are different or one of them is a space, and $w(a, b) = \frac{|c_\alpha(a)-c_\alpha(b)|}{c_\alpha(a)+c_\alpha(b)}$ if the α-β type of a and b are the same.

An alignment of two sequences $a_1 \cdots a_n$ and $b_1 \cdots b_m$ of secondary structures will add gaps, each marked by a '-', into each. The first sequence becomes $a'_1 \cdots a'_k$ and the second sequence becomes $b'_1 \cdots b'_k$, and for each $1 \le i \le k$, at least one of a_i and b_i must not be a gap. The total cost for the alignment from $a'_1 \cdots a'_k$ and $b'_1 \cdots b'_k$ is measured by $\sum_{i=1}^{k} w(a'_i, b'_i)$. The optimal alignment is to find the one with the least cost by the function $w()$.

Define $D(i,j)$ to be the cost of an optimal alignment between $a_1 \cdots a_i$ and $b_1 \cdots b_j$. We have the following recursion, which implies that $D(n,m)$ can be computed in $O(mn)$ time by the method of dynamic programming.

$$D(i,j) = \min \begin{cases} D(i-1,j-1) + w(a_i, b_j) \\ D(i-1,j) + w(a_i, -), \\ D(i,j-1) + w(-, b_j) \end{cases} \tag{4.1}$$

Secondary-structure-sequence-alignment(S, S')
Input: S is the first protein structure, and S' is the second protein structure.
Output: an alignment for the secondary structure sequences between S and S'.
Begin
Let $s_1 s_2 \cdots s_k$ be the sequence of the first protein S.
Let $s'_1 s'_2 \cdots s'_{k'}$ be the sequence of the second protein S'.
Apply the dynamic programming with weight function $w()$.
Output the alignment with the best score.
End (of Secondary-structure-sequence-alignment)

We use function Select-*via*-secondary-structure-sequence() to select those proteins that have their secondary structure sequences aligned well with that of input protein structure S_0.
Select-*via*-secondary-structure-sequence(S_0**, Structure-list)**
Input: S_0 is the input protein structure, Structure-list is the list of structures to be searched for similar proteins.
Output: a sublist of protein structures that can be well aligned with S_0 according to the secondary structure sequence alignment.
Begin
Let $L = \emptyset$.
For each protein structure S in the Structure-list
Begin
A =Secondary-structure-sequence-alignment(S_0, S).
If (alignment A is good enough) **Then** put S into L.
End (of For)
Return L.
End (of Select-*via*-secondary-structure-sequence)

4.2.3 3-D alignment for secondary structures

In this phase, we select those protein structures that have good geometric alignment by secondary structures. This phase is also fast since each protein has about 30 secondary structures in average. We just use two points to represent a secondary structure.
Build-Star$((s_1, s'_1), (s_2, s'_2), S, S')$

Input: S and S' are two protein structures, s_1 and s_2 are secondary structures in S, s'_1 and s'_2 are secondary structures in S', and there exists an rigid body alignment for (s_1, s'_1) and (s_2, s'_2).
Output: a star with center at $(s_1, s'_1), (s_2, s'_2)$.
Begin
Let Center=$\{(s_1, s'_1), (s_2, s'_2)\}$.
Let Star=Center.
For each pair of secondary structures (s, s') between S and S'.
Begin
If (there exists a rigid body transformation for Center and (s, s'))
Then Let Star=Star $\cup\{(s, s')\}$.
End (of For)
Return Star.
End (of Build-Star)

The function Prune-star() deletes some pairs in a star until there exists an alignment with $RMSD$ less than a threshold r.
Prune-star(Star, r)
Input: Star is a star of secondary structures, and r is a threshold.
Output: a new star of secondary structures that has rigid body alignment with $RMSD$ no more than r.
Begin
While ($RMSD$(Star)$> r$)
Remove the pair (s, s') of Star that has the largest distance dist(s, s').
End (of Prune-star)

The function Secondary-structure-3-D-alignment() aligns the 3-D secondary structures between two protein structures S and S'. The method is based on building and pruning stars.
Secondary-structure-3-D-alignment(S, S')
Input: S is a protein structure; S' is a protein structure.
Output: a 3-D alignment between the secondary structures of S and S'.
Begin
L =Secondary-structure-sequence-alignment(S, S').
Best-star= $\emptyset$.
For each pair (s_1, s_2) of neighbor secondary structures in L
Begin
Star=Build-Star(s_1, s_2).
Star=Prune-star(Star, r)
If (size(best-star)$<$ size(Star)) **Then** Best-star=Star.
End (of For)
Return best-star as an alignment.
End (of Secondary-structure-3-D-alignment)

The function Select-*via*-secondary-structure-3-D-alignment() selects those proteins that can be well aligned with S_0 by the Secondary-structure-3-D-alignment function.

Select-*via*-secondary-structure-3-D-alignment(S_0, Structure-list)
Input: S_0 is the input protein structure, and Structure-list is the list of structures to be searched for similar proteins.
Output: a sublist of protein structures that can be well aligned with S_0.
Begin
Let $L = \emptyset$.
For each protein structure S in the Structure-list
Begin
A =Secondary-structure-3-D-alignment(S_0, S).
If (alignment A is good enough) **Then** put S into L.
End (of For)
Return L.
End (of Select-*via*-secondary-structure-3-D-alignment)

4.2.4 C_α-atom level pair-wise protein structure alignment

In the bottom layer of our implementation, we apply a protein pairwise alignment algorithm which was first developed in our earlier work [103]. In this layer, the protein structure alignment algorithm should balance the speed and accuracy.

Pair-wise-alignment(S, S')
Input: S is the first protein structure, and S' is the second protein structure.
Output: an alignment between the C_α atoms in S and S'.
Begin
Let C be the C_α-atom chain of S.
Let C' be the C_α-atom chain of S'.
Find the similar local regions between two C_α-chains.
Let each local alignment be a node of a graph.
Add an edge between two nodes if they share a common rigid body transformation.
For each star in the graph
Begin
Prune those C_α-pairs until the $RMSD$ is small enough.
End (of For)
Output the alignment with the largest number of C_α pairs.
End (of Pair-wise-alignment)

The function Select-*via*-atom-level-alignment() selects those proteins that can be well aligned with S_0 in the C_α atoms level.

Select-*via*-atom-level-alignment(S_0, Structure-list)
Input: S_0 is the input protein structure, and Structure-list is the list of structures to be searched for similar proteins.
Output: a sublist L of protein structures from Structure-list such that each protein structure in L has good C_α-atom alignment with S_0.

Begin

Let $L = \emptyset$.

For each protein structure S in the Structure-list

Begin

A =Pair-wise-alignment(S_0, S).

If (alignment A is good enough) **Then** put S into L.

End (of For)

Return L.

End (of Select-*via*-atom-level-alignment)

4.2.5 Combining them together

Now we put all the layers together to form the entire algorithm. The first filter is based on the offline information, the second filter is based on the secondary structure sequence (α-β sequence) alignment, the third filter is based on the secondary structure 3-D alignment, and the fourth filter is based on the C_α atoms alignment.

Search-proteins(S_0, Structure-list)

Input: S_0 is the input protein structure, and Structure-list is the list of structures to be searched for similar proteins.

Output: a list of proteins structures that are similar to S_0.

Begin

L_1 =Select-*via*-offline-information(S_0, Structure-list).

L_2 =Select-*via*-secondary-structure-sequence(S_0, L_2).

L_3 =Select-*via*-secondary-structure-3-D-alignment(S_0, L_3).

L_4 =Select-*via*-atom-level-alignment(S_0, L_3).

Return L_4 as the list of proteins similar to S_0.

End (of Search-proteins)

4.3 Experimental Results and Comparison with SSM

Our algorithm has been fully implemented and tested. It is available for public access at `http://fpsa.cs.panam.edu/`. It is running on a single machine, and will be supported by a cluster of machines soon.

4.3.1 Speed and performance

We have observed that given a protein structure as input for our software, it can output a list of protein structures that are similar to it among those available proteins in the protein databank.

The total number of protein chains is about $100,000$. After the top filter which checks the number of C_α atoms, the average distance to the center of C_α chain, and the ratio of SSEs, there are about $20,000$ structures left.

The second level filter aligns the secondary structure sequences and can filter 80% to 90% structures from the output of the top layer. We often see that there are less than 2000 structures left after the second level filter.

The third level filter narrows down the number of structures to several hundreds in average. It depends on how many proteins are really similar to the input protein. The atom level alignment tool developed in [103] is efficient enough to align the input structures with those protein structures in several minutes.

4.3.2 Model of comparison with other tools

Each web server outputs a list of proteins that are expected to have the maximal similarity with the input protein. When comparing with another web-server, we output the same number of results and check their symmetric difference. Assume that W_1 represents our software and W_2 represents another web-server. Given a protein structure p, $W_1(p)$ is the list of proteins similar to p by the server W_1. $W_2(p)$ is the list of proteins similar to p by the server W_2.

Let $W_1(p) - W_2(p)$ be the list of proteins that belong to $W_1(p)$ but not $W_2(p)$ and $W_2(p) - W_1(p)$ be the list of proteins that belong to $W_2(p)$ but not $W_1(p)$. The two lists are of the same length since we let $W_1(p)$ and $W_2(p)$ be of the same length.

According to SSM [59], we use Q-score to measure the quality of alignment between two protein structures. The Q-score is defined by the formula: $Q(p_1, p_2) = \frac{N_{align}^2}{(1+(RMSD/R_0)^2)N_1N_2}$, where N_{align} is the number of pairs of aligned C_α atoms, N_1 is the number of C_α-atoms in the protein p_1, N_2 is the number of C_α-atoms in the protein p_2, and R_0 is an empirical value (chosen at 3).

Table 4.1: PDB IDs of the 88 test cases

No.	ID	No.	ID	No.	ID	No.	ID
1	1avhA	23	1acx_	45	1etu_	67	1hsbA
2	1cpcL	24	1cd8_	46	1fx1_	68	1ltsA
3	1cpcA	25	1cdtA	47	1paz_	69	1ltsD
4	1brd_	26	1cid_	48	1pfkA	70	1ovb_
5	1babB	27	1dfnA	49	1q21_	71	1poc_
6	1eco_	28	1hilA	50	1s01_	72	1ppn_
7	1fcs_	29	1hivA	51	1sbp_	73	1rnd_
8	1fha_	30	1hleB	52	1sbt_	74	1snc_
9	1fiaB	31	1mamH	53	1timA	75	1tfg_
10	1hbg_	32	1reiA	54	1treA	76	1tgsI
11	1hddC	33	1ten_	55	1ula_	77	2achA
12	1le4_	34	1tlk_	56	1wayB	78	2bpa1
13	1mbc_	35	2alp_	57	2had_	79	2act_
14	1mbs_	36	2aviA	58	2liv_	80	2sns_
15	1utg_	37	2bpa2	59	3gbp_	81	3il8_
16	1troA	38	2snv_	60	5cpa_	82	3rubS
17	256bA	39	7apiB	61	5p21_	83	3sgbI
18	2ccyA	40	8fabA	62	8abp_	84	3sicI
19	2mhbA	41	8fabB	63	8atcA	85	4blmA
20	2mhbB	42	1aba_	64	1ctf_	86	4tms_
21	4mba_	43	1cseI	65	1dnkA	87	9rnt_
22	4mbn_	44	1dhr_	66	1eaf_	88	9rsaA

We select 88 proteins that are listed in [70] and belong to different categories such as $\alpha, \beta, \alpha/\beta$, and $\alpha + \beta$. See Table 4.1 for the names of those proteins. Table A.4 in appendix section A.2 lists the number of similar proteins obtained by our algorithm and SSM, respectively. Figures 4.1 and 4.2 give the comparison between SSM and our software

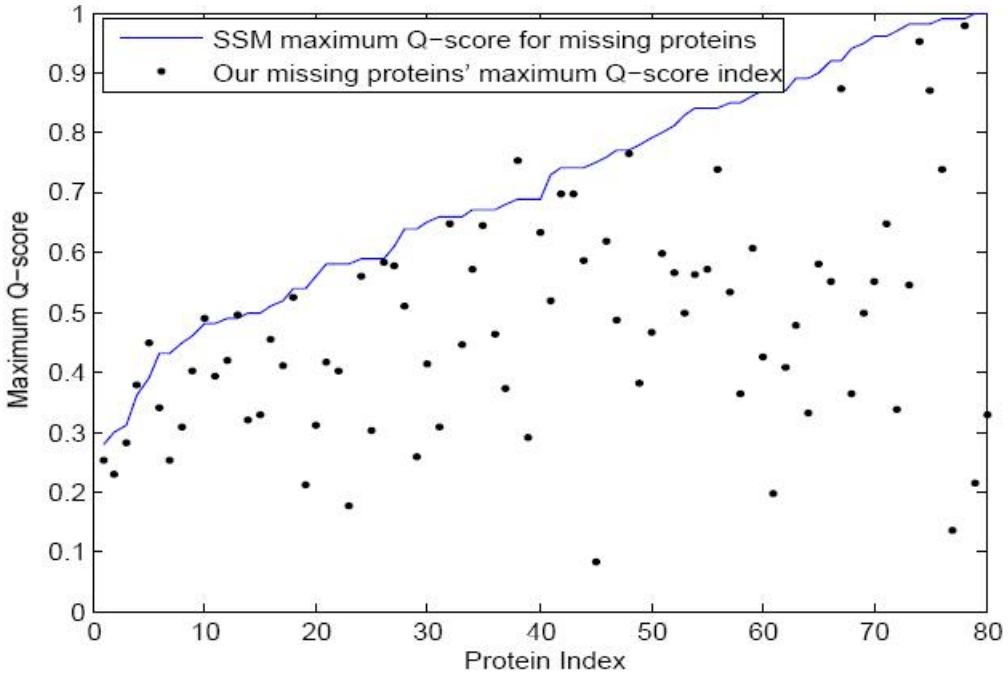


Figure 4.1: Comparison based on the maximum Q-score

based on the maximum and average Q-scores, respectively. The horizontal axis is the index of 80 proteins and the vertical axes in two figures are the maximum Q-score and the average Q-score, respectively. In our experiments, the eight proteins (1) 1avhA, (65) 1dnkA, (68) 1ltsA, (70) 1ovb_, (48) 1pfkA, (71) 1poc_, (78) 2bpa1, and (84) 3sicI have the same output between our software and SSM. Therefore, they are excluded from the results in Figures 4.1 and 4.2.

Define $MaxQ_1(p) = \max\{Q(p,p')|p' \in W_1(p) - W_2(p)\}$. Define $MaxQ_2(p) = \max\{Q(p,p')|p' \in W_2(p) - W_1(p)\}$. The curve in the first figure is the function $MaxQ_1(p)$ (assume that a protein p and its index are the same). Each point (p,q) in the curve has the relationship $q = MaxQ_1(p)$. On the other hand, a point (p',q') for a dot in the first figure has the relationship $q' = MaxQ_2(p')$. Figure 4.1 compares the maximum Q-scores of the proteins missed by the two software tools. We say that a protein is missed by our tool if it occurs in the SSM result list but not in ours; similarly, a protein is missed by the SSM tool if it occurs in our result list but not in the SSM's. For example, assume that for protein p, SSM returns m query results and our algorithm returns n results, then $\min(m,n) = r$ results with the best Q-scores are selected from each result set. Let $W_1(p)$ be the set of r best SSM results and $W_2(p)$ be the set of r best results of our algorithm. Assume that $W_1(p)$ and $W_2(p)$ have t same results, then $MaxQ_1(p)$ is the maximum Q-score of $r-t$ results that are outputted by SSM but missed by our algorithm. Similarly, $MaxQ_2(p)$ is the maximum Q-score of $r-t$ results that are outputted by our algorithm but missed by SSM. If $MaxQ_1(p) > MaxQ_2(p)$, then for protein p, we believe that our algorithm performs better than SSM. In Figure 4.1, SSM max Q-scores are sorted by an ascending order and shown as a curve, and the dots are the corresponding max Q-scores of our algorithm. Since most of

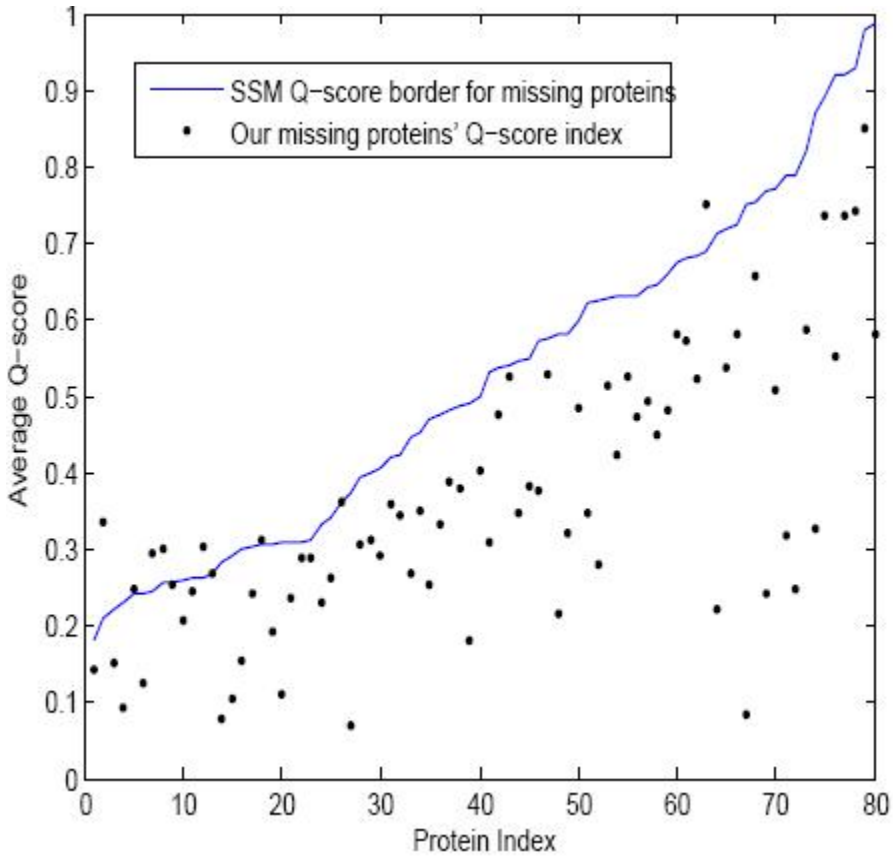


Figure 4.2: Comparison based on the average Q-score

the dots are below the curve, it indicates that the SSM query tool has a more serious missing problem than ours based on the maximum Q-score measure.

Define $AveQ_1(p) = \frac{\sum_{p' \in W_1(p) - W_2(p)} Q(p,p')}{|W_1(p) - W_2(p)|}$, where $|W_1(p) - W_2(p)|$ is the number of items in the set $W_1(p) - W_2(p)$. Define $AveQ_2(p) = \frac{\sum_{p' \in W_2(p) - W_1(p)} Q(p,p')}{|W_2(p) - W_1(p)|}$. The curve in the second figure is the function $AveQ_1(p)$ (assume that a protein p and its index are the same). Each point (p, q) in the curve has the relationship $q = AveQ_1(p)$. On the other hand, a point (p', q') for a dot in the second figure has the relationship $q' = AveQ_2(p')$. Figure 4.2 compares the average Q-scores of the proteins missed by the two software tools. Since most of the dots are below the curve, it indicates that the SSM query tool has a more serious missing problem than ours based on the average Q-score measure also.

Future work The algorithm currently runs on a single machine. We are building a cluster of machines to support the protein query system. It will be ready soon. Our current C_α-atom level alignment algorithm is not yet fast enough. We will try to use a load balance method to speed up the server in a cluster of PCs.

Chapter 5

Diameter of Protein Backbone and Sublinear Time Computations

Sublinear time computation is an active area of computer science in the recent years. A sublinear time algorithm has a sequence of elements $a_1, a_2, \cdots, a_n$ as input and can only access a part of the elements. Many sublinear time algorithms have been developed in the recent years. We give an incomplete list of sublinear time algorithms such as approximating matrix product [33], checking the polygon intersection [18], approximating the average degree in graph [37, 48], estimating the cost of minimum spanning tree [19, 30, 31], finding the geometric separators [41], computing the basis of abelian groups [21], property testing [47,78], and facility location [12]. Initially, the main research of sublinear time algorithms has been in the property testing with surveys in [40,45,46,60,82]. People tend to believe that there will be more and more sublinear time algorithms to emerge in the future. Therefore, it is important to study the power and limitation of sublinear time computations in both deterministic and randomized computation models.

A sublinear time algorithm usually uses a randomized method to access the input since it does not have enough time to see the entire input data. Most of the sublinear time algorithms developed in the recent years are randomized. A recent interesting derandomization approach by Zimand [105] showed that for some $\alpha > 0$, randomized algorithms of time complexity $T(n) < n^\alpha$ can be simulated by deterministic algorithms of time $\text{poly}(T(n))$ except on at most an $\exp(-\Omega(T(n) \log T(m))$ fraction of the instances.

In this chapter[4] we study the number of queries about the input sequence. In order to separate the power of sublinear time computations with different query complexity bounds, we select the problem to compute an approximate diameter for a sequence of points in a metric space. We realized this problem and its connection to sublinear time computation from our early research on the protein backbone alignment. From this approximate diameter problem, we show the existence of sublinear time algorithms at three different models, which are deterministic, bounded-error randomized, and zero-error randomized. We study the complexity of the sublinear time algorithms to approximate the diameter of a sequence of points. The separations of sublinear time computations under various complexity bounds and models in this chapter are based on various formulations of the diameter problem.

Three sublinear time computing models including deterministic, bounded-error randomized, and zero-error randomized models are studied in this chapter. We obtain a class

[4]Published work [42] used with permission of Springer Science and Business Media.

of separations about the power of sublinear time computations using several versions of the approximate diameter problem. We derive a dense sublinear time hierarchy for each of the three models. For every $0 < r < 1$ and $0 < \epsilon < r$, we show that the sublinear time deterministic computation with $O(n^r)$ queries to the input sequence is more powerful than sublinear time deterministic computation with $O(n^{r-\epsilon})$ queries and also the sublinear time deterministic computation with $O(n^r)$ queries to the input sequence cannot be simulated by sublinear time randomized computation with $O(n^{r-\epsilon})$ queries. We show that those separations by the number of queries imply similar dense time separations among sublinear time computations.

It is an interesting problem to identify what computational problems have the sublinear time algorithms. Our results show that the existence of sublinear time algorithms and their computational time depend on the restrictions on the format of input points in the metric space. We will show how those restrictions affect the existence of a sublinear time algorithm and its complexity.

We also show that the zero-error randomized sublinear time computation is more powerful than the deterministic sublinear time algorithm with similar time complexity and the bounded-error randomized sublinear time computation is more powerful than the zero-error randomized sublinear time algorithm with similar time complexity. We show that the bounded-error randomized sublinear time algorithms in time $O(n^r)$ cannot be simulated by a zero-error randomized sublinear time algorithm in $o(n)$ time or queries, where r is an arbitrary parameter in $(0, 1)$. We also show that zero-error randomized sublinear time algorithms in time $O(n^r)$ cannot be simulated by a deterministic sublinear time algorithm in $o(n)$ time or queries, where r is an arbitrary parameter in $(0, 1)$.

5.1 Definitions

A metric space S has a distance function dist(., .) that satisfies the following conditions: 1) $\text{dist}(p, p) = 0$ for every point $p \in S$; 2) $\text{dist}(p_1, p_2) = \text{dist}(p_2, p_1)$ for any two points $p_1, p_2 \in S$; and 3) $\text{dist}(p_1, p_3) \leq \text{dist}(p_1, p_2) + \text{dist}(p_2, p_3)$ for any three points $p_1, p_2, p_3 \in S$.

For an integer $d \geq 1$, R^d is the d-dimensinal Euclidean space, which is clearly a metric space.

Definition 5.1.

- Let $A = a_1, \cdots, a_n$ be a sequence of n points in a metric space. We often use $|A| = n$ to represent the number of points in A.
- Let $A = a_1, \cdots, a_n$ be a sequence of n points in a metric space. If for every pair of two consecutive points a_i and a_{i+1}, $\text{dist}(a_i, a_{i+1}) = t$, then the sequence A is called a t-sequence.
- Let $A = a_1, \cdots, a_n$ be a sequence of n points in a d-dimensional space. For every pair of two consecutive points a_i and a_{i+1}, if $t_1 \leq \text{dist}(a_i, a_{i+1}) \leq t_2$, then the sequence A is called a (t_1, t_2)-*sequence*. Define $\text{minInterDist}(A) = \min_{1 \leq i \leq n-1}(\text{dist}(a_i, a_{i+1}))$ and $\text{maxInterDist}(A) = \max_{1 \leq i \leq n-1}(\text{dist}(a_i, a_{i+1}))$.
- For a sequence of points A in a metric space, diameter(A) is the largest distance between two points of A.

- A real number d is $(1-\epsilon)$-*approximate* to the diameter of S of a sequence of points, if $(1-\epsilon)\text{diameter}(S) \leq d \leq \text{diameter}(S)$.

- A *path* of a randomized computation C of $r(n)$ random bits with the input sequence S is determined by a binary sequence B of length $r(n)$. Its output in the path B is denoted by $C(S, B)$.

- A *deterministic* $(1-\epsilon)$-*approximate algorithm* C with query complexity $q(n)$ for the diameter of sequence satisfies that 1) $C(S)$ is a $(1-\epsilon)$ approximation to diameter(S); and 2) C makes at most $q(n)$ queries to the points in S, where input S is a sequence of n points. Its query complexity is defined by a function $q(n)$ that for every input of length n points, the algorithm makes at most $q(n)$ queries. Its time complexity is defined by a function $t(n)$ that for every input of length n points, the algorithm stops in $t(n)$ steps.

- A *randomized* $(1-\epsilon)$-*approximate algorithm* C with $r(n)$ random bits for the diameter of sequence satisfies that 1) $C(S, B)$ is a $(1-\epsilon)$ approximation to diameter(S) with probability at least $\frac{3}{4}$; and 2) each path of C makes at most $q(n)$ queries to the points in S, where input S is a sequence of n points and B is a random binary sequence of length $r(n)$. A randomized algorithm can be also called *bounded-error* randomized algorithm.

- A *zero-error* randomized $(1-\epsilon)$-approximate algorithm C with $r(n)$ random bits for the diameter of sequence satisfies that 1) $C(S, B)$ is a $(1-\epsilon)$ approximation to diameter(S) with probability at least $\frac{3}{4}$; 2) no path gives a result that is not a $(1-\epsilon)$ approximation to diameter(S).

- A randomized $(1-\epsilon)$-approximate algorithm C (either bounded-error or zero-error) with $r(n)$ random bits and *time complexity* $t(n)$ for the diameter of sequence satisfies that $C(S, B)$ stops in $t(n)$ steps, where input S is an arbitrary sequence of n points and B is a random binary sequence of length $r(n)$.

- A randomized $(1-\epsilon)$-approximate algorithm C (either bounded-error or zero-error) with $r(n)$ random bits and *query complexity* $q(n)$ for the diameter of sequence satisfies that $C(S, B)$ makes at most $q(n)$ queries, where input S is an arbitrary sequence of n points and B is a random binary sequence of length $r(n)$.

A randomized approximate algorithm for the diameter problem is called *strict* if every path with non-empty output has the value no more than the diameter of the input sequence. All randomized approximate algorithms in this chapter are strict.

Sublinear time algorithms usually need some restriction about the input of data. For example, binary search takes $O(\log n)$ time, but it requires that the sequence of numbers is sorted. If a list of numbers is not sorted, any algorithm has to take $\Omega(n)$ time in the worst case.

5.2 Tight Separations among Sublinear Time Computations

We separate sublinear time computable functions with time complexity n^r from those with time complexity $n^{r-\epsilon}$ for any $0 < r < 1$ and any small $\epsilon > 0$. The separation is achieved in both deterministic and randomized computation models.

Definition 5.2. Let r be a nonnegative integer and $S = p_1p_2\cdots p_n$ be a (t_1, t_2)-sequence. The sequence $S' = p'_1p'_2\cdots p'_n$ is r-reliable rearrangement of S if S' is a permutation of $p_1p_2\cdots p_n$ and for each p_i, $p_i = p'_{i'}$ for some i' with $1 \le i' \le n$ and $|i - i'| \le r$.

Let M be a metric space and c, r, m, n be non-negative integers. Define $\Phi_M(c, r, m, n)$ to be the set of all sequences $H = q_1q_2\cdots q_n$ of n points in M such that H is an r-reliable rearrangement for a (t_1, t_2)-sequence S for some $0 < t_1 \le t_2$ with $\frac{t_2}{t_1} \le c$ and diameter$(S) \ge mt_1$. In particular, $\Phi_M(c, 0, m, n)$ is the set of all (t_1, t_2)-sequence S of length n in M with $\frac{t_2}{t_1} \le c$ and diameter$(S) \ge mt_1$. Sequence S is called a $\Phi_M(c, r, m, n)$-*sequence* if $S \in \Phi_M(c, r, m, n)$.

We first present a deterministic sublinear time approximate algorithm to compute the diameter of a t-sequence in a metric space. Its computational time is inversely proportional to the length of the diameter. The algorithm is described in a more general format by the following theorem and in the proof.

Theorem 5.3. *Assume that c is a positive constant, and α, μ and ϵ are constants in $(0, 1)$. Assume that M is a metric space with a $(1-\mu)$-factor approximate algorithm App_M of time complexity $C(k)$ for the diameter of k points in M for some nondecreasing function $C(k) : N \to N$. Then there exists a deterministic algorithm such that given a $\Phi_M(c, \frac{\epsilon(1-\alpha)}{2c}m, m, n)$-sequence B, it makes at most $O(\frac{n}{m})$ non-adaptive queries to the points of B and outputs a number x with $(1-\epsilon)(1-\mu)\cdot$diameter$(B) \le x \le$ diameter(B) in total time $O(\frac{n}{m})+C(O(\frac{n}{m}))$.*

Proof: Our algorithm selects an $O(\frac{n}{m})$ points set Q from the input sequence B and uses the diameter of Q to approximate the diameter of B. Select $\delta = \frac{\epsilon\alpha}{2c}$ and $\beta = \frac{\epsilon(1-\alpha)}{2c}m$. Assume that $A = p_1p_2\cdots p_n$ is a (t_1, t_2)-sequence such that B is a β-reliable rearrangement of A with $0 < t_1 \le t_2$, $\frac{t_2}{t_1} \le c$, and diameter$(A) \ge mt_1$. By the condition of the theorem, let $t_1 =$ minInterDist(A) and $t_2 =$ maxInterDist(A) be two positive real numbers with $t_1 \le t_2$ and $\frac{t_2}{t_1} \le c$. Our algorithm is described as follows:

Algorithm

Input: $B = p'_1, p'_2, \cdots, p'_n$ that is β-reliable-rearrangement of a (t_1, t_2)-sequence $A = p_1, p_2, \cdots, p_n$.

Output: an approximation x to diameter(A).

let $h = \lfloor \delta m \rfloor$;

select $q_i = p'_{h\cdot i}$ for $i = 1, \cdots, k = \left\lceil \frac{n}{h} \right\rceil$;

let Q be the sequence $q_1 \cdots q_k$;

output $x = App_M(Q)$;

End of Algorithm

Now we are going to prove that for the sequence Q constructed from B in the algorithm, $(1 - \epsilon)$diameter$(A) = (1 - \epsilon)$diameter$(B) \le$ diameter$(Q) \le$ diameter$(B) =$ diameter(A). Assume that p_i and p_j are two points in A such that dist$(p_i, p_j) =$ diameter(A).

Let i_1 be the number $1 \leq i_1 \leq k$ such that $|i_1 h - i| = \min_{1 \leq i_2 \leq k} |i_2 h - i|$ and j_1 be the number $1 \leq j_1 \leq k$ such that $|j_1 h - j| = \min_{1 \leq j_2 \leq k} |j_2 h - j|$. It is easy to see that $|i_1 h - i| \leq h$ and $|j_1 h - j| \leq h$. Since two consecutive points in A have distance at most t_2, we have

$$\text{dist}(p_i, p_{i_1 h}) \leq h \cdot t_2 \tag{5.1}$$

$$\text{dist}(p_j, p_{j_1 h}) \leq h \cdot t_2 \tag{5.2}$$

For each p'_k, it has another p_s such that $p_s = p'_k$ and $|s - k| \leq \beta$ since B is a β-reliable rearrangement of A. Therefore, we have

$$\text{dist}(p_k, p'_k) = \text{dist}(p_k, p_s) \leq \beta t_2. \tag{5.3}$$

We have the following inequalities:

$$\begin{aligned}
& \text{diameter}(A) && (5.4)\\
= \;& \text{diameter}(p_i, p_j) && (5.5)\\
\leq \;& \text{dist}(p_i, p_{i_1 h}) + \text{dist}(p_{i_1 h}, p'_{i_1 h}) + \text{dist}(p'_{i_1 h}, p'_{j_1 h}) + \text{dist}(p'_{j_1 h}, p_{j_1 h}) + \text{dist}(p_{j_1 h}, p_j) && (5.6)\\
\leq \;& h \cdot t_2 + \beta t_2 + \text{dist}(p'_{i_1 h}, p'_{j_1 h}) + \beta t_2 + h \cdot t_2 && (5.7)\\
\leq \;& h \cdot t_2 + \beta t_2 + \text{diameter}(Q) + \beta t_2 + h \cdot t_2 && (5.8)\\
\leq \;& 2(h + \beta) t_2 + \text{diameter}(Q) && (5.9)\\
\leq \;& 2(\frac{\epsilon\alpha}{2c} + \frac{\epsilon(1-\alpha)}{2c}) c \cdot m \cdot t_1 + \text{diameter}(Q) && (5.10)\\
\leq \;& \epsilon \cdot m t_1 + \text{diameter}(Q) && (5.11)\\
\leq \;& \epsilon \cdot \text{diameter}(A) + \text{diameter}(Q). && (5.12)
\end{aligned}$$

The transition from (5.5) to (5.6) is due to the triangle inequality in the metric space. The transition from (5.6) to (5.7) is due to inequalities (5.1), (5.2), and (5.3). The transition from (5.7) to (5.8) is because $p'_{i_1 h}$ and $p'_{j_1 h}$ are in Q. By (5.4)-(5.12), we have $(1 - \epsilon)\text{diameter}(A) \leq \text{diameter}(Q)$. On the other hand, all points in Q are from A. So, $\text{diameter}(Q) \leq \text{diameter}(A)$. Therefore, $(1 - \epsilon)\text{diameter}(A) \leq \text{diameter}(Q) \leq \text{diameter}(A)$. Since App_M gives factor $(1-\mu)$ approximation for the diameter of set Q, the output x satisfies $(1 - \epsilon)(1 - \mu) \cdot \text{diameter}(A) \leq x \leq \text{diameter}(A)$. Since B is a permutation of A, we have $\text{diameter}(B) = \text{diameter}(A)$. Therefore, $(1 - \epsilon)(1 - \mu) \cdot \text{diameter}(B) \leq x \leq \text{diameter}(B)$.

The number of queries of the algorithm is $|Q| = O(\frac{n}{m})$. The time for generating Q is $O(\frac{n}{m})$ and the time for computing $App_M(Q)$ is $C(O(\frac{n}{m}))$. ∎

Corollary 5.4. *Assume that α is a constant with $0 < \alpha < 1$, and ϵ is a small positive constant. Let t be a positive real number. Then there exists a deterministic $O(\frac{n}{m})$-time algorithm such that given an $\epsilon(1-\alpha)m/2$-*reliable*-rearrangement sequence B for a t-sequence A of n points in a metric space with diameter at least $m \cdot t$, it outputs a number x with $\frac{1-\epsilon}{2}\text{diameter}(B) \leq x \leq \text{diameter}(B)$.*

Proof: It is known that there exists an $O(k)$ time $\frac{1}{2}$-factor approximation algorithm to compute the diameter of k points in a metric space. The algorithm selects an arbitrary point

and finds the point with the largest distance, which is at least half of the diameter. Apply Theorem 5.3. ∎

Corollary 5.5. *Assume that α is a constant with $0 < \alpha < 1$, and ϵ is a small constant greater than 0. Let t be a positive real number. Then there exists a deterministic $O(\frac{n}{m})$-time algorithm such that given an $\epsilon(1-\alpha)m/2$-*reliable*-rearrangement sequence B for a t-sequence A of n points in R^1 with diameter at least $m \cdot t$, it outputs a number x with $(1-\epsilon)\text{diameter}(B) \leq x \leq \text{diameter}(B)$.*

Proof: In R^1, finding the diameter takes $O(k)$ time for an input of k points. ∎

Theorem 5.6 gives a lower bound about the randomized sub-linear time algorithms and matches the upper bound of Theorem 5.3.

Theorem 5.6. *Assume that ϵ is a constant in $(0,1)$ and $m = o(n)$. Then there is no randomized algorithm such that given a $\Phi_{R^1}(1,0,m,n)$-sequence S, the algorithm makes at most $o(\frac{n}{m})$ adaptive queries and outputs $(1-\epsilon)$-approximate diameter for S.*

Proof: See appendix section A.3. ∎

Corollary 5.5 and Theorem 5.6 imply the following corollaries, which give the dense separation for the sublinear time computations. It is easy to see that they imply the dense separation for the zero-error randomization computation since the power of zero-error randomized computation is between deterministic and bounded-error randomization computation.

Corollary 5.7. *Assume that ϵ is a constant in $(0,1)$. Then for every constant r in $(0,1)$ and constant δ in $(0,r)$, there is a function that can be $(1-\epsilon)$-approximated by n^r sublinear time deterministic algorithm, but there is no $n^{r-\delta}$ sublinear time $(1-\epsilon)$-approximate randomized algorithm.*

Corollary 5.8. *Assume that ϵ is a constant in $(0,1)$. For every constant r in $(0,1)$ and constant δ in $(0,r)$, there is a function that can be $(1-\epsilon)$-approximated by n^r sublinear time deterministic algorithm, but there is no $n^{r-\delta}$ sublinear time $(1-\epsilon)$-approximate deterministic algorithm.*

Corollary 5.9. *Assume that ϵ is a constant in $(0,1)$. Then for every constant r in $(0,1)$ and constant δ in $(0,r)$, there is a function that can be $(1-\epsilon)$-approximated by n^r sublinear time randomized algorithm, but there is no $n^{r-\delta}$ sublinear time $(1-\epsilon)$-approximate randomized algorithm.*

5.3 Comparing Randomized and Deterministic Computations

In this section, we show that randomized algorithms are more powerful than deterministic algorithms with the same computational time. We first present a randomized algorithm, then show that similar computation cannot be done in the deterministic algorithm with the similar complexity.

Theorem 5.10. *Assume that c is a positive constant, and α, μ and ϵ are constants in $(0,1)$. Assume that M is a metric space with a $(1-\mu)$-factor approximate algorithm App_M of complexity $C(k)$ for the diameter of k points in M for some nondecreasing function $C(k): N \to N$. Then there exists a randomized algorithm such that given a $\Phi_M(c,\infty,m,n)$-sequence B, it makes at most $O(\frac{n}{\epsilon m})$ non-adaptive queries to the points of B and outputs a number x with $(1-\epsilon)(1-\mu)\cdot \text{diameter}(B) \le x \le \text{diameter}(B)$ in total time $O(\frac{n}{\epsilon m}) + C(\frac{n}{\epsilon m})$, where $m = o(n)$.*

Proof: The algorithm selects a random $O(\frac{n}{\epsilon m})$ points set Q from the input sequence B and we show that diameter(Q) is close to the diameter of B with high probability. Select the constant $\delta = \frac{\epsilon}{2c}$. Select constant c_1 such that

$$(1 - \frac{1}{\frac{n}{\lfloor \delta m \rfloor}})^{\frac{c_1 n}{\lfloor \delta m \rfloor}} < \frac{0.001}{2}. \tag{5.13}$$

Our algorithm is described as follows:

Algorithm

Input: A sequence of n points $B = p'_1, p'_2, \cdots, p'_n$, which is a permutation of n points in a (t_1, t_2)-sequence $A = p_1, p_2, \cdots, p_n$ in a metric space with diameter$(A) \ge mt_1$.

Output: an approximation x to diameter(B).

let $h = \lfloor \delta m \rfloor$ and $k = \left\lceil \frac{c_1 n}{h} \right\rceil$;
select q_i from B randomly for $i = 1, \cdots, k$;
let Q be the sequence $q_1 \cdots q_k$;
output $x = APP_M(Q)$;

End of Algorithm

Now we are going to prove that for the sequence Q constructed from A in the algorithm, $(1-\epsilon)$diameter$(A) \le$ diameter$(Q) \le$ diameter(A). Assume that p_i and p_j are two points in A such that dist$(p_i, p_j) =$ diameter(A).

Partition the points $p_1 p_2 \cdots p_n$ into subsequences $P_1 P_2 \cdots P_t$ such that each P_i is of size at least h and at most $h+1$ for $i = 1, 2, \cdots, t$. Therefore, $\frac{n}{h+1} \le t \le \frac{n}{h}$. Assume that $p_i \in P_u$ and $p_j \in P_v$ for some u and v with $1 \le u, v \le t$. Since the size of P_i is at most $h+1$, the distance of two points in the same P_i is at most $t_2 h$.

We give the probability that $P_u \cap Q = \emptyset$ or $P_v \cap Q = \emptyset$. For a random point p from B, the probability that $p \notin P_u$ is $(1 - \frac{|P_u|}{n})$. The probability that $P_u \cap Q = \emptyset$ is $(1 - \frac{|P_u|}{n})^k$. Therefore, the probability that $P_u \cap Q = \emptyset$ or $P_v \cap Q = \emptyset$ is $(1 - \frac{|P_u|}{n})^k + (1 - \frac{|P_v|}{n})^k \le 2(1 - \frac{h}{n})^k < 0.001$ by the inequality (5.13).

Therefore, the probability is at least 0.99 that both $P_u \cap Q \neq \emptyset$ and $P_v \cap Q \neq \emptyset$. Now we assume that $p' \in P_u \cap Q$ and $p'' \in P_v \cap Q$. Therefore,

$$\begin{aligned}
\text{diameter}(Q) &\geq \text{dist}(p', p'') && (5.14)\\
&\geq \text{dist}(p_i, p_j) - \text{dist}(p_i, p') - \text{dist}(p_j, p'') && (5.15)\\
&\geq \text{diameter}(A) - t_2 h - t_2 h && (5.16)\\
&= \text{diameter}(A) - 2t_2 h && (5.17)\\
&\geq \text{diameter}(A) - 2ct_1\delta \cdot m && (5.18)\\
&\geq (1-\epsilon)\text{diameter}(A). && (5.19)
\end{aligned}$$

The transition from (5.14) to (5.15) follows from the triangle inequality in the metric space M. We have proven that $(1-\epsilon)\text{diameter}(A) \leq \text{diameter}(Q)$. On the other hand, all points in Q are from A. So, $\text{diameter}(Q) \leq \text{diameter}(A)$. Therefore, $(1-\epsilon)\text{diameter}(A) \leq \text{diameter}(Q) \leq \text{diameter}(A)$. Since APP_M is an $(1-\mu)$-approximate algorithm, we have $(1-\epsilon)(1-\mu)\text{diameter}(A) \leq x = APP_M(Q) \leq \text{diameter}(A)$. Since $\text{diameter}(A) = \text{diameter}(B)$, we have $(1-\epsilon)(1-\mu)\text{diameter}(B) \leq x \leq \text{diameter}(B)$. ∎

Corollary 5.11. *Assume that c is a positive constant, α, μ and ϵ are constants in $(0,1)$. Then there exists a randomized algorithm such that given a $\Phi_{R^1}(c, \infty, m, n)$-sequence B, it makes at most $O(\frac{n}{\epsilon m})$ non-adaptive queries to the points of B and outputs a number x with $(1-\epsilon) \cdot \text{diameter}(B) \leq x \leq \text{diameter}(B)$ in total time $O(\frac{n}{\epsilon m})$.*

Proof: There exists an $O(k)$ time algorithm such that given a set of k points in R^1, it returns the diameter, which is the difference between the leftmost and rightmost points. Apply Theorem 5.10. ∎

Theroem 5.12 gives a lower bound for the deterministic algorithms for computing the approximate diameter problem. Corollary 5.11 and Theroem 5.12 give the separation between randomized and deterministic computations.

Theorem 5.12. *Let ϵ be a constant in $(0,1)$ and $m = o(n)$. Then there is no deterministic algorithm that given a $\Phi_{R^1}(1, 8(\lceil \epsilon m \rceil + 2), m, n)$ sequence B, it makes no more than $(n - m - 1)/2$ adaptive queries to the input points and outputs a $(1-\epsilon)$-approximation to the diameter of B.*

Proof: See appendix section A.3. ∎

Definition 5.13.

- For a function $f(n) : N \to N$, a nondeterministic computation is a strong $\text{NQ}(f(n))$ if it makes at most $f(n)$ queries at each path, and has at least one path with non-empty output.
- An $\text{NQ}(f(n))$ computation is a $(1-\epsilon)$-approximation for the diameter problem if its every path with non-empty output gives $(1-\epsilon)$-approximation for the diameter.

Theorem 5.14 and Corollary 5.11 imply a separation between bounded-error randomized computation and zero-error randomized computation.

Theorem 5.14. *Assume that $\epsilon > 0$ is a constant and $m = o(n)$. Then there is no $NQ((n-m)/4)$ computation such that given a $\Phi_{R^1}(1, \infty, m, n)$-sequence S', it outputs a $(1-\epsilon)$-approximation to the diameter of B.*

Proof: See appendix section A.3. ∎

All randomized algorithms in this chapter for the diameter problem are strict as they only output distance no more than the diameter of the input sequence. It is easy to verify that the algorithms in Theorems 5.3, 5.10, 5.17, and 5.22 are all strict. With this condition, we have the following theorem to amplify the probability of the randomized algorithms for the diameter problem.

Theorem 5.15. *Assume that k is an integer and $\epsilon \in (0,1)$. Assume that A is a randomized $(1-\epsilon)$-approximate algorithm for computing the diameter of a sequence of points and runs in time $t(n)$ and makes $q(n)$ queries. Then there exists another randomized $(1-\epsilon)$-approximate algorithm with $kt(n)$ time and $kq(n)$ queries such that it outputs the $(1-\epsilon)$-approximate diameter with at most $\frac{1}{4^k}$ probability to fail.*

Proof: The revised algorithm A' is as follows: Randomly select k paths from A. Assume that $r_1, \cdots, r_k$ are the output from the k paths. Return the largest elements among $r_1, \cdots, r_k$. Since A has at most $\frac{1}{4}$ probability to fail, the failure probability is at most $\frac{1}{4^k}$. Since each path runs in $O(t(n))$ time and makes at most $q(n)$ queries, A' runs in $kt(n)$ time and makes $kq(n)$ queries. ∎

5.4 Zero-error Randomized Algorithm and Its Complexity

In this section, we show a zero-error randomized algorithm. We also derive a lower bound for the deterministic algorithms. This shows that zero-error randomized algorithms are more powerful than deterministic algorithms.

Definition 5.16. Let M be a metric space.

- Let $S' = q_1, q_2, \cdots, q_n$ be a rearrangement of a sequence of points $S = p_1 p_2, \cdots, p_n$. A point q_i is called a *still* point if $q_i = p_i$.
- A function $f(x) \to N$ can be *c-approximated* by a FZ$[n^r]$ computation algorithm if the algorithm makes at most n^r queries, gives output with probability at least $\frac{2}{3}$, and each output y has $cf(x) \leq y \leq f(x)$.
- Let $S' = q_1, q_2, \cdots, q_n$ be a rearrangement of a sequence of points $S = p_1 p_2, \cdots, p_n$. A point q_i in S' is called *v-stable* if $q_i = p_j$ with $|i-j| \leq v$.
- Let $S' = q_1, q_2, \cdots, q_n$ be a rearrangement of a sequence of points $S = p_1 p_2, \cdots, p_n$. S' is called *(u, v, α)-stable* if for every u consecutive points set Q from S', Q has at least αu v-stable points.

- For a sequence $S = q_1 q_2 \cdots q_n$ of points in M, the sequence $S^* = (q'_1, i_1)(q'_2, i_2) \cdots (q'_n, i_n)$ is called a *marked sequence* of S, where $(q'_1, i_1)(q'_2, i_2) \cdots (q'_n, i_n)$ is a permutation of $(q_1, 1)(q_2, 2) \cdots (q_n, n)$. Define $E(S^*) = S$.

- Let $\Lambda_M(c, m_1, m_2, r, m, n)$ be the set of all marked sequences $(q_1, a_1)(q_2, a_2) \cdots (q_n, a_n)$ such that 1) $S' = q_1 q_2 \cdots q_n$ is a permutation of a (t_1, t_2)-sequence $S = p_1 p_2 \cdots p_n$ of n points in M for some $0 < t_1 < t_2$ with $\frac{t_2}{t_1} \leq c$; 2) every m_1 consecutive points in S' have at least m_2 points q_i which are r-stable between S' and S; 3) the diameter of S is at least $m \cdot t_1$. and 4)$(q_1, a_1)(q_2, a_1) \cdots (q_n, a_n)$ is a permutation of $(p_1, 1)(p_2, 2) \cdots (p_n, n)$

- Let Γ be a class of marked sequences. A *zero-error randomized* $(1 - \epsilon)$*-approximate* algorithm C with $r(n)$ random bits for the diameter of sequence in Γ if for every input $S \in \Gamma$, we have 1) at least $\frac{3}{4}$ paths of C has non-empty output; and 2) each non-empty output in a path is a $(1 - \epsilon)$ approximation to diameter(S). Its time complexity and query complexity are defined similarly as that in Definition 5.1.

Theorem 5.17 shows a zero-error randomized algorithm to approximate the diameter of a marked sequence.

Theorem 5.17. *Assume that M is a metric space with a $(1 - \mu)$-factor approximate algorithm App_M of time complexity $C(k)$ for the diameter of k points in M for some nondecreasing function $C(k) : N \to N$. Then for every constant $\epsilon \in (0, 1)$, there exist positive constants β_1, β_2, and $\alpha < \beta_1$, and zero-error randomized $(1 - \epsilon)$-approximate algorithm such that given a $\Lambda_M(c, \beta_1 m, \alpha m, \beta_2 m, m, n)$-sequence $S' = (q_1, a_1) \cdots (q_n, a_n)$, the algorithm makes at most $O(\frac{n}{m} \log \frac{n}{m})$ non-adaptive queries to the items of S' and outputs a number x with $(1-\epsilon)(1-\mu) \cdot \text{diameter}(E(S')) \leq x \leq \text{diameter}(E(S'))$ in total time $O(\frac{n}{m}) + C(O(\frac{n}{m}))$, where $m = o(n)$.*

Proof: For a marked input sequence $S' = (q_1, a_1) \cdots (q_n, a_n)$, let $E(S')$ be a permutation of a (t_1, t_2)-sequence $S = p_1 p_2 \cdots p_n$ with $\frac{t_2}{t_1} \leq c$. A point q_i is called r-stable if it is r-stable between $E(S')$ and S. The algorithm selects $O(\frac{n}{\epsilon m})$ random points from $E(S')$ and puts those r-stable points into the set Q. We show that diameter(Q) is close to the diameter of S with high probability. Select β_1 and β_2 such that $((\beta_1 + 1)c + \beta_2) \leq \frac{\epsilon}{2}$. Let β_0 be a constant such that $\frac{\beta_0}{\alpha} > 1$. Our algorithm is described as follows:

Algorithm
Input: A $\Lambda(c, \beta_1 m, \alpha m, \beta_2 m, m, n)$-sequence $S' = (q_1, a_1)(q_2, a_2) \cdots (q_n, a_n)$.
Output: an approximation x to diameter$(q_1 q_2 \cdots q_n)$.
let $w = \beta_0 \frac{n}{m} \log \frac{n}{m}$;
select w items (q_i, a_i) from S' randomly
for $i = 1$ to w
if $|a_i - i| \leq \beta_2 m$ then put q_i into set Q;
output $x = APP_M(Q)$;
End of Algorithm

Assume $S' = (q_1, a_1)(q_2, a_2) \cdots (q_n, a_n)$. Select $w = \beta_0 \frac{n}{m} \log \frac{n}{m}$ tuples (q_i, a_i) randomly from S' and put q_i into the set Q if $|a_i - i| \leq \beta_2 m$. Assume that dist(q_{i_1}, q_{j_1}) is the diameter of S. Partition $E(S')$ into $Q_1 Q_2 \cdots Q_k$ such that each Q_i contains some consecutive points

in $E(S')$ and $\beta_1 m \leq |Q_i| \leq \beta_1 m + 1$. We have $k \leq \frac{n}{\beta_1 m}$. Assume that $q_{i_1} \in Q_{i'}$ and $q_{j_1} \in Q_{j'}$. Since each Q_i contains at least $\beta_1 m$ points, each Q_i contains at least αm $\beta_2 m$-stable points. With probability at most

$$(1 - (\frac{\alpha m}{n}))^w = (1 - (\frac{\alpha m}{n}))^{(\frac{\alpha n}{m})\frac{\beta_0}{\alpha}\log\frac{n}{m}} \leq (\frac{1}{2})^{\frac{\beta_0}{\alpha}\log\frac{n}{m}} \leq (\frac{m}{n})^{\frac{\beta_0}{\alpha}}, \tag{5.20}$$

no $\beta_2 m$-stable point is selected in Q_i. With probability at most $k(1 - (\frac{\alpha m}{n}))^w \leq k(\frac{n}{m})^{\frac{\beta_0}{\alpha}} = o(1)$, Q does not contain any $\beta_2 m$-stable point q_i marked with $|a_i - i| \leq \beta_2 m$ in Q_i for some i with $1 \leq i \leq k$.

With probability at least $1 - o(1)$, some $\beta_2 m$-stable points q_i marked with $|a_i - i| \leq \beta_2 m$ from both $Q_{i'}$ and $Q_{j'}$ will be selected. Check if each interval has marked $\beta_2 m$-stable points. With probability at least $1 - o(1)$, each Q_i has $\beta_2 m$-stable point q_i marked with $|a_i - i| \leq \beta_2 m$ selected. Assume that $q_{i''}$ is a $\beta_2 m$-stable point in Q_i, $q_{j''}$ is a $\beta_2 m$-stable point in $Q_{j'}$, and both $q_{i''}$ and $q_{j''}$ are in Q. The distance between q_{i_1} and $q_{i''}$ is at most $\beta_1 m t_2 \leq \beta_1 m \cdot c t_1 \leq \frac{\epsilon}{2} m t_1 \leq \frac{\epsilon}{2}\text{dist}(q_{i_1}, q_{j_1})$. Similarly, we have $\text{dist}(q_{j_1}, q_{j''}) \leq \frac{\epsilon}{2} m \leq \frac{\epsilon}{2}\text{dist}(q_{i_1}, q_{j_1})$. By the triangle inequality in a metric space, we have $\text{dist}(q_{i''}, q_{j''}) \geq \text{dist}(q_{i_1}, q_{j_1}) - \text{dist}(q_{i_1}, q_{i''}) - \text{dist}(q_{j_1}, q_{j''}) \geq \text{dist}(q_{i_1}, q_{j_1}) - \frac{\epsilon}{2}\text{dist}(q_{i_1}, q_{j_1}) - \frac{\epsilon}{2}\text{dist}(q_{i_1}, q_{j_1}) = (1 - \epsilon)\text{dist}(q_{i_1}, q_{j_1})$. When a path selects at least one $\beta_2 m$-stable point from each Q_i and puts it into Q, the path gives $(1 - \epsilon)$-approximation for the diameter. ∎

We have the following theorem to separate the sublinear time zero-error randomized computations from sublinear time deterministic computations.

Theorem 5.18. *Assume that c is a positive constant, ϵ is a constant in $(0, 1)$, β is a constant in $(0, c)$, and $m = o(n)$. Then there is no deterministic algorithm such that given a $\Lambda_{R^1}(1, cm, \beta m, 0, m, n)$-sequence S' it makes $o(n)$ adaptive queries to the input and outputs a $(1 - \epsilon)$ approximation to the diameter of $E(S')$.*

Proof: See appendix section A.3. ∎

5.5 A Consideration for Random Bits

We need $O(\log n)$ random bits to generate a random position between 0 and $n - 1$ so that we can randomly select one element in the input sequence in our randomized algorithms (Theorems 5.10 and 5.17). The following theorem shows that the conversion has small difference in probability by using the modular operation.

Theorem 5.19. *Assume that m and n are two integers with $m \leq n$. Then for a random number $r \in [0, n-1]$, $r' = r (\text{mod } m)$ is a number in $[0, m-1]$ such that for any two integers $a, b \in [0, m-1]$, $|\Pr[r' = a] - \Pr[r' = b]| \leq \frac{1}{n}$.*

Proof: For the integers in $\{0, 1, \cdots, n-1\}$, if they are converted to integers in $\{0, 1, \cdots, m-1\}$, every integer in $\{0, 1, \cdots, m-1\}$ is hit at most $\left\lceil \frac{n}{m} \right\rceil$ and at least $\left\lfloor \frac{n}{m} \right\rfloor$ times. We have $\left\lceil \frac{n}{m} \right\rceil - \left\lfloor \frac{n}{m} \right\rfloor \leq 1$. Thus, $|\Pr[r' = a] - \Pr[r' = b]| \leq \frac{\left\lceil \frac{n}{m} \right\rceil}{n} - \frac{\left\lfloor \frac{n}{m} \right\rfloor}{n} \leq \frac{1}{n}$. ∎

If we use $2\log n$ bits to generate a random number in the range $[0, n-1]$, the probability difference between any two different numbers in $[0, n-1]$ is at most $\frac{1}{n^2}$. This does not affect the performance of our randomized algorithms in this chapter.

5.6 Self-avoiding Sequences

In this section, we develop a sublinear time algorithm to approximate a self-avoiding sequence in any fixed dimensional Euclidean space R^d. We also prove a tight lower bound about the query complexity of approximation algorithms.

Definition 5.20. A point $p = (i_1, \cdots, i_d)$ in d-dimensional space is a grid point if $i_1, \cdots, i_d$ are all integers. A sequence of grid points $p_1 p_2 \cdots p_n$ in the d-dimensional space is *self-avoiding* if $p_i \neq p_j$ for $i \neq j$, and $\text{dist}(p_i, p_{i+1}) = 1$ for $i = 1, 2, \cdots, n-1$.

Self-avoiding sequences are common in nature. One example is the C_α atoms in the backbone of a protein molecule. The distance of two C_α atoms is larger than a threshold and the distance between two consecutive C_α atoms is almost fixed.

The volume of the d-dimensional ball of radius R is computed by the following formula (See [91]):

$$V_d(R) = \begin{cases} \frac{2^{(d+1)/2}\pi^{(d-1)/2}}{1\cdot 3\cdots(d-2)\cdot d} R^d & \text{if } d \text{ is odd} \\ \\ \frac{2^{d/2}\pi^{d/2}}{2\cdot 4\cdots(d-2)\cdot d} R^d & \text{otherwise} \end{cases} \tag{5.21}$$

Let $V_d(r) = v_d \cdot r^d$, where v_d is constant for fixed dimensional number d. In particular, $v_1 = 2, v_2 = \pi$ and $v_3 = \frac{4\pi}{3}$.

Lemma 5.21. *Let $d \geq 1$ be the dimension number. Then (i) every d-dimensional ball of radius r intersects at most $v_d \cdot (\frac{r+\sqrt{d}h}{h})^d$ cubes of size h^d; and (ii) every d-dimensional ball of radius r contains at least $v_d \cdot (\frac{r-\sqrt{d}h}{h})^d$ cubes of size h^d.*

Proof: Every two points inside a cube of size h have distance at most $\sqrt{d}h$. (i) If an h^d cube C intersects a ball B of radius r at center o, then C is fully contained by the ball B' of radius $r + \sqrt{d}h$ at the same center o. As the volume of cube C is h^d, the number of h^d cubes fully contained by C' is no more than the volume size of the ball C' divided by h^d. (ii) If an h^d cube C intersects a ball B'' of radius $r - \sqrt{d}h$ at center o, then C is contained in the ball B of radius r at the same center o. The number of those cubes of size h^d intersecting C'' is at least the volume size of the ball C'' divided by h^d. ∎

We have Theorem 5.22 and Corollary 5.23 for the sublinear time algorithm to compute the approximate diameter for any self-avoiding sequence in any fixed dimensional Euclidean space.

Theorem 5.22. *Assume that ϵ and δ are positvie constants, and d is a dimension number with $1 \leq d < (1-\delta)v_d^{\frac{-1}{d}} n^{\frac{1}{d}}/2$. Then*

i. There exists an algorithm that given a self-avoiding grid sequence S of length n, it queries at most $O(v_d^{\frac{1}{d}} n^{1-\frac{1}{d}})$ points in R^d and outputs a $(1-\epsilon)$-approximation for the diameter of S.

ii. If d is a fixed dimensional number, then there exists an $O(n^{1-\frac{1}{d}})$ time algorithm to compute a $(1-\epsilon)$-approximation for the diameter of a self-avoiding sequence S in R^d.

Proof: Assume that u is the diameter of the self-avoiding grid sequence S. Then all the grid points in the sequence S are contained in a ball of radius u. By Lemma 5.21, $u \geq v_d^{\frac{-1}{d}} n^{\frac{1}{d}} - \sqrt{d} = \Omega(v_d^{\frac{-1}{d}} n^{\frac{1}{d}})$. By Theorem 5.3, we have proved the first part of Theorem 5.22.

As in the proof of Corollary A.1, there exists an $O(k + (\frac{1}{\delta^{2d}})) = O(k)$ time $(1-\delta)$-factor approximate algorithm App_{R^d} to compute the diameter of k points set H in R^d. By Theorem 5.3, we have proved the second part of Theorem 5.22. ∎

Corollary 5.23. *For any constant $1 > \epsilon > 0$, there exists an $O(\sqrt{n})$-time deterministic algorithm that given a self-avoiding grid sequence S of length n in R^2, it computes a $(1-\epsilon)$-approximation for the diameter of S.*

Corollary 5.24. *For any constants $\epsilon, \delta > 0$ and fixed dimension number $d > 0$, there exists an algorithm that given a self-avoiding grid sequence S of length n, it queries at most $O(n^{1-\frac{1}{d}})$ points in R^d and outputs a $(1-\epsilon)$-approximation for the diameter of S.*

We have Theorem 5.25 for the time lower bound for approximating the diameter of a self-avoiding sequence in R^d. The lower bound matches the upper bound in Theorem 5.22.

Theorem 5.25. *Assume that d is a fixed dimension number. Then for any constant $c \in (0,1)$, there is no deterministic algorithm that uses $o(n^{1-\frac{1}{d}})$ queries to compute a c-approximate diameter for a self-avoiding grid sequence in R^d.*

Proof: See appendix section A.3. ∎

Future work By studying the approximate diameter problem for a sequence of points in a metric space, we have obtained a class of separations of some sublinear time computations. The sublinear time algorithms developed in this chapter may be used to other related problems, such as the furthest neighbor problem which is extensively applied in database and streaming applications. We expect to see further complexity research about the power of sublinear time computations.

Chapter 6

Reconstructing Haplotypes from SNP Matrices

6.1 Introduction

Abstractly, a genome can be considered a string over the alphabet of nucleotides $\{A, G, C, T\}$. It is accepted that the genomes between any two humans are over 99% identical [53, 90]. The remaining sites which exhibit substantial variation (in at least 5% of the population) are called Single Nucleotide Polymorphisms (SNPs). The values of a set of SNPs on a particular chromosome copy define a *haplotype*. While a haplotype is a string over the four nucleotide bases, typical SNP sites only vary between two values. Therefore, we can logically represent a haplotype as a binary string over the alphabet $\{A, B\}$. The haplotype of an individual chromosome can be thought of as a "genetic fingerprint" specifying the bulk of the genetic variability among distinct members of the same species. Determining haplotypes is thus a key step in the analysis of genetic variation.

More concretely, we assume we are given m strings of length n, each over the alphabet $\{A, B, -\}$ denoting the sequence of each fragment, where $-$ denotes a *hole* in which no value is measured at that position for that fragment. From this basic setup, a number of problems have been considered for various optimization functions. In particular, one can consider removing certain fragments from the input data such that the remaining set of fragments can be divided into two separate sets, each set consisting of fragments that do not contain any conflicts (a site in which two strings contain different, non-hole values). Such a removal implicitly derives two haplotypes corresponding to the two groups. This problem is referred to as the minimum fragment removal problem (MFR). A second problem involves removing or discounting a subset of the SNP sites to create two consistent sets of fragments. This is referred to as the minimum SNP removal problem (MSR). Finally, a third problem is to flip or correct a number of sites for various fragments. This is referred to as the minimum error correction problem (MEC).

In this chapter[5] , we develop a probabilistic approach to overcome some of the difficulties caused by the incompleteness and inconsistency occurred in the input fragments. In our model, we assume the input data fragments are attained from two unknown haplotype strings. Some number of the fragments are derived from the first haplotype, some from the second haplotype. Each fragment is assumed to be generated according to two error parameters, α_1 and α_2, representing inconsistency errors and incompleteness errors respectively.

[5]Published work [23] used with permission of Imperial College Press and Mary Ann Liebert Inc.

These parameters represent the percentage chance that any given position will be sequenced incorrectly from the base haplotype, or will be omitted as a hole, respectively. Further, we assume a third parameter β denoting a minimum difference percentage between the two base haplotype strings. In the simulated SNP matrix in Figure 1.6 there are 10 fragments and each one has 50 SNP sites. The inconsistent error rate α_1 is 4%, the incomplete error rate α_2 is 10%, and the dissimilarity rate β between two haplotypes is 20%.

With respect to these three parameters, we develop algorithms that output the base haplotypes with probability as a function of α_1, α_2, and β. In particular, we design three algorithms in our probabilistic model that can reconstruct the two unknown haplotypes from the given matrix of haplotype fragments with provable high probability and in time linear in the size of the input matrix. The first algorithm assumes prior knowledge of the α_1 parameter. The second algorithm does not require prior knowledge of any parameters at a modest cost to run time. The third does not require prior knowledge of parameters and attains similar analytical bounds as the second algorithm, yet exhibits a substantial improvement in accuracy.

For all algorithms, we present experimental results that conform with the respective theoretically efficient performance. The software of our algorithms is available for public access and for real-time on-line demonstration.

This chapter is organized as follows: in Section 6.2 we formally define the problem and the probabilistic model we use, in Section 6.3 we develop some technical lemmas, in Section 6.4 we provide a haplotype reconstruction algorithm that utilizes prior knowledge of an error parameter, in Section 6.5 we present an algorithm that requires no prior knowledge of model parameters, in Section 6.6 we provide a third algorithm with provably good accuracy that also performs particularly well in practice, and in Section 6.7 we detail experimental results for our algorithms.

6.2 A Probabilistic Model

Assume that we have two haplotypes H_1, H_2, denoted as $H_1 = a_1a_2\cdots a_m$ and $H_2 = b_1b_2\cdots b_m$. Let $\Gamma = \{S_1, S_2, \ldots, S_n\}$ be a set of n fragments obtained from the DNA sequencing process with respect to the two haplotypes H_1 and H_2. In this case, each $S_i = c_1c_2\cdots c_m$ is either a fragment of H_1 or H_2. Because we lose the information concerning the DNA strand to which a fragment belongs, we do not know whether S_i is a fragment of H_1 or H_2. Suppose that S_i is a fragment of H_1. Because of reading errors or corruptions that may occur during the sequencing process, there is a small chance that either $c_j \neq$ - and $c_j \neq a_j$, or $c_j =$ -, for $1 \leq j \leq m$, where the symbol - denotes a hole or missing value. For the former, the information of the fragment S_i at the j-th SNP site is inconsistent, and we use α_1 to denote the rate of this type of inconsistency error. For the latter, the information of S_i at the j-th SNP is incomplete, and we use α_2 to denote the rate of this type of incompleteness error. It is known (e.g., [13,69,93]) that α_1 and α_2 are in practice between 3% to 5%. Also, it is realistically reasonable to believe that the dissimilarity, denoted by β, between the two haplotypes H_1 and H_2 is big. Often, β is measured using the Hamming distance between H_1 and H_2 divided by the length m of H_1 and H_2, and is assumed to be large, say, $\beta \geq 0.2$. It is also often assumed that roughly half of the fragments in Γ are from each of the two

haplotypes H_1 and H_2.

In the experimental studies of algorithmic solutions to the singular haplotype reconstruction problem, we often need to generate synthetic data to evaluate the performance and accuracy of a given algorithm. One common practice (e.g., [13, 69, 93]) is as follows: First, choose two haplotypes H_1 and H_2 such that the dissimilarity between H_1 and H_2 is at least β. Second, make n_i copies of H_i, $i = 1, 2$. Third, for each copy $H = a_1 a_2 \cdots a_m$ of H_i, for each $j = 1, 2, \ldots, m$, with probability α_1, flip a_j to a'_j so that they are inconsistent. Also, independently, a_j has probability α_2 to be a hole -. A synthetic data set is then generated by setting parameters m, $n_1, n_2, \beta, \alpha_1$ and α_2. Usually, n_1 is roughly the same as n_2, and $\beta \approx 0.2$, $\alpha_1 \in [0.01, 0.05]$, and $\alpha_2 \in [0.1, 0.3]$.

Motivated by the above reality of the sequencing process and the common practice in experimental algorithm studies, we will present a probabilistic model for the singular haplotype reconstruction problem. First, we need to introduce some necessary notations and definitions.

Let $\Sigma_1 = \{A, B\}$ and $\Sigma_2 = \{A, B, \text{-}\}$. For a set C, $|C|$ denotes the number of elements in C. For a fragment (or a sequence) $S = a_1 a_2 \cdots a_m \in \Sigma_2^m$, $S[i]$ denotes the character a_i, and $S[i, j]$ denotes the substring $a_i \cdots a_j$ for $1 \leq i \leq j \leq m$. $|S|$ denotes the length m of S. When no confusion arises, we alternatively use the terms fragment and sequence.

Let $G = g_1 g_2 \cdots g_m \in \Sigma_1^m$ be a fixed sequence of m characters. For any sequence $S = a_1 \cdots a_m \in \Sigma_2^m$, S is called a $\mathcal{F}_{\alpha_1,\alpha_2}(m, G)$ sequence if for each a_i, with probability at most α_1, a_i is not equal to g_i and $a_i \neq$ -; and with probability at most α_2, $a_i =$ -.

For a sequence S, define *holes*(S) to be the number of holes in the sequence S. If A is a subset of $\{1, \cdots, m\}$ and S is a sequence of length m, *holes*$_A(S)$ is the number of $i \in A$ such that $S[i]$ is a hole.

For two sequences $S_1 = a_1 \cdots a_m$ and $S_2 = b_1 \cdots b_m$ of the same length m, for any $A \subseteq \{1, \cdots, m\}$, define

$$\textit{diff}(S_1, S_2) = \frac{|\{i \in \{1, 2, \cdots, m\} | a_i \neq \text{ - and } b_i \neq \text{ - and } a_i \neq b_i\}|}{m}$$

$$\textit{diff}_A(S_1, S_2) = \frac{|\{i \in A | a_i \neq \text{ - and } b_i \neq \text{ - and } a_i \neq b_i\}|}{|A|}.$$

For a set of sequences $\Gamma = \{S_1, S_2, \cdots, S_k\}$ of length m, define *vote*(Γ) to be the sequence H of the same length m such that $H[i]$ is the most frequent character among $S_1[i], S_2[i], \cdots, S_k[i]$ for $i = 1, 2, \cdots, m$.

We often use an $n \times m$ matrix M to represent a list of n fragments from Σ_2^m and call M an SNP fragment matrix. For $1 \leq i \leq n$, let $M[i]$ represent the i-th row of M, i.e., $M[i]$ is a fragment in Σ_2^m.

We now define our probabilistic model:

The Probabilistic Singular Haplotype Reconstruction Problem: Let β, α_1 and α_2 be small positive constants. Let $G_1, G_2 \in \Sigma_1^m$ be two haplotypes with $\textit{diff}(G_1, G_2) \geq \beta$. For any given $n \times m$ matrix M of SNP fragments such that n_i rows of M are $\mathcal{F}_{\alpha_1,\alpha_2}(m, G_i)$ sequences, $i = 1, 2$, $n_1 + n_2 = n$, reconstruct the two haplotypes G_1 and G_2, which are unknown to the users, from M as accurately as possible with high probability. We call β (resp., α_1, α_2) dissimilarity rate (resp., inconsistency error rate, incompleteness error rate).

6.3 Technical Lemmas

For probabilistic analysis we need the following two Chernoff bounds (see [74]), which can be derived from the work in [68].

Lemma 6.1. *[68] Let $X_1, \cdots, X_n$ be n independent random $0,1$ variables, where X_i takes 1 with probability at most p. Let $X = \sum_{i=1}^{n} X_i$. Then for any $1 \geq \epsilon > 0$, $\Pr(X > pn + \epsilon n) < e^{-\frac{1}{3}n\epsilon^2}$. Let $X_1, \cdots, X_n$ be n independent random $0,1$ variables, where X_i takes 1 with probability at least p. Let $X = \sum_{i=1}^{n} X_i$. Then for any $1 \geq \epsilon > 0$, $\Pr(X < pn - \epsilon n) < e^{-\frac{1}{2}n\epsilon^2}$.*

We shall prove several technical lemmas for algorithm analysis in the next three sections.

Lemma 6.2. *Let S be a $\mathcal{F}_{\alpha_1,\alpha_2}(m, G)$ sequence. Then, for any $0 < \epsilon \leq 1$, with probability at most $2e^{-\frac{\epsilon^2 m}{3}}$, $diff(G, S) > \alpha_1 + \epsilon$ or $holes(S) > (\alpha_2 + \epsilon)m$.*

Proof: Let X_k, $k = 1 \cdots, m$, be random variables such that $X_k = 1$ if $S[k] \neq G[k]$ and $S[k] \neq -$, or 0 otherwise. By the definition of the $\mathcal{F}_{\alpha_1,\alpha_2}(m, G)$ sequences, X_k are independent and $Pr(X_k = 1) \leq \alpha_1$. So, by Lemma 6.1, with probability at most $e^{-\frac{\epsilon^2 m}{3}}$, $X_1 + \cdots + X_m > (\alpha_1 + \epsilon)m$. Thus, we have $diff(G, S) > \alpha_1 + \epsilon$ with probability at most $e^{-\frac{\epsilon^2 m}{3}}$. Similarly, with probability at most $e^{-\frac{\epsilon^2 m}{3}}$, $holes(S) > (\alpha_2 + \epsilon)m$. ∎

Lemma 6.3. *Assume that A is a fixed subset of $\{1, 2, \cdots, m\}$. Let S be a $\mathcal{F}_{\alpha_1,\alpha_2}(m, G)$ sequence. Then, for any $0 < \epsilon \leq 1$, with probability at most $2e^{-\frac{\epsilon^2|A|}{3}}$, $diff_A(G, S) > \alpha_1 + \epsilon$ or* $\text{holes}_A(S) > (\alpha_2 + \epsilon)|A|$.

Proof: Let S' be the subsequence consisting of all the characters $S[i]$, $i \in A$, with the same order as in S. Similarly, let G' be the subsequence consisting of all the characters $G[i]$, $i \in A$, with the same order as in G. It is easy to see that $diff_A(S, G) = diff(S', G')$. The lemma follows from a similar proof for Lemma 7.8. ∎

Lemma 6.4. *Let N_i be a set of n_i many $\mathcal{F}_{\alpha_1,\alpha_2}(m, G_i)$ sequences, $i = 1, 2$. Let β and ϵ be two positive constants such that $2\alpha_1 + 2\alpha_2 + 2\epsilon < 1$, and $diff(G_1, G_2) \geq \beta$. Then, with probability at most $2(n_1 + n_2)e^{-\frac{\epsilon^2\beta m}{3}}$, $diff(S_i, S_j) \leq \beta(1 - 2\alpha_1 - 2\alpha_2 - 2\epsilon)$ for some $S_i \in N_i$ and some $S_j \in N_j$ with $i \neq j$.*

Proof: For each G_i, let A_i be the set of indexes $\{k \in \{1, 2, \cdots, m\} | G_i[k] \neq G_j[k]\}$, where $i \neq j$. Since $diff(G_i, G_j) \geq \beta$ and $|G_i| = |G_j| = m$, we have $|A_i| \geq \beta m$. For any $\mathcal{F}_{\alpha_1,\alpha_2}(m, G_i)$ sequence S, by Lemma 6.3, with probability at most $2e^{-\frac{\epsilon^2|A_i|}{3}} \leq 2e^{-\frac{\epsilon^2\beta m}{3}}$, $diff_{A_i}(S, G_i) > \alpha_1 + \epsilon$ or $holes_{A_i}(S) > (\alpha_2 + \epsilon)|A_i|$. Hence, with probability at most $2n_i e^{-\frac{\epsilon^2\beta m}{3}}$, $diff_{A_i}(S, G_i) > \alpha_1 + \epsilon$ or $holes_{A_i}(S) > (\alpha_2 + \epsilon)|A_i|$, for some $S \in N_i$. Therefore, with probability at most $2(n_1 + n_2)e^{-\frac{\epsilon^2\beta m}{3}}$, we have $diff_{A_i}(S, G_i) > \alpha_1 + \epsilon$ or $holes_{A_i}(S) > (\alpha_2 + \epsilon)|A_i|$, for some $S \in N_i$,

for some $i = 1$ or 2. In other words, with probability at least $1 - 2(n_1 + n_2)e^{-\frac{\epsilon^2 \beta m}{3}}$, we have $\mathit{diff}_{A_i}(S, G_i) \leq \alpha_1 + \epsilon$ and $\mathit{holes}_{A_i}(S) \leq (\alpha_2 + \epsilon)|A_i|$, for all $S \in N_i$ and for $i = 1$ and 2.

For any $\mathcal{F}_{\alpha_1,\alpha_2}(m, G_i)$ sequence S_i, $i = 1, 2$, if $\mathit{diff}_{A_i}(S_i, G_i) \leq \alpha_1 + \epsilon$ and $\mathit{holes}_{A_i}(S_i) \leq (\alpha_2 + \epsilon)|A_i|$, then $\mathit{diff}(S_1, S_2) \geq \mathit{diff}_{A_i}(S_1, S_2) \geq \beta(1 - 2\alpha_1 - 2\alpha_2 - 2\epsilon)$. Thus, with probability at least $1 - 2(n_1 + n_2)e^{-\frac{\epsilon^2 \beta m}{3}}$, we have $\mathit{diff}(S_1, S_2) \geq \beta(1 - 2\alpha_1 - 2\alpha_2 - 2\epsilon)$, for every $S_1 \in N_1$ and every $S_2 \in N_2$. In other words, with probability at most $2(n_1 + n_2)e^{-\frac{\epsilon^2 \beta m}{3}}$, we have $\mathit{diff}(S_1, S_2) < \beta(1 - 2\alpha_1 - 2\alpha_2 - 2\epsilon)$, for some $S_1 \in N_1$ and some $S_2 \in N_2$. ∎

Lemma 6.5. *Let α_1, α_2 and ϵ be three small positive constants that satisfy $0 < 2\alpha_1 + \alpha_2 - \epsilon < 1$. Assume that $N = \{S_1, \cdots, S_n\}$ is a set of $\mathcal{F}_{\alpha_1,\alpha_2}(m, G)$ sequences. Let $H = \mathit{vote}(N)$. Then, with probability at most $2m(e^{-\frac{\epsilon^2 n}{2}})$, $G \neq H$.*

Proof: Given any $1 \leq j \leq m$, for any $1 \leq i \leq n$, let X_i be random variables such that $X_i = 1$ if $S_i[j] \neq G[j]$ and $S_i[j] \neq -$, or 0 otherwise. By the definition of the $\mathcal{F}_{\alpha_1,\alpha_2}(m, G)$ sequences, X_i are independent and $Pr(X_i = 1) \leq \alpha_1$. So, by Lemma 6.1, with probability at most $e^{-\frac{\epsilon^2 n}{2}}$, $X_1 + \cdots + X_n < (\alpha - \epsilon)n$. That is, with probability at most $e^{-\frac{\epsilon^2 n}{2}}$, there are fewer than $(\alpha_1 - \epsilon)n$ characters $S_i[j]$ such that $S_i[j] \neq G[j]$ and $S_i[j] \neq$ -. Similarly, with probability at most $e^{-\frac{\epsilon^2 n}{2}}$, there are fewer than $(\alpha_2 - \epsilon)n$ characters $S_i[j]$ such that $S_i[j] =$ -. Thus, with probability at most $2me^{-\frac{\epsilon^2 n}{2}}$, there are fewer than $(\alpha_1 + \alpha_2 - 2\epsilon)n$ characters $S_i[j]$ such that $S_i[j] \neq G[j]$ for some $1 \leq j \leq m$. This implies that, with probability at least $1 - 2me^{-\frac{\epsilon^2 n}{2}}$, there are more than $(1 - \alpha_1 - \alpha_2 + 2\epsilon)n$ characters $S_i[j]$ such that $S_i[j] = G[j]$ for any $1 \leq j \leq m$. Since $0 < 2\alpha_1 + \alpha_2 - \epsilon < 1$ by the assumption of the theorem, we have $(\alpha_1 + \epsilon)n < (1 - \alpha_1 - \alpha_2 + 2\epsilon)n$. This further implies that with probability at least $1 - 2me^{-\frac{\epsilon^2 n}{2}}$, $\mathit{vote}(N)[j] = G[j]$ for any $1 \leq j \leq m$, i.e., $\mathit{vote}(N) = G$. ∎

6.4 When the Inconsistency Error Parameter Is Known

Theorem 6.6. *Assume that $\alpha_1, \alpha_2, \beta$, and $\epsilon > 0$ are small positive constants that satisfy $4(\alpha_1 + \epsilon) < \beta$ and $0 < 2\alpha_1 + \alpha_2 - \epsilon < 1$. Let $G_1, G_2 \in \Sigma_1^m$ be the two unknown haplotypes such that $\mathit{diff}(G_1, G_2) \geq \beta$. Let M be any given $n \times m$ matrix of SNP fragments such that M has n_i fragments that are $\mathcal{F}_{\alpha_1,\alpha_2}(m, G_i)$ sequences, $i = 1, 2$, and $n_1 + n_2 = n$. There exists an $O(nm)$ time algorithm that can find two haplotypes H_1 and H_2 with probability at least $1 - 2ne^{-\frac{\epsilon^2 m}{3}} - 2me^{-\frac{\epsilon^2 n_1}{2}} - 2me^{-\frac{\epsilon^2 n_2}{2}}$ such that either $H_1 = G_1$ and $H_2 = G_2$, or $H_1 = G_2$ and $H_2 = G_1$.*

Proof: The algorithm, denoted as SHR-One, is described as follows.

Algorithm SHR-One

Input: M, an $n \times m$ matrix of SNP fragments.
Parameters α_1 and ϵ.

Output: Two haplotypes H_1 and H_2.

Set $\Gamma_1 = \Gamma_2 = \emptyset$.

Randomly select a fragment $r = M[j]$ for some $1 \leq j \leq n$.

For every fragment r' from M do
If ($\mathit{diff}(r, r') \leq 2(\alpha_1 + \epsilon)$) then put r' into Γ_1
Let $\Gamma_2 = M - \Gamma_1$.
Let $H_1 = \mathit{vote}(\Gamma_1)$ and $H_2 = \mathit{vote}(\Gamma_2)$.
return H_1 and H_2.

End of Algorithm

Claim 1. With probability at most $ne^{-\frac{\epsilon^2 m}{3}}$, we have either $\mathit{diff}(f, G_1) > \alpha_1 + \epsilon$ for some $\mathcal{F}_{\alpha_1,\alpha_2}(m, G_1)$ sequence f in M, or $\mathit{diff}(g, G_2) > \alpha_1 + \epsilon$ for some $\mathcal{F}_{\alpha_1,\alpha_2}(m, G_2)$ sequence g in M.

By Lemma 7.8, for any fragment $f = M[k]$ such that f is a $\mathcal{F}_{\alpha_1,\alpha_2}(m, G_1)$ sequence, with probability at most $e^{-\frac{\epsilon^2 m}{3}}$ we have $\mathit{diff}(f, G_1) > \alpha_1 + \epsilon$. Since there are n_1 many $\mathcal{F}_{\alpha_1,\alpha_2}(m, G_1)$ sequences in M, with probability at most $n_1 e^{-\frac{\epsilon^2 m}{3}}$, we have $\mathit{diff}(f, G_1) > \alpha_1 + \epsilon$ for some $\mathcal{F}_{\alpha_1,\alpha_2}(m, G_1)$ sequence f in M. Similarly, with probability at most $n_2 e^{-\frac{\epsilon^2 m}{3}}$, we have $\mathit{diff}(g, G_2) > \alpha_1 + \epsilon$ for some $\mathcal{F}_{\alpha_1,\alpha_2}(m, G_2)$ sequence g in M. Combining the above completes the proof for Claim 1.

Claim 2. Let M_i be the set of all the $\mathcal{F}_{\alpha_1,\alpha_2}(m, G_i)$ sequences in M, $i = 1, 2$. With probability at least $1 - ne^{-\frac{\epsilon^2 m}{3}}$, Γ_1 and Γ_2 is a permutation of M_1 and M_2.

By the assumption of the theorem, the fragment r of M is either a $\mathcal{F}_{\alpha_1,\alpha_2}(m, G_1)$ sequence or a $\mathcal{F}_{\alpha_1,\alpha_2}(m, G_2)$ sequence. We assume that the former is true. By Claim 1, with probability at least $1 - ne^{-\frac{\epsilon^2 m}{3}}$, we have $\mathit{diff}(f, G_1) \leq \alpha_1 + \epsilon$ for all $\mathcal{F}_{\alpha_1,\alpha_2}(m, G_1)$ sequences f in M, and $\mathit{diff}(g, G_2) \leq \alpha_1 + \epsilon$ for all $\mathcal{F}_{\alpha_1,\alpha_2}(m, G_2)$ sequences g in M. Hence, for any fragment r' in M, if r' is a $\mathcal{F}_{\alpha_1,\alpha_2}(m, G_1)$ sequence, then with probability at least $1 - ne^{-\frac{\epsilon^2 m}{3}}$, we have $\mathit{diff}(r, r') \leq \mathit{diff}(r, G_1) + \mathit{diff}(r', G_1) \leq 2(\alpha_1 + \epsilon)$. This means that, with probability at least $1 - ne^{-\frac{\epsilon^2 m}{3}}$, all $\mathcal{F}_{\alpha_1,\alpha_2}(m, G_1)$ sequences in M will be included in Γ_1. Now, consider the case that r' is a $\mathcal{F}_{\alpha_1,\alpha_2}(m, G_2)$ sequence in M. Since $\mathit{diff}(G_1, G_2) \leq \mathit{diff}(G_1, r) + \mathit{diff}(r, G_2) \leq \mathit{diff}(G_1, r) + \mathit{diff}(r, r') + \mathit{diff}(r', G_2)$, we have $\mathit{diff}(r, r') \geq \mathit{diff}(G_1, G_2) - \mathit{diff}(G_1, r) - \mathit{diff}(G_2, r')$. By the given condition of $\mathit{diff}(G_1, G_2) \geq \beta$ and $4(\alpha_1 + \epsilon) < \beta$, with probability at least $1 - ne^{-\frac{\epsilon^2 m}{3}}$, we have $\mathit{diff}(r, r') \geq \beta - \mathit{diff}(G_1, r) - \mathit{diff}(G_2, r') \geq \beta - 2(\alpha_1 + \epsilon) > 2(\alpha_1 + \epsilon)$, i.e., r' will not be added to Γ_1. Therefore, with probability at least $1 - ne^{-\frac{\epsilon^2 m}{3}}$, $\Gamma_1 = M_1$ and $\Gamma_2 = M - \Gamma_1 = M_2$. Similarly, if r is a $\mathcal{F}_{\alpha_1,\alpha_2}(m, G_2)$ sequence, with probability at least $1 - ne^{-\frac{\epsilon^2 m}{3}}$, $\Gamma_1 = M_2$ and $\Gamma_2 = M - \Gamma_1 = M_1$. This completes the proof of Claim 2.

Suppose that Γ_1 and Γ_2 is a permutation of M_1 and M_2. Say, without loss of generality, $\Gamma_1 = M_1$ and $\Gamma_2 = M_2$. By Lemma 6.5, with probability at most $2me^{-\frac{\epsilon^2 n_1}{2}} + 2me^{-\frac{\epsilon^2 n_2}{2}}$, $\mathit{vote}(\Gamma_1) \neq G_1$ or $\mathit{vote}(\Gamma_2) \neq G_2$. Hence, by Claim 2, with probability at most $2ne^{-\frac{\epsilon^2 m}{3}} + 2me^{-\frac{\epsilon^2 n_1}{2}} + 2me^{-\frac{\epsilon^2 n_2}{2}}$, $\mathit{vote}(\Gamma_1) \neq G_1$ or $\mathit{vote}(\Gamma_2) \neq G_2$.

Concerning the computational time of the algorithm, we need to compute the difference between the selected fragment r and each of the remaining $n-1$ fragments in the matrix M. Finding the difference between r and r' takes $O(m)$ steps. So, the total computational time is $O(nm)$, which is linear in the size of the input matrix M. ∎

6.5 When Parameters Are Not Known

In this section, we consider the case that the parameters α_1, α_2 and β are unknown. However, we assume the existence of those parameters for the input matrix M of SNP fragments. We will show that in this case we can still reconstruct the two unknown haplotypes from M with high probability.

Theorem 6.7. *Assume that $\alpha_1, \alpha_2, \beta$, and $\epsilon > 0$ are small positive constants that satisfy $2\alpha_1+2\alpha_2+2\epsilon < 1$, $0 < 2\alpha_1+\alpha_2-\epsilon < 1$, and $\beta(1-2\alpha_1-2\alpha_2-2\epsilon) > 2(\alpha_1+\epsilon)$. Let $G_1, G_2 \in \Sigma_1^m$ be the two unknown haplotypes such that diff$(G_1, G_2) \geq \beta$. Let M be any given $n \times m$ matrix of SNP fragments such that M has n_i fragments that are $\mathcal{F}_{\alpha_1,\alpha_2}(m, G_i)$ sequences, $i = 1, 2$, and $n_1 + n_2 = n$. Then, there exists an $O(umn)$ time algorithm that can find two haplotypes H_1 and H_2 with probability at least $1-(1-\gamma)^u - 4ne^{-\frac{\epsilon^2\beta m}{3}} - 2me^{-\frac{\epsilon^2 n_1}{2}} - 2me^{-\frac{\epsilon^2 n_2}{2}}$ such that H_1, H_2 is a permutation of G_1, G_2, where $\gamma = \frac{n_1 n_2}{n(n-1)}$ and u is an integer parameter.*

Proof: The algorithm, denoted as SHR-Two, is described as follows.

Algorithm SHR-Two

Input: M, an $n \times m$ matrix M of SNP fragments.
u, a parameter to control the loop.
Output: two haplotypes H_1 and H_2.
Let $d_{\min} = \infty$ and $\mathcal{M} = \emptyset$.
For ($k = 1$ to u) do { //**the k-loop**
Let $M_1 = M_2 = \emptyset$ and $d_1 = d_2 = 0$.
Randomly select two fragments $r_1 = M[i_1], r_2 = M[i_2]$ from M
For every fragment r' from M do {
If ($diff(r_i, r') = min\{diff(r_1, r'), diff(r_2, r')\}$ for $i = 1$ or 2, then put r' into M_i.
}
Let $d_i = \max\{diff(r_i, r') | r' \in M_i\}$ for $i = 1, 2$.
Let $d = \max\{d_1, d_2\}$.
If ($d < d_{\min}$) then let $\mathcal{M} = \{M_1, M_2\}$ and $d_{min} = d$.
}
return $H_1 = vote(M_1)$ and $H_2 = vote(M_2)$.

End of Algorithm

Claim 3. With probability at most $(1-\gamma)^u$, r_1, r_2 is not a permutation of a $\mathcal{F}_{\alpha,\beta}(m, G_1)$ sequence and a $\mathcal{F}_{\alpha,\beta}(m, G_2)$ sequence in all of the k-loop iterations.

For randomly selected fragments r_1 and r_2, with probability γ, r_1, r_2 is a permutation of a $\mathcal{F}_{\alpha,\beta}(m, G_1)$ sequence and a $\mathcal{F}_{\alpha,\beta}(m, G_2)$ sequence in M. When the k-loop is repeated u times, with probability at most $(1-\gamma)^u$, r_1, r_2 is not a permutation of a $\mathcal{F}_{\alpha,\beta}(m, G_1)$ sequence and a $\mathcal{F}_{\alpha,\beta}(m, G_2)$ sequence in all of the u loop iterations. Thus, Claim 3 is true.

Let N_i be the set of the n_i fragments in M that are $\mathcal{F}_{\alpha_1,\alpha_2}(m, G_i)$ sequences, $i = 1, 2$.

Claim 4. With probability at most $4ne^{-\frac{\epsilon^2\beta m}{3}}$, $diff(G_i, S) > \alpha_1 + \epsilon$ or $holes(S) > (\alpha_2 + \epsilon)m$ for some S from N_i for some $i = 1$ or 2; or $diff(S_1, S_2) \leq \beta(1 - 2\alpha_1 - 2\alpha_2 - 2\epsilon)$ for some $S_1 \in N_1$ and some $S_2 \in N_2$.

By Lemma 7.8, for every fragment S from N_i, with probability at most $2e^{-\frac{\epsilon^2 m}{3}}$, $\mathit{diff}(G_i, S) > \alpha_1 + \epsilon$ or S has more than $(\alpha_2 + \epsilon)m$ holes. Thus, with probability at most $2ne^{-\frac{\epsilon^2 m}{3}}$, $\mathit{diff}(G_i, S) > \alpha_1 + \epsilon$ or $\mathit{holes}(S) > (\alpha_2 + \epsilon)m$ for some S from N_i for some $i = 1$ or 2.

By Lemma 6.4, with probability at most $2ne^{-\frac{\epsilon^2 \beta m}{3}}$, $\mathit{diff}(S_1, S_2) \leq \beta(1-2\alpha_1-2\alpha_2-2\epsilon)$ for some $S_1 \in N_1$ and some $S_2 \in N_2$.

The above analysis completes the proof for Claim 4.

Claim 5. Let $H_1 = \mathit{vote}(M_1)$ and $H_2 = \mathit{vote}(M_2)$ be the two haplotypes returned by the algorithm. With probability at most $(1-\gamma)^u + 4ne^{-\frac{\epsilon^2 \beta m}{3}}$, M_1, M_2 is not a permutation of N_1, N_2.

We assume that (1) $\mathit{diff}(S_1, S_2) > \beta(1 - 2\alpha_1 - 2\alpha_2 - 2\epsilon)$ for every S_1 from N_1 and every S_2 from N_2; and (2) $\mathit{diff}(G_i, S) \leq \alpha_1 + \epsilon$ and $\mathit{holes}(S) \leq (\alpha_2 + \epsilon)m$ for all $S \in N_i$ for $i = 1, 2$. We consider possible choices of the two random fragments r_1 and r_2 in the following.

At any iteration of the k-loop, if $r_1 \in N_1$ and $r_2 \in N_2$, then by (2) we have $\mathit{diff}(r_1, r') \leq \mathit{diff}(r_1, G_1) + \mathit{diff}(r', G_1) \leq 2(\alpha_1 + \epsilon)$ for any $r' \in N_1$; and $\mathit{diff}(r_2, r') \leq \mathit{diff}(r_2, G_2) + \mathit{diff}(r', G_2) \leq 2(\alpha_1 + \epsilon)$ for any $r' \in N_2$. By (1) and the given condition of the theorem, we have, $\mathit{diff}(r_1, r') > \beta(1 - 2\alpha_1 - 2\alpha_2 - 2\epsilon) > 2(\alpha_1 + \epsilon)$ for any $r' \in N_2$; and $\mathit{diff}(r_2, r') > \beta(1 - 2\alpha_1 - 2\alpha_2 - 2\epsilon) > 2(\alpha_1 + \epsilon)$ for any $r' \in N_1$. This implies that at this loop iteration we have $M_1 = N_1, M_2 = N_2$ and $d \leq 2(\alpha_1 + \epsilon)$. Similarly, if at this iteration $r_1 \in N_2$ and $r_2 \in N_1$, then $M_1 = N_2, M_2 = N_1$ and $d \leq 2(\alpha_1 + \epsilon)$.

If $r_1, r_2 \in N_1$ at some iteration of the k-loop, then for any $r' \in N_2$, either $r' \in M_1$ or $r' \in M_2$. In either case, by (1) of our assumption and the given condition of the theorem, we have $d \geq \beta(1 - 2\alpha_1 - 2\alpha_2 - 2\epsilon) > 2(\alpha_1 + \epsilon)$ at this iteration. Similarly, if $r_1, r_2 \in N_2$ at some iteration of the k-loop, then we also have $d > 2(\alpha_1 + \epsilon)$ at this iteration.

It follows from the above analysis that, under the assumption of (1) and (2), once we have $r_1 \in N_1$ and $r_2 \in N_2$ or $r_1 \in N_2$ and $r_2 \in N_1$ at some iteration of the k-loop, then M_1, M_2 is a permutation of N_1, N_2 at the end of this iteration. Furthermore, if M_1 and M_2 are replaced by M_1' and M_2' after this iteration, then M_1', M_2' must also be a permutation of N_1, N_2. By Claims 3 and 4, with probability at most $(1-\gamma)^u + 4ne^{-\frac{\epsilon^2 \beta m}{3}}$, the assumption of (1) and (2) is not true, or $r_1 \in N_1$ and $r_2 \in N_2$ (or $r_1 \in N_2$ and $r_2 \in N_1$) is not true at all the iterations of the k-loop. Hence, with probability at most $(1-\gamma)^u + 4ne^{-\frac{\epsilon^2 \beta m}{3}}$, the final list of M_1 and M_2 returned by the algorithm is not a permutation of N_1, N_2, so the claim is proved.

For M_1 and M_2 returned by the algorithm, we assume without loss of generality $M_i = N_i$, $i = 1, 2$. By Lemma 6.5 and the given condition of the theorem, with probability at most $2me^{-\frac{\epsilon^2 n_1}{2}} + 2me^{-\frac{\epsilon^2 n_2}{2}}$, we have $H_1 = \mathit{vote}(M_1) \neq G_1$ or $H_2 = \mathit{vote}(M_2) \neq G_2$. Thus, by Claim 6, with probability at most $(1-\gamma)^u + 4ne^{-\frac{\epsilon^2 \beta m}{3}} + 4me^{-\frac{\epsilon^2 n}{2}}$, we have $H_1 \neq G_1$ or $H_2 \neq G_2$.

It is easy to see that the time complexity of the algorithm is $O(umn)$, which is linear in the size of M for a constant u. ∎

6.6 Tuning the Dissimilarity Measure

In this section, we consider a different dissimilarity measure in algorithm SHR-TWO to improve its ability of error tolerance. We use the sum of the differences between r_i and every fragment $r' \in M_i$, $i = 1, 2$, to measure the dissimilarity of the fragments in M_i with r_i. The new algorithm SHR-Three is given in the following. We will present experimental results in Section 6.7 to show that algorithm SHR-Three is reliable and robust in dealing with possible outliers in the data sets.

Algorithm SHR-Three

Input: M, an $n \times m$ matrix of SNP fragments.
u, a parameter to control the loop.

Output: two haplotypes H_1 and H_2.

Let $d_{\min} = \infty$ and $\mathcal{M} = \emptyset$.

For ($k = 1$ to u) do { //**the** k**-loop**

Let $M_1 = M_2 = \emptyset$ and $d_1 = d_2 = 0$.

Randomly select two fragments $r_1 = M[i_1], r_2 = M[i_2]$ from M

For every fragment r' from M do {

If ($\mathit{diff}(r_i, r') = min\{\mathit{diff}(r_1, r'), \mathit{diff}(r_2, r')\}$ for $i = 1$ or 2, then put r' into M_i.

}

Let $d_i = \sum_{r' \in M_i} \mathit{diff}(r_i, r')$ for $i = 1, 2$.

Let $d = \max\{d_1, d_2\}$.

If ($d < d_{\min}$) then let $\mathcal{M} = \{M_1, M_2\}$ and $d_{min} = d$.

}

return $H_1 = \mathit{vote}(M_1)$ and $H_2 = \mathit{vote}(M_2)$.

End of Algorithm

Theorem 6.8. *Assume that $\alpha_1, \alpha_2, \beta$, and $\epsilon > 0$ are small positive constants that satisfy $2\alpha_1 + 2\alpha_2 + 2\epsilon < 1$, $0 < 2\alpha_1 + \alpha_2 - \epsilon < 1$, $\eta > \frac{2(\alpha_1+\epsilon)}{\beta(1-2\alpha_1-2\alpha_2-2\epsilon)}$ with $\eta = \frac{\min(n_1,n_2)}{2n}$, and $\beta(1 - 2\alpha_1 - 2\alpha_2 - 2\epsilon) > 2(\alpha_1 + \epsilon)$. Let $G_1, G_2 \in \Sigma_1^m$ be the two unknown haplotypes such that $\mathit{diff}(G_1, G_2) \geq \beta$. Let M be any given $n \times m$ matrix of SNP fragments such that M has n_i fragments that are $\mathcal{F}_{\alpha_1,\alpha_2}(m, G_i)$ sequences, $i = 1, 2$, and $n_1 + n_2 = n$. Then, there exists an $O(umn)$ time algorithm that can find two haplotypes H_1 and H_2 with probability at least $1 - (1 - \gamma)^u - 4ne^{-\frac{\epsilon^2\beta m}{3}} - 2me^{-\frac{\epsilon^2 n_1}{2}} - 2me^{-\frac{\epsilon^2 n_2}{2}}$ such that H_1, H_2 is a permutation of G_1, G_2, where $\gamma = \frac{n_1 n_2}{n(n-1)}$ and u is an integer parameter.*

Proof: Let N_i be the set of the n_i many $\mathcal{F}_{\alpha_1,\alpha_2}(m, G_i)$ sequences in M, for $i = 1, 2$.

We first notice that both Claims 3 and 4 in the proof of Theorem 6.7 hold here following the same analysis. However, we need to prove the following claim with different analysis:

Claim 6. Let $H_1 = \mathit{vote}(M_1)$ and $H_2 = \mathit{vote}(M_2)$ be the two haplotypes returned by the algorithm. With probability at most $(1 - \gamma)^u + 4ne^{-\frac{\epsilon^2\beta m}{3}}$, M_1, M_2 is not a permutation of N_1, N_2.

We assume that (1) $diff(S_1, S_2) > \beta(1 - 2\alpha_1 - 2\alpha_2 - 2\epsilon)$ for every S_1 from N_1 and every S_2 from N_2; and (2) $diff(G_i, S) \leq \alpha_1 + \epsilon$ and $holes(S) \leq (\alpha_2 + \epsilon)m$ for each S from N_i $(i = 1, 2)$.

We shall consider the two cases:

Case 1. At some iteration of the k-loop, both r_1 and r_2 are selected from the same N_i for $i = 1$ or 2. For each $r' \in N_j$, $j \neq i$, by assumption (1) we have both $diff(r_i, r') > \beta(1 - 2\alpha_1 - 2\alpha_2 - 2\epsilon)$, $i = 1, 2$. Notice that r' must be either in M_1 or M_2. So, at least half of the fragments in N_j will be either in M_1 or M_2. Therefore, at this iteration, we have $d \geq \frac{1}{2}n_j\beta(1-2\alpha_1-2\alpha_2-2\epsilon) \geq \eta n\beta(1-2\alpha_1-2\alpha_2-2\epsilon) = \eta\beta(1-2\alpha_1-2\alpha_2-2\epsilon)n > 2(\alpha_1+\epsilon)n$.

Case 2. At some iteration of the k-loop, r_1 and r_2 are selected from different N_i for $i = 1$ and 2. Without loss of generality, $r_i \in N_i$, $i = 1, 2$. For each $r' \in N_1$, by assumption (2) we have $diff(r_1, r') \leq 2(\alpha_1 + \epsilon)$; by assumption (1) and the given condition of the theorem we have $diff(r_2, r') > \beta(1 - 2\alpha_1 - 2\alpha_2 - 2\epsilon) > 2(\alpha_1 + \epsilon)$. Similarly, for each $r' \in N_2$, $diff(r_2, r') \leq 2(\alpha_1 + \epsilon)$, and $diff(r_1, r') > 2(\alpha_1 + \epsilon)$. Therefore, at this iteration, we have $M_1 = N_1$ and $M_2 = N_2$, and $d \leq 2(\alpha_1 + \epsilon)n_1 + 2(\alpha_1 + \epsilon)n_2 = 2(\alpha_1 + \epsilon)n$.

The two cases implies that under the assumption of (1) and (2), if at any iteration of the k-loop, we have $r_1 \in N_1$ and $r_2 \in N_2$, or $r_1 \in N_2$ and $r_2 \in N_1$, then the final list of the two sets M_1, M_2 is a permutation of N_1, N_2. Hence, Claim 6 follows from Claims 3 and 4 in the proof of Theorem 6.7, which are true here as we mentioned earlier.

Now, we assume that the final list of the two sets M_1, M_2 is a permutation of N_1, N_2. By Lemma 6.5, with probability at most $2m(e^{-\frac{\epsilon^2 n_1}{2}}) + 2m(e^{-\frac{\epsilon^2 n_2}{2}})$, $H_1 = vote(M_1)$, $H_2 = vote(M_2)$ is not a permutation of G_1, G_2. This, together with Claim 6, completes the probabilistic claim of the theorem.

It is easy to see that the time complexity of the algorithm is $O(umn)$, which is linear in the size of M. ∎

6.7 Experimental Results

We design a MATLAB program to test both the accuracy and the speed of algorithm SHR-Three. Due to the difficulty of getting real data from the public domain, our experiment is based on the literature [9,93] which is considered as common practice. A random matrix of SNP fragments is created as follows: (1) Haplotype 1 is generated at random with length m $(m \in \{50, 100, 150\})$. (2) Haplotype 2 is generated by copying all the bits from haplotype 1 and flipping each bit with probability β $(\beta \in \{0.1, 0.2, 0.3\})$. This simulates the dissimilarity rate β between two haplotypes. (3) Each haplotype is copied $\frac{n}{2}$ times so that the matrix has m columns and $n(n \in \{10, 20, 30\})$ rows. (4) Set each bit in the matrix to - with probability α_2 $(\alpha_2 \in \{0.1, 0.2, 0.3\})$. This simulates the incompleteness error rate α_2 in the matrix. (5) Flip each non-empty bit with probability $\alpha_1(\alpha_1 \in \{0.01, 0.02, ..., 0.1\})$. This is the simulation of the inconsistency error rate of α_1.

Figure 6.1 shows the performance of algorithm SHR-Three with different parameter settings in accordance with those in [9]. See Tables A.5 to A.8 in the appendix section A.4 for result details. The typical parameters used in [9] are $m = 100, n = 20, \beta = 0.2, \alpha_2 = 0.2$ and $0.01 \leq \alpha_1 \leq 0.05$. These are considered to be close to the real situations. In Figure 6.1, the results are the average time and the reconstruction rate of the 1000 executions of

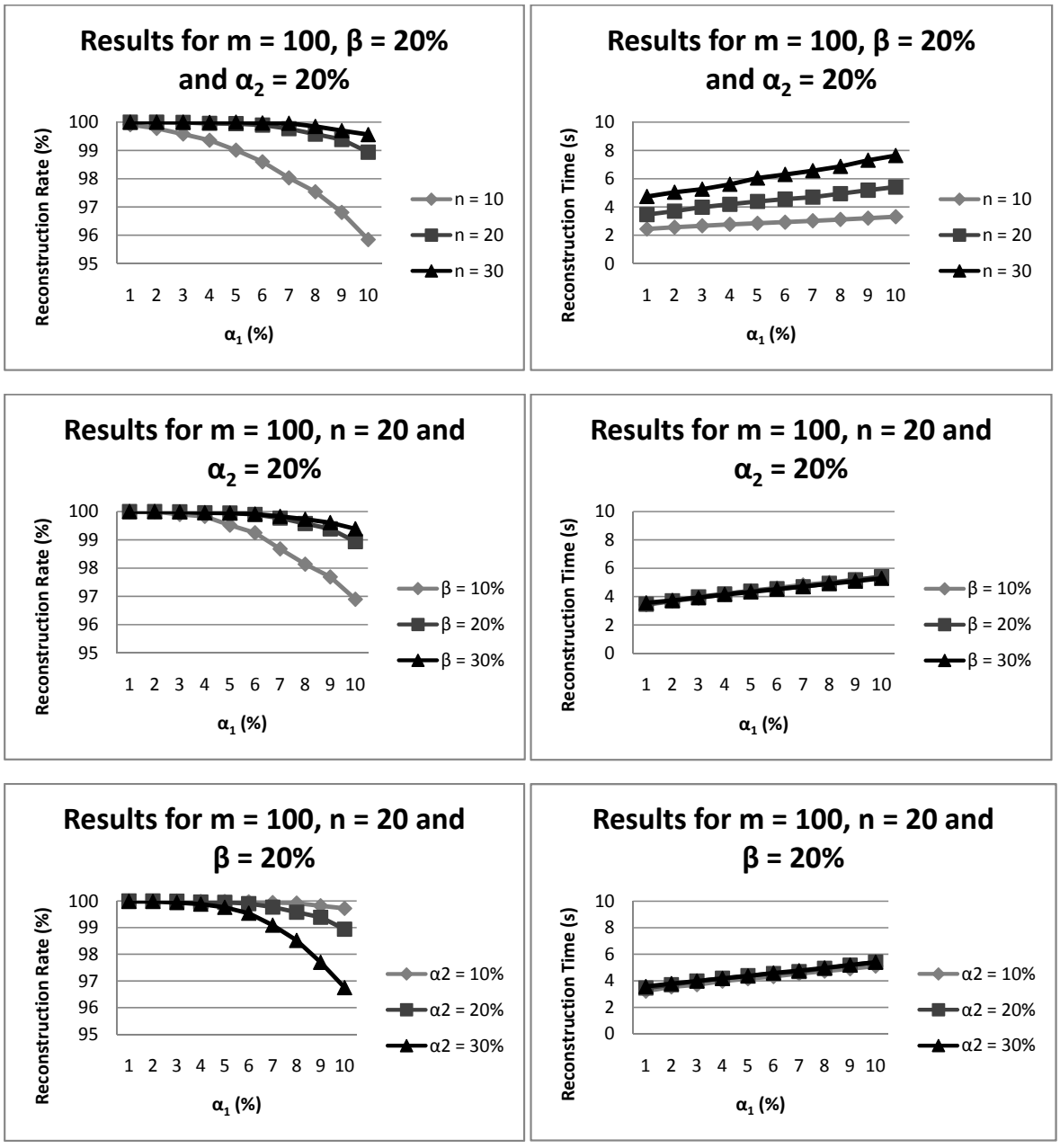


Figure 6.1: Performance of algorithm SHR-Three

algorithm SHR-Three. A new random matrix is used for each execution. The reconstruction rate is defined as the ratio of the total number of correctly reconstructed bits to the total number of bits in two haplotypes. The computing environment is a PC machine with a typical configuration of 1.6GHz AMD Turion 64X2 CPUs and 1GB memory.

The software of our algorithms is available for public access and for real-time on-line demonstration at `http://fpsa.cs.uno.edu/HapRec/HapRec.html`. Thank Liqiang Wang for implementing the programs in Java and setting up this web site.

Summary We have developed three linear time random algorithms for the singular haplotype reconstruction problem with provable high probability in reconstruction rate. Our experimental results conform with the theoretical efficiency and accuracy of the algorithms.

It should be pointed out that our work can be extended to reconstruct multiple haplotypes from a set of fragments. Our approach also opens the door to develop probabilistic methods for other variants of the haplotyping problem involving both inconsistency and incompleteness errors.

Chapter 7

Non-Breaking Similarity of Genomes with Gene Repetitions

7.1 Introduction

Until a few years ago, in genome rearrangement research, it is always assumed that every gene appears in a genome exactly once. Under this assumption, the genome rearrangement problem is in essence the problem of comparing and sorting signed/unsigned permutations [51, 77]. In the case of breakpoint distance, given two perfect genomes (every gene appears exactly once, i.e., there is no gene repetition) it is easy to compute their breakpoint distance in linear time.

However, perfect genomes are hard to obtain and so far they can only be obtained in several small virus genomes. For example, perfect genomes do not occur on eukaryotic genomes where paralogous genes are common [76, 84]. On the one hand, it is important in practice to compute genomic distances, e.g., Hannenhalli and Pevzner's method [51], when no gene duplication arises; on the other hand, one might have to handle this gene duplication problem as well. In 1999, Sankoff proposed a way to select, from the duplicated copies of genes, the common ancestor gene such that the distance between the reduced genomes (*exemplar genomes*) is minimized [84]. A general branch-and-bound algorithm was also implemented in [84]. Recently, Nguyen, Tay and Zhang proposed to use a divide-and-conquer method to compute the exemplar breakpoint distance empirically [76].

For the theoretical part of research, it was shown that computing the signed reversals and breakpoint distances between exemplar genomes are both NP-complete [15]. Two years ago, Blin and Rizzi further proved that computing the conserved interval distance between exemplar genomes is NP-complete [14]; moreover, it is NP-complete to compute the minimum conserved interval matching (i.e., without deleting the duplicated copies of genes). In [22, 25] it was shown that the exemplar genomic distance problem does not admit any approximation (regardless of the approximation factor) unless P=NP, as long as G=H implies that d(G,H)=0. This implies that for the exemplar breakpoint distance problem, there does not exist any approximation.

In [17] three new kinds of genomic similarities were considered. These similarity measures do not satisfy the condition that G=H implies that d(G,H)=0. Among them, the common interval distance problem seems to be the most interesting one. When gene duplications are allowed, Chauve *et al.* proved that the problem is NP-complete.

In this chapter[6] , we define a new similarity measure called *non-breaking similarity*. Intuitively, this is the complement of the traditional breakpoint distance measure. Compared with the problem of computing exemplar breakpoint distance, which is a minimization problem, we want to maximize the number of non-breaking points for the exemplar non-breaking similarity problem. Unfortunately, we show that Independent Set can be reduced to ENbS, and this reduction implies that ENbS is W[1]-complete (and ENbS does not have a factor-n^{ϵ} approximation). This reduction works even when one of the two genomes is given exemplar.

While the W[1]-completeness [32] and the recent lower bound results [20] implies that if k is the optimal solution value, unless an unlikely collapse occurs in parameterized complexity theory, ENbS is not solvable in time $f(k)n^{o(k)}$, for any function f. We show that for several practically interesting cases of the problem, there are polynomial time algorithms. This is done by parameterize some quantities in the input genomes, followed by some traditional algorithmic techniques.

7.2 Preliminaries

In the genome comparison and rearrangement problem, we are given a set of genomes, each of which is a signed/unsigned sequence of genes. In general a genome could contain a set of such sequences. The genomes we focus in this chapter are typically called *singletons*. The order of the genes corresponds to the position of them on the linear chromosome and the signs correspond to which of the two DNA strands the genes are located. While most of the past research are under the assumption that each gene occurs in a genome once, this assumption is problematic in reality for eukaryotic genomes or the likes where duplications of genes exist [84]. Sankoff proposed a method to select an *exemplar genome*, by deleting redundant copies of a gene, such that in an exemplar genome any gene appears exactly once. The resulting exemplar genomes have the property that certain genomic distance between them is minimized [84].

The following definitions are very much following those in [15, 25]. Given n *gene families* (alphabet) $\mathcal{F}$, a genome $\mathcal{G}$ is a sequence of elements of $\mathcal{F}$. (Throughout this chapter, we will consider unsigned genomes, though our results can be applied to signed genomes as well.) In general, we allow the repetition of a gene family in any genome. Each occurrence of a gene family is called a *gene*, though we will not try to distinguish a gene and a gene family if the context is clear.

The number of a gene g appearing in a genome $\mathcal{G}$ is called the occurrence of g in $\mathcal{G}$, written as $occ(g, \mathcal{G})$. A genome $\mathcal{G}$ is called *r-repetitive*, if all the genes from the same gene family occur at most r times in $\mathcal{G}$. For example, if $\mathcal{G} = abcbaa$, $\text{occ}(b, \mathcal{G}) = 2$ and $\mathcal{G}$ is a 3-repetitive genome.

For a genome $\mathcal{G}$, alphabet($\mathcal{G}$) is the set of all the characters (genes) that appear at least once in $\mathcal{G}$. A genome G is an exemplar genome of $\mathcal{G}$ if alphabet(G) = alphabet($\mathcal{G}$), each gene in alphabet($\mathcal{G}$) appears exactly once in G; i.e., G is derived from $\mathcal{G}$ by deleting all the redundant genes (characters) in $\mathcal{G}$. For example, let $\mathcal{G} = bcaadage$ there are two exemplar genomes: *bcadge* and *bcdage*.

[6]Published work [24] used with permission of Springer Science and Business Media.

For two exemplar genomes G and H such that alphabet(G) = alphabet(H) and $|\text{alphabet}(G)| = |\text{alphabet}(H)| = n$, a breakpoint in G is a two-gene substring $g_i g_{i+1}$ such that $g_i g_{i+1}$ is not a substring in H. The number of breakpoints in G (symmetrically in H) is called the *breakpoint distance*, denoted as $\text{bd}(G, H)$. For two genomes $\mathcal{G}$ and $\mathcal{H}$, their *exemplar breakpoint distance* $\text{ebd}(\mathcal{G}, \mathcal{H})$ is the minimum $\text{bd}(G, H)$, where G and H are exemplar genomes derived from $\mathcal{G}$ and $\mathcal{H}$.

For two exemplar genomes G and H such that alphabet(G) = alphabet(H) $|\text{alphabet}(G)| = |\text{alphabet}(H)| = n$, a *non-breaking point* is a common two-gene substring $g_i g_{i+1}$ that it appears in both G and H. The number of non-breaking points between G and H is also called the *non-breaking similarity* between G and H, denoted as $\text{nbs}(G, H)$. Clearly, we have $\text{nbs}(G, H) = n - 1 - \text{bd}(G, H)$. For two genomes $\mathcal{G}$ and $\mathcal{H}$, their *exemplar non-breaking similarity* $\text{enbs}(\mathcal{G}, \mathcal{H})$ is the maximum $\text{nbs}(G, H)$, where G and H are exemplar genomes derived from $\mathcal{G}$ and $\mathcal{H}$. Again we have $\text{enbs}(\mathcal{G}, \mathcal{H}) = n - 1 - \text{ebd}(\mathcal{G}, \mathcal{H})$.

The Exemplar Non-breaking Similarity (ENbS) Problem is formally defined as follows:
Instance: Genomes $\mathcal{G}$ and $\mathcal{H}$, each is of length $O(m)$ and each covers n identical gene families (i.e., at least one gene from each of the n gene families appears in both $\mathcal{G}$ and $\mathcal{H}$); integer K.
Question: Are there two respective exemplar genomes of $\mathcal{G}$ and $\mathcal{H}$, G and H, such that the non-breaking similarity between them is at least K?
In the next two sections, we present several results for the optimization versions of these problems, namely, to compute or approximate the maximum value K in the above formulation. Given a maximization problem Π, let the optimal solution of Π be OPT. We say that an approximation algorithm $\mathcal{A}$ provides a *performance guarantee* of α for Π if for every instance I of Π, the solution value returned by $\mathcal{A}$ is at least OPT/α. (Usually we say that $\mathcal{A}$ is a factor-α approximation for Π.) Typically we are interested in polynomial time approximation algorithms.

7.3 Inapproximability Results

For the ENbS problem, let O_{ENbS} be the corresponding optimal solution value. Apparently we have the following lemma.

Lemma 7.1. $0 \leq O_{ENbS} \leq n - 1$.

Proof: Let the n gene families be denoted by $1, 2, ..., n$. We only consider the corresponding exemplar genomes G, H. The lower bound of O_{ENbS} is achieved by setting $G = 123 \cdots (n-1)n$ and H can be set as follows: when n is even, $H = (n-1)(n-3) \cdots 531n(n-2) \cdots 642$; when n is odd, $H = (n-1)(n-3) \cdots 642n135 \cdots (n-4)(n-2)$. It can be easily proved that between G, H there is no non-breaking point. The upper bound of O_{ENbS} is obtained by setting $G = H$ in which case any two adjacent genes form a non-breaking point. ∎

The above lemma also implies that different from the Exemplar Breakpoint Distance (EBD) problem, which does not admit any approximation at all (as deciding whether the optimal solution value is zero is NP-complete), the same cannot be said on ENbS. Given $\mathcal{G}$

and $\mathcal{H}$, it can be easily shown that deciding whether $O_{ENbS} = 0$ can be done in polynomial time (hence it is easy to decide whether there exists some approximation for ENbS—for instance, as $O_{ENbS} \leq n-1$, if we can decide that $O_{ENbS} \neq 0$ then it is easy to obtain a factor-$O(n)$ approximation for ENbS). However, the next theorem shows that even when one of $\mathcal{G}$ and $\mathcal{H}$ is given exemplar ENbS still does not admit a factor-$n^{1-\epsilon}$ approximation.

Theorem 7.2. *If one of $\mathcal{G}$ and $\mathcal{H}$ is exemplar and the other is 2-repetitive, the Exemplar Non-breaking Similarity Problem does not admit a factor $n^{1-\epsilon}$ approximation unless P=NP.*

Proof: We use a reduction from Independent Set to the Exemplar Non-breaking Similarity Problem in which each of the n genes appears in $\mathcal{G}$ exactly once and in $\mathcal{H}$ at most twice. Independent Set is a well known NP-complete problem which cannot be approximated within a factor of $n^{1-\epsilon}$ [52].

Given a graph $T = (V, E), V = \{v_1, v_2, \cdots, v_N\}, E = \{e_1, e_2, \cdots, e_M\}$, we construct $\mathcal{G}$ and $\mathcal{H}$ as follows. (We assume that the vertices and edges are sorted by their corresponding indices.) Let A_i be the sorted sequence of edges incident to v_i. For each v_i we add v_i' as an additional gene and for each e_i we add x_i, x_i' as additional genes. We have two cases: $N+M$ is even and $N+M$ is odd. We mainly focus on the case when $N+M$ is even. In this case, the reduction is as follows.

Define $Y_i = v_i A_i v_i'$, if $i \leq N$ and $Y_{N+i} = x_i x_i'$, if $i \leq M$.

$\mathcal{G} : v_1 v_1' v_2 v_2' \cdots v_N v_N' x_1 e_1 x_1' x_2 e_2 x_2' \cdots x_M e_M x_M'$.

$\mathcal{H} : Y_{N+M-1} Y_{N+M-3} \cdots Y_1 Y_{N+M} Y_{N+M-2} \cdots Y_2$.

(Construct $\mathcal{H}$ as $Y_{N+M-1} Y_{N+M-3} \cdots Y_2 Y_{N+M} Y_1 Y_3 \cdots Y_{N+M-2}$ when $N+M$ is odd. The remaining arguments will be identical.)

We claim that T has an independent set of size k iff the exemplar non-breaking similarity between $\mathcal{G}$ and $\mathcal{H}$ is k. Notice that $\mathcal{G}$ is already an exemplar genome, so $G = \mathcal{G}$.

If T has an independent set of size k, then the claim is trivial. Firstly, construct the exemplar genome H as follows. For all i, if v_i is in the independent set, then delete A_i in $Y_i = v_i A_i v_i'$ (also delete all redundant edges in A_s in $\mathcal{H}$ for which v_s is not in the independent set of T). There are k non-breaking points between G, H—notice that any vertex v_i which is in the independent set gives us a non-breaking point $v_i v_i'$. The final exemplar genomes obtained, G and H, obviously have k exemplar non-breaking points.

If the number of the exemplar non-breaking points between $\mathcal{G}$ and $\mathcal{H}$ is k, the first thing to notice is that $Y_i = x_i x_i'$ $(N < i \leq N+M)$ cannot give us any non-breaking point. So the non-breaking points must come from $Y_i = v_i A_i v_i'$ $(i \leq N)$, with some A_i properly deleted (i.e., such a Y_i becomes $v_i v_i'$ in H). Moreover, there are exactly k such A_i's deleted. We show below that any two such completely deleted A_i, A_j correspond to two independent vertices v_i, v_j in T. Assume that there is an edge e_{ij} between v_i and v_j, then as both A_i, A_j are deleted, both of the two occurrences of the gene e_{ij} will be deleted from $\mathcal{H}$. A contradiction. Therefore, if the number of the exemplar non-breaking points between $\mathcal{G}$ and $\mathcal{H}$ is k, there is an independent set of size k in T.

To conclude the proof of this theorem, notice that the reduction take polynomial time (proportional to the size of T). ∎

In the example shown in Figure 7.1, we have

$\mathcal{G} : v_1 v_1' v_2 v_2' v_3 v_3' v_4 v_4' v_5 v_5' x_1 e_1 x_1' x_2 e_2 x_2' x_3 e_3 x_3' x_4 e_4 x_4' x_5 e_5 x_5'$ and

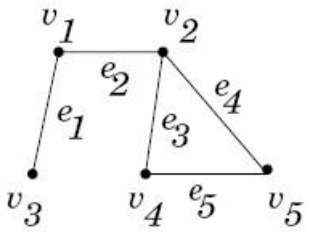


Figure 7.1: Illustration of a simple graph for the reduction

$\mathcal{H} : x_4x'_4x_2x'_2v_5e_4e_5v'_5v_3e_1v'_3v_1e_1e_2v'_1x_5x'_5x_3x'_3x_1x'_1v_4e_3e_5v'_4v_2e_2e_3e_4v'_2$.
Corresponding to the optimal independent set $\{v_3, v_4\}$, we have
$H : x_4x'_4x_2x'_2v_5e_5v'_5v_3v'_3v_1e_1e_2v'_1x_5x'_5x_3x'_3x_1x'_1v_4v'_4v_2e_2e_3e_4v'_2$. The two non-breaking points are $[v_3v'_3], [v_4v'_4]$.

We comment that EBD and ENbS, even though complement to each other, are still different problems. With respect to the above theorem, when $\mathcal{G}$ is exemplar and $\mathcal{H}$ is not, there is a factor-$O(\log n)$ approximation for the EBD problem [25]. This is significantly different from ENbS, as shown in the above theorem.

7.4 Polynomial Time Algorithms for Some Special Cases

The proof of Theorem 1 also implies that ENbS is W[1]-complete, as Independent Set is W[1]-complete [32]. Following the recent lower bound results of Chen, *et al.*, if k is the optimal solution value for ENbS then unless an unlikely collapse occurs in parameterized complexity theory, ENbS is not solvable in time $f(k)n^{o(k)}$, for any function f [20]. Nevertheless, we show below that for several practically interesting cases of the problem, there are polynomial time algorithms. The idea is to set a parameter in the input genomes (or sequences, as we will use interchangeably from now on) and design a polynomial time algorithm when such a parameter is $O(\log n)$.

We first present a few extra definitions. For a genome $\mathcal{G}$ and a character g, $\text{span}(g, \mathcal{G})$ is the maximal distance between the two positions that are occupied by g in the genome $\mathcal{G}$. For example, if $\mathcal{G} = abcbaa$, $\text{span}(a, \mathcal{G}) = 5$ and $\text{span}(b, \mathcal{G}) = 2$. For a genome $\mathcal{G}$ and $c \geq 0$, we define $\text{totalocc}(c, \mathcal{G}) = \sum_{g \text{ is a character in } \mathcal{G} \text{ and } \text{span}(g,\mathcal{G}) \geq c} \text{occ}(g, \mathcal{G})$.

Assume that c and d are positive integers. A (c, d)-*even partition* for a genome $\mathcal{G}$ is $\mathcal{G} = \mathcal{G}_1\mathcal{G}_2\mathcal{G}_3$ with $|\mathcal{G}_2| = c$ and $|\mathcal{G}_1| + \lfloor |\mathcal{G}_2|/2 \rfloor = d$.

For a genome $\mathcal{G}$ and integers $c, d > 0$, a (c, d)-split G_1, G_2, G_3 for $\mathcal{G}$ is derived from a (c', d)-even partition $\mathcal{G} = \mathcal{G}_1\mathcal{G}_2\mathcal{G}_3$ for $\mathcal{G}$ for some $c \leq c' \leq 2c$ and satisfies the following conditions 1)-6):

(1) $\text{alphabet}(\mathcal{G}) = \text{alphabet}(G_1G_2G_3)$.

(2) We can further partition $\mathcal{G}_2$ into $\mathcal{G}_2 = \mathcal{G}_2^1\mathcal{G}_2^2\mathcal{G}_2^3$ such that $|\mathcal{G}_2^2| \leq c + 1$, and there is at least one gene g with all its occurrences in $\mathcal{G}$ being in $\mathcal{G}_2^2$. We call such a gene g as a whole gene in $\mathcal{G}_2^2$.

(3) G_2 is obtained from $\mathcal{G}_2^2$ by deleting some genes and every gene appears at most once in G_2. And, G_2 contains one occurrence of every whole gene in $\mathcal{G}_2^2$.

(4) G_1 is obtained from $\mathcal{G}_1\mathcal{G}_2^1$ by deleting all genes in $\mathcal{G}_1\mathcal{G}_2^1$ which also appear in G_2.

(5) G_3 is obtained from $\mathcal{G}_2^3\mathcal{G}_3$ by deleting all genes in $\mathcal{G}_2^3\mathcal{G}_3$ which also appear in G_2.

(6) G_2 has no gene common with either G_1 or G_3.

Finally, for a genome $\mathcal{G}$ and integers $c, d \geq 0$, a (c,d)-decomposition is G_1x, G_2G_3, where G_1, G_2, G_3 is a (c,d)-split for $\mathcal{G}$ and x is the first character of G_2. We have the following lemma. In the following, whenever a different pair of genomes are given we assume that they are drawn from the same n gene families.

Lemma 7.3. *Assume that c, d are integers satisfying $c \geq 0$ and $|\mathcal{G}| - 2c \geq d \geq 2c$. and $\mathcal{G}$ is a genome with* $\mathrm{span}(g, \mathcal{G}) \leq c$ *for every gene g in $\mathcal{G}$. Then, (1) the number of (c,d)-decompositions is at most 2^{c+1}; (2) every exemplar genome of $\mathcal{G}$ is also an exemplar genome of $G_1G_2G_3$ for some (c,d)-split G_1, G_2, G_3 of $\mathcal{G}$.*

Proof: (1). Since $\mathrm{span}(g, \mathcal{G}) \leq c$ for every gene g in $\mathcal{G}$, it is easy to see that there is a c', $c \leq c' \leq 2c$, such that we can find (c,d)-splits G_1, G_2 and G_3 from a (c',d)-even partition $\mathcal{G} = \mathcal{G}_1\mathcal{G}_2\mathcal{G}_3$ with $\mathcal{G}_2 = \mathcal{G}_2^1\mathcal{G}_2^2\mathcal{G}_2^3$. Since $|\mathcal{G}_2^2| \leq c+1$, there are at most 2^{c+1} possible ways to obtain G_2. Therefore, the total number of decompositions is at most 2^{c+1}. (2) is easy to see. ∎

Lemma 7.4. *Let c be a positive constant and ϵ be an arbitrary small positive constant. There exists an $O(n^{c+2+\epsilon})$-time algorithm such that given an exemplar genome G, in which each genes appears exactly once, and $\mathcal{H}$, in which* $\mathrm{span}(g, \mathcal{H}) \leq c$ *for every g in $\mathcal{H}$, it returns* $\mathrm{enbs}(G, \mathcal{H})$.

Proof: We use the divide-and-conquer method to compute $\mathrm{enbs}(G, \mathcal{H})$. The separator is put at the middle of $\mathcal{H}$ with width c. The genes within the region of separator are handled by a brute-force method.

Algorithm
$A(G, \mathcal{H})$
  Input: G is a genome with no gene repetition,
    and $\mathcal{H}$ is a genome such that $\mathrm{span}(g, \mathcal{H}) \leq c$ for each gene in $\mathcal{H}$.
  let $s = 0$ and $d = |\mathcal{H}|/2$.
  for every (c,d)-decomposition H_1x, H_2H_3 of $\mathcal{H}$)
    begin
      if the length of H_1x and H_2H_3 is $\leq \log n$
        then compute $A(G, H_1x)$ and $A(G, H_2H_3)$ by brute-force;
        else let $s' = A(G, H_1x) + A(G, H_2H_3)$;
      if $(s < s')$ **then** $s = s'$
    end
  return s;
End of Algorithm

The correctness of the algorithm is easy to verify. By Lemma 7.3 and the description of the algorithm, the computational time is based on the following recursive equation: $T(n) \leq (2^{c+1}(2T(n/2+c)) + c_0n$, where c_0 is a constant. We show by induction that $T(n) \leq c_1n^{c+2+\epsilon}$, where c_1 is a positive constant. The basis is trivial when n is small since we can select constant c_1 large enough. Assume that $T(n) \leq c_1n^{c+2+\epsilon}$ is true all $n < m$.

$T(m) \leq 2^{c+1}(2T(m/2+c) + c_0 m \leq 2(2^{c+1}c_1(m/2+c)^{c+2+\epsilon}) + c_0 m < c_1 m^{c+2+\epsilon}$ for all large m.

∎

We now have the following theorem.

Theorem 7.5. *Let c be a positive constant. There exists an $O(3^{\lfloor t/3 \rfloor} n^{c+2+\epsilon})$-time algorithm such that given two genomes $\mathcal{G}$ and $\mathcal{H}$ with $t = \text{totalocc}(1, \mathcal{G}) + \text{totalocc}(c, \mathcal{H})$, it returns $\text{enbs}(\mathcal{G}, \mathcal{H})$.*

Proof:

Algorithm:

$d = 0$;
for each gene g_1 in $\mathcal{G}$ with $\text{span}(g_1, \mathcal{G}) \geq 1$
begin
  for each position p_1 of g_1 in $\mathcal{G}$
  begin
    remove all g_1's at all positions other than p_1;
  end
  assume that $\mathcal{G}$ has been changed to G;
  for each gene g_2 in $\mathcal{H}$ with $\text{span}(g_2, \mathcal{H}) > c$
  begin
    for each position p_2 of g_2 in $\mathcal{H}$
    begin
      remove all g_2's at all positions other than p_2;
    end
    assume that $\mathcal{H}$ has been changed to $\mathcal{H}'$;
    compute $d_0 = \text{enbs}(G, \mathcal{H}')$ following Lemma 7.4;
    if $(d < d_0)$ **then** $d = d_0$;
  end
end
return d;

End of Algorithm

Let g_i, $1 \leq i \leq m$, be the genes in $\mathcal{G}$ and $\mathcal{H}$ with $\text{span}(g_1, \mathcal{G}) \geq 1$ in $\mathcal{G}$ or $\text{span}(g_2, \mathcal{H}) > c$ in $\mathcal{H}$. We have $t = k_1 + \cdots + k_m$. Let k_i be the number of occurences of g_i. Notice that $k_i \geq 2$. The number of cases to select the positions of those genes in $\mathcal{G}$ and the positions of those genes in $\mathcal{H}$ is at most $k_1 \cdots k_m$, which is at most $4 \cdot 3^{\lfloor t/3 \rfloor}$ by Lemma 6 in the Appendix. In G, every gene appears exactly once. In $\mathcal{H}'$, every gene has span bounded by c. Therefore, their distance can be computed in $O(n^{c+2+\epsilon})$ steps by Lemma 7.4.

∎

Lemma 7.6. *Let $k \geq 3$ be a fixed integer. Assume that $k_1, k_2, \cdots, k_m$ are m integers that satisfies $k_i \geq 2$ for $i = 1, 2, \cdots, m$ and $k_1 + k_2 + \cdots + k_m = k$. Then $k_1 k_2 \cdots k_m \leq 4 \cdot 3^{\lfloor \frac{k}{3} \rfloor}$.*

Proof: We assume that for fixed k, m is the largest integer that makes the product $k_1k_2\cdots k_m$ maximal and $k_1+k_2+\cdots+k_m=k$. We claim that $k_i\leq 3$ for all $i=1,2,\cdots,m$. Otherwise, without loss of generality, we assume that $k_m>3$. Clearly, $2\cdot(k_m-2)\geq k_m$. Replace k_m by $k'_m=2$ and $k_{m+1'}=k_m-2$. We still have that $k_1+k_2+\cdots+k_{m-1}+k'_m+k'_{m+1}=k$ and $k_1k_2\cdot k_{m-1}k'_mk'_{m+1}\geq k_1k_2\cdots k_m$. This contradicts that m is maximal. Therefore, each $k_i(i=1,2,\cdots,m)$ is either 2 or 3 while $k_1+k_2+\cdots+k_{m-1}+k_m=k$ and $k_1k_2\cdots k_m$ is still maximal. It is impossible that there are at least three 2s among $k_1,k_2,\cdots,k_m$. This is because that $2+2+2=3+3$ and $2\cdot 2\cdot 2<3\cdot 3$. On the other hand, the number of 3s among $k_1,k_2,\cdots,k_m$ is at most $\lfloor\frac{k}{3}\rfloor$ since $k_1+k_2+\cdots+k_{m-1}+k_m=k$. ∎

Next, we define a new parameter measure similar to the Maximum Adjacency Disruption (MAD) number in [17].

Assume that $\mathcal{G}$ and $\mathcal{H}$ are two genomes/sequences. For a gene g, define $\text{shift}(g,\mathcal{G},\mathcal{H})=\max_{\mathcal{G}[i]=g,\mathcal{H}[j]=g}|i-j|$, where $\mathcal{G}[i]$ is the gene/character of $\mathcal{G}$ at position i. A *space-permitted* genome $\mathcal{G}$ may have space symbols in it. For two space-permitted genomes $\mathcal{G}_1$ and $\mathcal{G}_2$, a non-breaking point g_1g_2 satisfies that g_1 and g_2 appear at two positions of $\mathcal{G}$ without any other genes/characters except some spaces between them, and also at two positions of $\mathcal{H}$ without any other genes except spaces between them.

For a genome $\mathcal{G}$ and integers $c,d>0$, an exact (c,d)-split G_1,G_2,G_3 for $\mathcal{G}$ is obtained from a (c,d)-even partition $\mathcal{G}=\mathcal{G}_1\mathcal{G}_2\mathcal{G}_3$ for $\mathcal{G}$ and satisfies the following conditions (1)-(5):

(1) $\text{alphabet}(\mathcal{G})=\text{alphabet}(G_1G_2G_3)$.

(2) G_2 is obtained from $\mathcal{G}_2$ by replacing some characters with spaces and every non-space character appears at most once in G_2.

(3) G_1 is obtained from $\mathcal{G}_1$ by changing all $\mathcal{G}$ characters that also appear in G_2 into spaces.

(4) G_3 is obtained from $\mathcal{G}_3$ by changing all $\mathcal{G}_3$ characters that also appear in G_2 into spaces.

(5) G_2 has no common non-space character with either G_1 or G_3.

We now show the following lemmas.

Lemma 7.7. *Let c,k,d be positive integers. Assume that $\mathcal{G}$ is a space-permitted genome with $\text{span}(g,\mathcal{G})\leq c$ for every character g in $\mathcal{G}$, and $\mathcal{G}$ only has spaces at the first kc positions and spaces at the last kc positions. If $|\mathcal{G}|>2(k+4)c$ and $(k+2)c<d<|\mathcal{G}|-(k+2)c$, then $\mathcal{G}$ has at least one exact $(2c,d)$-split and for every exact $(2c,d)$-split G_1,G_2,G_3 for $\mathcal{G}$, G_2 has at least one non-space character.*

Proof: For $(k+2)c<d<|\mathcal{G}|-(k+2)c$, it is easy to see that $\mathcal{G}$ has a subsequence S of length $2c$ that starts from the d-th position in $\mathcal{G}$ and has no space character. For every subsequence S of length $2c$ of $\mathcal{G}$, if S has no space character, it has at least one character in $\mathcal{G}$ that only appears in the region of S since $\text{span}(g,\mathcal{G})\leq c$ for every character g in $\mathcal{G}$. ∎

Lemma 7.8. *Let c be a positive constant. There exists an $O(n^{2c+1+\epsilon})$ time algorithm such that, given two space-permitted genomes/sequences $\mathcal{G}$ and $\mathcal{H}$, it returns $\text{enbs}(\mathcal{G},\mathcal{H})$, if $\text{shift}(g,\mathcal{G},\mathcal{H})\leq c$ for each non-space character g, $\mathcal{G}$ and $\mathcal{H}$ only have spaces at the first and last $4c$ positions, and $|\mathcal{G}|\geq 16c$ and $|\mathcal{H}|\geq 16c$.*

Proof: Since $\text{shift}(g, \mathcal{G}, \mathcal{H}) \leq c$ for every gene/character g in $\mathcal{G}$ or $\mathcal{H}$, we have $\text{span}(g, \mathcal{G}) \leq 2c$ and $\text{span}(g, \mathcal{H}) \leq 2c$ for every character g in $\mathcal{G}$ or $\mathcal{H}$.

Algorithm

$B(\mathcal{G}, \mathcal{H})$

Input: $\mathcal{G}, \mathcal{H}$ are two space-permitted genomes.

assume that $|\mathcal{G}| \leq |\mathcal{H}|$;

let $s = 0$ and $d = \lfloor |\mathcal{G}|/2 \rfloor$;

for every exact $(2c, d)$-split G_1, G_2, G_3 of $\mathcal{G}$

begin

for every exact $(2c, d)$-split H_1, H_2, H_3 of $\mathcal{H}$

begin

if the length of $\mathcal{G}$ and $\mathcal{H}$ is $\leq \log n$

then compute $\text{enbs}(\mathcal{G}, \mathcal{H})$ by brute-force;

else let $s = B(G_1G_2, H_1H_2) + B(G_2G_3, H_2H_3) - B(G_2, H_2)$;

if $(s < s')$ **then** $s = s'$;

end

end

return s;

End of Algorithm

Following the divide-and-conquer method, it is easy to see that G_1G_2, H_1H_2, G_2G_3 and H_2H_3 have spaces in the first and last $2c$ positions. This is because $\text{span}(g, \mathcal{G}) \leq 2c, \text{span}(g, \mathcal{H}) \leq 2c$ for every character g. $B(G_2, H_2)$ can be determined by a linear scan, since both of them are exemplar. The computational time is determined by the recurrence relation: $T(n) = (2^{2c} + 2c)(2T(\frac{n}{2} + 2c) + O(n))$, which has solution $T(n) = O(n^{2c+1+\epsilon})$ as we show in the Lemma 7.4. ∎

Finally, we have the following theorem.

Theorem 7.9. *Let c be a positive constant. There exists an $O(3^{\lfloor t/3 \rfloor} n^{2c+1+\epsilon})$ time algorithm such that given two genomes $\mathcal{G}$ and $\mathcal{H}$ with a total of t genes g satisfies* $\text{shift}(g, \mathcal{G}, \mathcal{H}) > c$, *it returns* $\text{enbs}(\mathcal{G}, \mathcal{H})$.

The idea to prove this theorem is as follows. We consider all possible ways to replace every gene g, $\text{shift}(g, \mathcal{G}, \mathcal{H}) > c$, with space in $\mathcal{G}$ and $\mathcal{H}$, while keeping one occurence of g in $\mathcal{G}$ and $\mathcal{H}$. For each pair of such resulting $\mathcal{G}'$ and $\mathcal{H}'$, we consider to use the agorithm in Lemma 5 to compute $\text{enbs}(\mathcal{G}', \mathcal{H}')$. Notice that we may have spaces not only in the two ends but also in the middle of $\mathcal{G}'$ or $\mathcal{H}'$. However, we can modify the method of selecting exact (c, d)-splits for the two genome. The new method is to start at the middle position of $\mathcal{G}'$ (or $\mathcal{H}'$) to find the nearest non-space gene either in the right part or the left of the middle position. Say, such a gene is u in the right part of the middle position of $\mathcal{H}'$. Then, we determine $\mathcal{H}_2$ by including c positions to the right of u and also including c or more positions to the left to make sure that the middle position is also included. The rest part in the left of $\mathcal{H}_2$ is $\mathcal{H}_1$, and the rest in the right of $\mathcal{H}_2$ is $\mathcal{H}_3$. It is easy to see that the numnber of genes (not spaces) in $\mathcal{H}_2$ is no more than $2c$. Similarly, we can determine an even partition for $\mathcal{G}_1$. Notice also that spaces do not contribute to constructing exact (c, d)-splits. Therefore, $\text{enbs}(\mathcal{G}', \mathcal{H}')$ can be computed, following the spirit of the algorithm in Lemma 5.

Summary We have defined a new measure—non-breaking similarity of genomes and have proved that the exemplar version of the problem does not admit an approximation of factor $n^{1-\epsilon}$, and moreover, the problem is W[1]-complete. On the other hand, we have presented polynomial time algorithms for several practically interesting cases.

Chapter 8

Concluding Remarks and Future Work

8.1 Concluding Remarks

In this dissertation some structure and sequence analysis related bioinformatics problems are discussed. Some algorithms are presented, experiments are performed and the algorithms are evaluated on the basis of experimental results and/or theoretical analysis.

In Chapter 2 a graph based algorithm with two phases is proposed for the pairwise protein structure alignment. The algorithm is $RMSD$ flexible in that it allows the user to define a reasonable maximum $RMSD$ value for it. To test the performance of this preliminary algorithm, two sets with 30 pairs of protein backbone chains are established and tests are performed to compare our algorithm with two famous algorithms, DaliLite and CE. The results show that in many cases our algorithm obtains a larger N_{mat} when the $RMSD$ is the same or smaller.

Chapter 3 improves the preliminary algorithm by introducing the concept of "double-center stars" and the self-learning strategy. The updated algorithm is tested on a much stricter basis. Various test sets from related papers are collected, 224 pairs of chains are used, and the algorithm is compared with DaliLite, CE and SSM (another widely used protein structure alignment tool). According to the results, the updated algorithm shows better performance than the preliminary one. In most cases, when compared with DaliLite, CE and SSM, the N_{mat} of our algorithm is larger when its $RMSD$ is the same or smaller, and interestingly, this trend is more obvious in the cases where the structural similarities between the test chain pairs are weak. The algorithm has been implemented as a web alignment tool and is available for public access at `http://fpsa.cs.uno.edu/` with a mirror web site at `http://fpsa.cs.panam.edu/FPSA/`.

Supported by the above protein structure alignment algorithm, in Chapter 4 an algorithm for finding similar structures with a given structure in the Protein Data Bank is developed. It first designs a series of filters to scan the whole database and exclude those dissimilar structures step by step, and then with the help of an optimized structure alignment algorithm, it selects and ranks those structures that are most similar to the query structure. The filters narrow down the search space from over 100,000 chains to 20,000, 2,000 and finally several hundred in average. Also, with our model of comparison with other tools and 88 protein chains from different categories, the performance of our algorithm is compared with SSM. The experiments indicate that our algorithm misses fewer similar structures than SSM does, when the similarity is measured by both the maximum Q-score

and the average Q-score, where Q-score is a measurement proposed by SSM to evaluate the similarity between protein chain pairs. Our web query tool is available for public access at `http://fpsa.cs.panam.edu/FPSQ/`.

The development of the above algorithms brings an interesting problem about the calculation of the diameter of a 3-D sequence of points, which is discussed in Chapter 5. A series of deterministic, zero-error and bounded-error randomized algorithms are presented to approximate the diameter in sub-linear time, and a class of tight separations about the computational capability of each approximate algorithm is obtained. Interestingly enough, these discoveries originated from our protein structure alignment research become a bridge connecting our work and the field of computational complexity theory.

Although the primary effort of this dissertation is on protein structure alignment and its extended problems, it also discusses some sequence analysis related topics. In Chapter 6 a probabilistic model and some algorithms are proposed for reconstructing haplotypes from SNP matrices with both incomplete and inconsistent errors. Two copies of the haplotypes are randomly selected and other copies are classified according to their distances to those two class centers. The classification is repeated a number of times to ensure that each of the two centers belongs to a different haplotype. The experiments with simulated data show both high accuracy and high speed of our algorithms, which conform with the provable theoretical efficiency and accuracy of the algorithms.

In Chapter 7 the problem of genome comparison is studied based on the non-breaking similarity. The concept of non-breaking similarity is introduced as the complement of traditional breakpoint distance, and the problem of computing exemplar breakpoint distance, a minimization problem, is converted to maximizing the number of non-breaking points between genomes. Since the Independent Set problem can be linearly reduced to this problem, it is proved that the exemplar non-breaking similarity problem does not admit any factor-$n^{1-\epsilon}$ approximation unless P=NP, and the problem is also W[1]-complete. Further, for several practically interesting cases of the Exemplar Non-breaking Similarity problem, some polynomial time algorithms are presented.

8.2 List of Publications

The research effort put forth in this dissertation has resulted in the following publications:

1. "A Flexible algorithm for pairwise protein structure alignment", Zhiyu Zhao and Bin Fu, Proceedings International Conference on Bioinformatics and Computational Biology 2007 (see also [103]).

2. "Feedback Algorithm and Web-Server for Protein Structure Alignment", Zhiyu Zhao, Bin Fu, Francisco J. Alanis and Christopher M. Summa, Journal of Computational Biology, June 2008; a compact version is to be presented in the 7th Annual International Conference on Computational Systems Bioinformatics (CSB'08)) (see also [104]).

3. "New Algorithm and Web Server for Finding Proteins with Similar 3D Structures", Zaixin Lu, Zhiyu Zhao, Sergio Garcia and Bin Fu, accepted by the 2008 International Conference on Bioinformatics & Computational Biology (BIOCOMP'08) (see also [71]).

4. "Separating Sublinear Time Computations by Approximate Diameter", Bin Fu and Zhiyu Zhao, accepted by the second Annual International Conference on Combinatorial Optimization and Applications (COCOA'08) (see also [42]).

5. "Linear Time Probabilistic Algorithms for the Singular Haplotype Reconstruction Problem from SNP Fragments", Zhixiang Chen, Bin Fu, Robert Schweller, Boting Yang, Zhiyu Zhao and Binhai Zhu, Journal of Computational Biology, June 2008; a preliminary version was presented in the Proceedings of The Sixth Asia Pacific Bioinformatics Conference (APBC'08) (see also [23]).

6. "Non-breaking Similarity of Genomes with Gene Repetitions", Zhixiang Chen, Bin Fu, Jinhui Xu, Boting Yang, Zhiyu Zhao and Binhai Zhu, Proceedings of 18th Annual Symposium on Combinatorial Pattern Matching (CPM'07), LNCS 4580 (see also [24]).

8.3 Future Work

There is still a long way to go within the research scope covered in this dissertation, since it deals with some fundamental problems in Bioinformatics. Regarding the structure analysis area, considering other useful information such as sequence information, secondary structures and side chain information may be possible ways to improve our current protein structure alignment approach. Also, the speed performance of our protein structure alignment and query tools can be improved by applying those discussed methods. The improvements will increase the usability and reliability of our tools. In addition, the sublinear time algorithms for approximating the diameter of a sequence of points in a metric space may be used to solve other related problems, such as the furthest neighbor problem which is extensively applied in database and streaming applications. Furthermore, it is well known that the multiple protein structure alignment problem is NP-hard. We have found that our approach for the pairwise protein structure alignment, in which a clique can be well approximated by a star, can bring efficient heuristic algorithms for the multiple structure alignment. We will make efforts in attacking this difficult problem. Protein structure prediction, one of the most challenging problems in Bioinformatics, is also a very interesting research topic that I would like to pursue in my scientific career. Regarding the sequence analysis problems in this dissertation, it should be pointed out that the work about haplotype reconstruction can be extended to reconstructing multiple haplotypes from a set of fragments. Our approach also opens a door to develop probabilistic methods for other variants of the haplotyping problem involving both inconsistency and incompleteness errors. Also, the polynomial time algorithms for several practically interesting cases discussed in the genome rearrangement chapter inspire us to develop applicable algorithms for other similar problems. In practice, the practical datasets usually have some special properties so those negative results might not hold. Working along this line may be a good way to discover efficient and useful methods for attacking some difficult bioinformatics problems.

Bibliography

[1] http://www.rcsb.org/pdb/home/home.do.

[2] http://www.ncbi.nlm.nih.gov/sites/entrez?db=nucleotide.

[3] http://en.wikipedia.org/.

[4] http://scop.mrc-lmb.cam.ac.uk/scop/.

[5] http://www.ebi.ac.uk/DaliLite/index.html.

[6] http://cl.sdsc.edu/ce/ce_align.html.

[7] http://www.ebi.ac.uk/msd-srv/ssm/cgi-bin/ssmserver.

[8] http://www.wwpdb.org/.

[9] M. S. Alessandro Panconesi. Fast hare: A fast heuristic for single individual snp haplotype reconstruction. In *Algorithms in Bioinformatics, 4th International Workshop, WABI 2004, Lecture Notes in Computer Science 3240*, pages 266–277, 2004.

[10] N. N. Alexandrov and D. Fischer. Analysis of topological and montopological structural similarities in the pdb: new examples from old structures. *Proteins*, 25:354–365, 1996.

[11] C. B. Anfinsen. Principles that govern the folding of protein chains. *Science*, 181:223–230, 1973.

[12] M. Badoiu, A. Czumaj, P. Indyk, and C. Sohler. Facility location in sublinear time. In *Proceedings of 32nd Annual International Colloquium on Automata, Languages and Programming*, pages 866–877, 2005.

[13] V. Bafna, S. Istrail, G. Lancia, and R. Rizzi. Polynomial and apx-hard cases of the individual haplotyping problem. *Theoretical Computer Science*, 335:109–125, 2005.

[14] G. Blin and R. Rizzi. Conserved interval distance computation between non-trivial genomes. In *Proc. 11th Intl. Ann. Comput. and Combinatorics (COCOON'05), LNCS 3595*, pages 22–31, 2005.

[15] D. Bryant. *The complexity of calculating exemplar distances, In D. Sankoff and J. Nadeau, editors, Comparative Genomics: Empirical and Analytical Approaches to Gene Order Dynamics, Map Alignment.* Kluwer Acad. Pub., 2000.

[16] O. Camoglu, T. Kahveci, and A. K. Singh. Psi: Indexing protein structures for fast similarity search. In *Proceedings of Elventh International Conference on Intelligent Systems for Molecular Biology*, pages 81–83, 2003.

[17] C. Chauve, G. Fertin, R. Rizzi, and S. Vialette. Genomes containing duplicates are hard to compare. In *Proc. 2nd Intl. Workshop on Bioinformatics Research and Applications (IWBRA'06), LNCS 3992*, pages 783–790, 2006.

[18] B. Chazelle, D. Liu, and A. Magen. Sublinear geometric algorithms. *SIAM Journal on Computing*, 35:627–646, 2005.

[19] B. Chazelle, R. Rubfinfeld, and L. Trevisan. Approximating the minimum spanning tree weight in sublinear time. *SIAM Journal on computing*, 34:1370–1379, 2005.

[20] J. Chen, X. Huang, I. Kanj, and G. Xia. Linear fpt reductions and computational lower bounds. In *Proc. 36th ACM Symp. on Theory Comput. (STOC'04)*, pages 212–221, 2004.

[21] L. Chen and B. Fu. Linear and sublinear time algorithms for the basis of abelian groups. *Electronic Colloquium on Computational Complexity, TR07-052*, 2007.

[22] Z. Chen, B. Fu, R. Fowler, and B. Zhu. Lower bounds on the application of the exemplar conserved interval distance problem of genomes. In *Proc. 12th Intl. Ann. Comput. and Combinatorics (COCOON'06), LNCS 4112*, pages 245–254, 2006.

[23] Z. Chen, B. Fu, R. Schweller, B. Yang, Z. Zhao, and B. Zhu. Linear time probabilistic algorithms for the singular haplotype reconstruction problem from snp fragments. *Journal of Computational Biology*, 15(5):535 – 546, June 2008. (A preliminary version was presented in the Proceedings of The Sixth Asia Pacific Bioinformatics Conference (APBC'08), pages 333–342).

[24] Z. Chen, B. Fu, J. Xu, B. Yang, Z. Zhao, and B. Zhu. Non-breaking similarity of genomes with gene repetitions. In *Proceedings of 18th Annual Symposium on Combinatorial Pattern Matching (CPM'07), LNCS 4580*, pages 119–130, 2007.

[25] Z. Chen, B. Fu, and B. Zhu. The approximability of the exemplar breakpoint distance problem. In *Proc. 2nd Intl. Conf. on Algorithmic Aspects in Information and Management (AAIM'06), LNCS 4041*, pages 291–302, 2006.

[26] L. P. Chew, K. Kedem, D. P. Huttenlocher, and J. Kleinberg. Fast detection of geometric substructure in proteins. *Journal of Computational Biology*, 6(3–4):313–325, 1999.

[27] P.-H. Chi, G. Scott, and C.-R. Shyu. A fast protein structure retrieval system using image-based distance matrices and multidimensional index. In *Proceedings of the 4th IEEE Symposium on Bioinformatics and Bioengineering*, pages 522–532, 2004.

[28] R. Cilibrasi, L. van Iersel, S. Kelk, and J. Tromp. On the complexity of several haplotyping problems. In *Algorithms in Bioinformatics, 5th International Workshop, WABI 2005, Lecture Notes in Computer Science*, volume 3692, pages 128–139, 2005.

[29] A. Clark. Inference of haplotypes from pcr-amplified samples of diploid populations. *Molecular Biology Evolution*, 7:111–122, 1990.

[30] A. Czumaj, F. Ergun, L. Fortnow, I. N. A. Magen, R. Rubinfeld, and C. Sohler. Sublinear approximation of euclidean minimum spanning tree. *SIAM Journal on Computing*, 35:91109, 2005.

[31] A. Czumaj and C. Sohler. Estimating the weight of metric minimum spanning trees in sublinear-time. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, pages 175–183, 2004.

[32] R. Downey and M. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.

[33] P. Drineas and R. Kannan. Fast monte-carlo algorithms for approximate matrix multiplication. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*, page 452459, 2001.

[34] D. Eggert, A. Lorusso, and R. Fisher. A comparison of four algorithms for estimating 3-d rigid transformations. In *British Machine Vision Conference*, pages 237–246, 1995.

[35] I. Eidhammer, I. Jonassen, and W. R. Taylor. *Protein Bioinformatics: An Algorithmic Approach to Sequence and Structure Analysis*. John Wiley and Sons, 2004.

[36] A. Falicov and F. E. Cohen. A surface of minimum area metric for the structureal comparison of protein. *Journal of Molecular Biology*, 258:871–892, 1996.

[37] U. Feige. On sumes of independent random variables with unbounded variance and estimating the average degree in a graph. *SIAM Journal on Computing*, 35:964–984, 2006.

[38] D. Fischer, A. Elofsson, D. Rice, and D. Eisenberg. Assessing the performance of fold recognition methods by means of a comprehensive benchmark. In *Proc. 1st Pacific Symposium on Biocomputing*, pages 300–318, 1996.

[39] D. Fischer, R. Nussinov, and H. Wolfson. 3d substructure matching in protein molecules. In *Proc. 3rd Intl Symp. Combinatorial Pattern Matching, LNCS 644*, pages 136–150, 1992.

[40] E. Fischer. The art of uninformed decision: A primer to property testing. *Bulletin of the EATCS*, 75:97–126, 2001.

[41] B. Fu and Z. Chen. Sublinear-time algorithms for width-bounded geometric separators and their applications to protein side-chain packing problems. *Accepted by the Journal of Combinatorial Optimization, the preliminary version was presented in AAIM'06, Lecture Notes in Computer Science 3328, pages 149-160.*

[42] B. Fu and Z. Zhao. Separating sublinear time computations by approximate diameter. In *the second Annual International Conference on Combinatorial Optimization and Applications (COCOA'08)*, 2008. (Accepted).

[43] L. M. Genovese, F. Geraci, and M. Pellegrini. A fast and accurate heuristic for the single individual SNP haplotyping problem with many gaps, high reading error rate and low coverage. *Algorithms in Bioinformatics*, 4645:49–60, 2007.

[44] A. Godzik. The structural alignment between two proteins: Is there a unique answer? *Protein Science*, 5:1325–1338, 1996.

[45] O. Goldreich. Combinatorial proterty testing (a survey). In *In* P. Pardalos, S. Rajasekaran, and J. Rolim *editors, Proceesdings of the* DIMACS *workshop on radnomziation methods in algorithm design, volume 43 of* DIMACS*, series in Discrete Mathematics and Theoretical Computer Science*, pages 45–59, 1997.

[46] O. Goldreich. Property testing in massive graphs. *In J. Abello, P. M. Pardalos, and M. Resende, editor, Handbook of massive data sets*, pages 123–147, 2002.

[47] O. Goldreich and D. Ron. On testing expansion in bounded-degree graphs. Technical Report 00-20, Electronic Colloquium on Computational Complexity, http://www.eccc.uni-trier.de/eccc/, 2000.

[48] O. Goldreich and D. Ron. Approximating average parameters of graphs. Technical Report 05-73, Electronic Colloquium on Computational Complexity, http://www.eccc.uni-trier.de/eccc/, 2005.

[49] D. Gusfield. A practical algorithm for optimal inference of haplotype from diploid populations. pages 183–189, 2000.

[50] D. Gusfield. Haplotyping as perfect phylogeny: Conceptual framework and efficient solutions. In *the Sixth Annual International Conference on Computational Biology*, pages 166–175, 2002.

[51] S. Hannenhalli and P. Pevzner. Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals. *J. ACM*, 46(1):1–27, 1999.

[52] J. Hastad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, 182:105–142, 1999.

[53] M. Hoehe, K. Kopke, B. Wendel, K. Rohde, C. Flachmeier, K. Kidd, W. Berrettini, and G. Church. Sequence variability and candidate gene analysis in complex disease: association of opioid receptor gene variation with substance dependence. *Human Molecular Genetics*, 9(19):2895–2908, 2000.

[54] L. Holm and C. Sander. Protein structure comparison by alignment of distance matrices. *Journal of Molecular Biology*, 233:123–138, 1993.

[55] V. A. Ilyin, A. Abyzov, and C. M.Leslin. Structural alignment of proteins by a novel topofit method, as a superimposition of common volumes at a topomax point. *Protein Science*, 13:1865–1874, 2004.

[56] I. Koch, T. Lengauer, and E. Wanke. An algorithm for finding maximal common subtopolgies in a set of protein structures. *Journal of Computational Biology*, 3-2:289–306, 1996.

[57] R. Kolodny, P. Koehl, and M. Levitt. Comprehensive evaluation of protein structure alignment methods: scoring by geometric measures. *Journal of Molecular Biology*, 346(4):1173–1188, 2005.

[58] R. Kolodny, N. Linial, and M. Levitt. Approximate protein structural alignment in polynomial time. *Proceedings of the National Academy of Sciences of the United States of America*, 101(33):12201–12206, 2004.

[59] E. Krissinel and K. Henrick. Secondary-structure matching (ssm), a new tool for fast protein structure alignment in three dimensions. *Acta Crystallographica*, D60:2256–2268, 2004.

[60] R. Kumar and R. Rubinfeld. Sublinear time algorithms. *SIGACT News*, 34:57–67, 2003.

[61] G. Lancia, M. C. Pinotti, and R. Rizzi. Haplotyping polulations by purs parsimongy: complexity and algorithms. *INFORMS Journal on computing*, 16:348–359, 2004.

[62] G. Lancia and R. Rizzi. A polynomial solution to a special case of the parsimongy haplotyping problem. *Operations Research letters*, 34:289–295, 2006.

[63] R. H. Lathrop. The protein threading problem with sequence amino acid interaction preferences is np-complete. *Protein Engineering*, 7:1059–1068, 1994.

[64] A. M. Lesk. *Introduction to Bioinformatics*. Oxford University Press Inc., 2002.

[65] U. Lessel and D. Schomburg. Similarities between protein 3-d structures. *Protein Engineering*, 7(10):1175–1187, 1994.

[66] M. Levitt. Growth of novel protein structural data. *Proceedings of the National Academy of Sciences of the United States of America*, 104:3183–3188, 2007.

[67] L. Li, J. H. Kim, and M. S. Waterman. Haplotype reconstruction from SNP alignment. *Journal of Computational Biology*, 11(2–3):507–518, 2004.

[68] M. Li, B. Ma, and L. Wang. On the closest string and substring problems. *Journal of the ACM*, 49(2):157–171, 2002.

[69] R. Lippert, R. Schwartz, G. Lancia, and S. Istrail. Algorithmic stragegies for the single nucleotide polymorphism haplotype assembly problem. *Briefings in bioinformatics*, 3:23–31, 2002.

[70] W. Liu, F. Mao, L. Lai, and Y. Han. Protein fold recognition based on structural classification. *Acta Biophysica Sinica*, 15:126–136, 1999.

[71] Z. Lu, Z. Zhao, S. Garcia, and B. Fu. New algorithm and web server for finding proteins with similar 3d structures. In *the 2008 International Conference on Bioinformatics & Computational Biology (BIOCOMP'08)*, 2008. (Accepted).

[72] T. Madej, J. F. Gibrat, and S. H. Bryant. Threading a database of protein cores. *Proteins*, 23:356–369, 1995.

[73] K. Mizguchi and N. Go. Comparison of spatial arrangements of secondary structural elements in proteins. *Protein Eng.*, 8:353–362, 1995.

[74] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 2000.

[75] A. G. Murzin, S. E. Brenner, T. Hubbard, and C. Chothia. Scop: a structural classification of proteins database for the investigation of sequences and structures. *Journal of Molecular Biology*, 247:536–540, 1995.

[76] C. Nguyen, Y. Tay, and L. Zhang. Divide-and-conquer approach for the exemplar breakpoint distance. *Bioinformatics*, 21(10):2171–2176, 2005.

[77] e. O. Gascuel. *Mathematics of Evolution and Phylogeny*. Oxford University Press, 2004.

[78] S. G. O. Goldreich and D. Ron. Property testing and its connection to learning and approximation. *J. ACM*, 45:653750, 1998.

[79] A. Ortiz, C. Strauss, and O. Olmea. Mammoth (matching molecular models obtained from theory): an automated method for model comparison. *Protein Science*, 11(11):2606–2621, 2002.

[80] G. A. Petsko and D. Ringe. *Protein Structure and Function*. New Science Press, 2004.

[81] R. Rizzi, V. Bafna, S. Istrail, and G. Lancia. Practical algorithms and fixed-parameter tractability for the single individual snp haplotyping problem. In *Algorithms in Bioinformatics: Second International Workshop, WABI 2002, Rome, Italy, September 17-21*, pages 29–43, 2002.

[82] D. Ron. Handbok of randomzied algorithm. *Bulletin of the EATCS*, II:597–649, 2001.

[83] S. D. Rufino and T. L. Blundell. Structure-based identification and clustering of protein families and superfamilies. *Journal of Comput. Aided. Mol. Dec.*, 233:123–138, 1994.

[84] D. Sankoff. Genome rearrangement with gene families. *Bioinformatics*, 16(11):909–917, 1999.

[85] I. N. Shindyalov and P. E. Bourne. Protein structure alignment by incremental combinatorial extension (ce) of the optimal path. *Protein Engineering*, 11:739–747, 1998.

[86] A. P. Singh and D. L. Brutlag. Hierarchical protein superposition using both secondary structure and atomic representation. In *Proc. Intelligent Systems for Molecular Biology*, pages 284–293, 1997.

[87] A. Sturtevant and T. Dobzhansky. Inversions in the third chromosome of wild races of *drosophila pseudoobscura*, and their use in the study of the history of the species. *Proceedings of the National Academy of Sciences of the United States of America*, 22:448–450, 1936.

[88] W. R. Taylor. Protein structure comparison using iterated double dynamic programming. *Protein Science*, 9:654–665, 1999.

[89] W. R. Taylor and C. Orengo. Protein structure alignment. *Journal of Molecular Biology*, 208(1):1–22, 1989.

[90] J. Terwilliger and K. Weiss. Linkage disquilibrium mapping of complex disease: fantasy and reality? *Current Opinion in Biotechnology*, 9(6):578–594, 1998.

[91] W. F. Trench. *Advanced Calculus*. Harper & Row, New York, 1978.

[92] S. Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(4):376–380, 1991.

[93] R.-S. Wang, L.-Y. Wu, Z.-P. Li, and X.-S. Zhang. Hyplotype reconstruction from SNP fragments by minimum error correction. *BioInformatics*, 21(10):2456–2462, 2005.

[94] R.-S. Wang, L.-Y. Wu, X.-S. Zhang, and L. Chen. A markov chain model for haplotype assembly from SNP fragments. *Genome Informatics*, 17(2):162–171, 2006.

[95] G. Watterson, W. Ewens, T. Hall, and A. Morgan. The chromosome inversion problem. *J. Theoretical Biology*, 99:1–7, 1982.

[96] M. Xie and J. Wang. An improved (and practical) parameterized algorithm for the individual haplotyping problem MFR with mate-pairs. *Algorithmica (online)*, 2007.

[97] J. Ye, R. Janardan, and S. Liu. Pairwise protein structure alignment based on an orientation-independent backbone representation. *Journal of Bioinformatics and Computational Biology*, 4(2):699–717, 2005.

[98] Y. Ye and A. Godzik. Database searching by flexible protein structure alignment. *Protein Science*, 13(7):1841–1850, 2004.

[99] G. Yona and K. Kedem. The urms-rms hybrid algorithm for fast and sensitive local protein structure alignment. *Journal of Computational Biology*, 12:12–32, 2005.

[100] X.-S. Zhang, R.-S. Wang, L.-Y. Wu, and W. Zhang. Minimum conflict individual haplotyping from SNP fragments and related genotype. *Evolutionary Bioinformatics Online*, 2:271–280, 2006.

[101] Y. Zhang and J. Skolnick. Tm-align: a protein structure alignment algorithm based on the tm-score. *Nucleic Acids Research*, 33:2302–2309, 2005.

[102] Y. Zhao, L. Wu, J. Zhang, and X. Z. R. Wang. Haplotype assembly from aligned weighted SNP fragments. *Computational Biology and Chemistry*, 29:281–287, 2005.

[103] Z. Zhao and B. Fu. A flexible algorithm for pairwise protein structure alignment. In *Proceedings International Conference on Bioinformatics and Computational Biology 2007*, pages 16–22, 2007.

[104] Z. Zhao, B. Fu, F. J. Alanis, and C. M. Summa. Feedback algorithm and web-server for protein structure alignment. *Journal of Computational Biology*, 15(5):505 – 524, June 2008. (A compact version is to be presented in the 7th Annual International Conference on Computational Systems Bioinformatics (CSB'08)).

[105] M. Zimand. On derandomizing probabilistic sublinear-time algorithms. In *Proceedings of the 22nd IEEE conference on computational complexity*, pages 1–9, 2007.

Chapter A

Appendix

A.1 Some Tables in Chapter 3

See Tables A.1 through A.3 for result details of Figure 3.4.

A.2 A Table in Chapter 4

See Table A.4 for result details of 88 protein queries.

A.3 Some Details in Chapter 5

Corollary A.1. *Assume that c is a positive constant, d is a fxied dimension number, α is a constant in $(0,1)$, and ϵ is a small constant greater than 0. Let t be a positive real number. Then there exists a deterministic $O(\frac{n}{m}+(\frac{1}{\epsilon^{2d}}))$-time algorithm such that given an $\epsilon(1-\alpha)m/2$-*reliable*-rearrangement sequence B for a t-sequence A of n points in R^d with diameter at least $m \cdot t$, it outputs a number x with $(1-\epsilon)\text{diameter}(A) \leq x \leq \text{diameter}(A)$.*

Proof: We just need to prove that for any constant $\delta \in (0,1)$, there exists an $O(k+(\frac{1}{\delta^{2d}}))$ time $(1-\delta)$-factor approximate algorithm App_{R^d} to compute the diameter of k points set H in R^d. Let d be a fixed dimensional number. Find a $\frac{1}{2}$-factor approximate diameter D of H (see the proof of Corollary 5.4). The approximate diameter D can be found in time $O(k)$ as described in the proof of Corollary 5.4. There exists a $(4D)^d$ cube region G that contains all points in H. Partition G into small cubes of size $(\frac{\delta D}{2\sqrt{d}})^d$. For each cube C that contains points in H, select one point from $H \cap C$ and put it into set Q. The number of small cubes of size $(\frac{\delta D}{2\sqrt{d}})^d$ in G is at most $O((\frac{1}{\delta})^d)$ since d is fixed. We have $|Q| = O((\frac{1}{\delta})^d)$. Compute the diameter of Q by brute force method in time $O(|Q|^2)$. ∎

Lemma A.2. *For any even number n and two numbers $p_1 < p_2$ in R^1, there exists a $\text{dist}(p_2,p_1)$-sequence $S = p_1 q_1 q_2 \cdots q_{n-2} p_2$ in R^1 such that $p_1 < q_i$ for $i = 1, \cdots, n-2$ and $\text{diameter}(S) \geq \frac{n \cdot \text{dist}(p_1,p_2)}{2}$. The sequence S is denoted as $\text{unfolding}_{R^1}(p_1,p_2,n)$.*

Table A.1: Comparing SLIPSA with CE

	CE	SLIPSA			CE	SLIPSA	
No.	n/r	n/r	n^+/r^-	No.	n/r	n/r	n^+/r^-
136	48/5.2	79/5.2	64.58/0.00	62	84/4.5	87/4.5	3.57/0.00
73	56/4.6	92/4.4	64.29/4.35	139	84/3.5	87/3.5	3.57/0.00
135	56/4.5	81/4.3	44.64/4.44	13	253/2.6	262/2.6	3.56/0.00
221	64/5.3	90/5.2	40.63/1.89	143	116/3.9	120/3.9	3.45/0.00
61	171/3.9	227/3.9	32.75/0.00	37	366/3.4	378/3.4	3.28/0.00
20	54/4	70/4	29.63/0.00	210	92/2	95/2	3.26/0.00
133	56/7.3	71/7.3	26.79/0.00	10	248/2.4	256/2.4	3.23/0.00
88	80/5.4	101/5.3	26.25/1.85	174	124/4.2	128/4.1	3.23/2.38
182	80/6	101/6	26.25/0.00	170	125/4	129/4	3.20/0.00
180	80/6	100/6	25.00/0.00	82	97/2.9	100/2.9	3.09/0.00
90	64/7	78/6.6	21.88/5.71	87	130/3.1	134/3.1	3.08/0.00
179	88/4.5	107/4.5	21.59/0.00	51	394/3.1	406/3.1	3.05/0.00
181	62/3.9	75/3.9	20.97/0.00	31	133/1.8	137/1.8	3.01/0.00
131	72/5	86/4.9	19.44/2.00	162	139/2.3	143/2.3	2.88/0.00
127	80/4.1	95/4	18.75/2.44	134	70/2.7	72/2.7	2.86/0.00
138	48/2.9	56/2.9	16.67/0.00	151	140/2.1	144/2.1	2.86/0.00
107	54/3	62/2.9	14.81/3.33	77	107/3.9	110/3.9	2.80/0.00
100	97/5.6	111/5.6	14.43/0.00	166	114/3.8	117/3.8	2.63/0.00
176	88/4.3	100/4.3	13.64/0.00	56	115/3.2	118/3.2	2.61/0.00
178	75/4.4	85/4.3	13.33/2.27	89	115/3.2	118/3.2	2.61/0.00
95	115/5.8	130/5.7	13.04/1.72	8	236/2.5	242/2.5	2.54/0.00
177	62/3.8	70/3.8	12.90/0.00	122	40/5.3	41/5.3	2.50/0.00
129	48/3.6	54/3.5	12.50/2.78	80	84/2.9	86/2.9	2.38/0.00
142	80/4.1	90/4.1	12.50/0.00	6	257/2.7	263/2.7	2.33/0.00
224	56/5.5	63/5.3	12.50/3.64	154	137/2.1	140/1.9	2.19/9.52
36	48/4.3	53/4.3	10.42/0.00	158	137/2.1	140/1.9	2.19/9.52
92	107/4.2	118/4.2	10.28/0.00	125	92/4	94/3.9	2.17/2.50
200	78/3.2	86/3.2	10.26/0.00	209	92/2	94/2	2.17/0.00
104	71/3.2	78/3.1	9.86/3.13	70	187/2.4	191/2.4	2.14/0.00
103	64/2.6	70/2.6	9.38/0.00	64	94/4.1	96/4	2.13/2.44
19	111/4.2	120/4.2	8.11/0.00	97	94/4.1	96/4	2.13/2.44
130	64/4	69/4	7.81/0.00	29	97/1.9	99/1.9	2.06/0.00
141	64/4.4	69/4.2	7.81/4.55	23	98/2.2	100/2.2	2.04/0.00
69	76/2	81/2	6.58/0.00	47	50/1.1	51/1.1	2.00/0.00
169	117/3.5	124/3.5	5.98/0.00	71	104/4.3	106/4.3	1.92/0.00
84	275/3	291/3	5.82/0.00	46	159/2	162/2	1.89/0.00
219	86/2.2	91/2.1	5.81/4.55	44	54/1.9	55/1.8	1.85/5.26
98	94/4.3	99/4.1	5.32/4.65	165	112/3	114/3	1.79/0.00
4	302/1.5	318/1.5	5.30/0.00	18	244/3	248/3	1.64/0.00
65	268/3.1	282/3.1	5.22/0.00	28	122/1.9	124/1.9	1.64/0.00
99	97/5.4	102/5.4	5.15/0.00	43	61/3	62/3	1.64/0.00
96	117/4.3	123/4.2	5.13/2.33	111	61/3.2	62/3.1	1.64/3.13
183	62/4.4	65/4.4	4.84/0.00	137	61/2.7	62/2.7	1.64/0.00
76	64/3.8	67/3.5	4.69/7.89	172	122/3.8	124/3.8	1.64/0.00
173	130/4.9	136/4.9	4.62/0.00	94	124/2.7	126/2.7	1.61/0.00
48	87/1.9	91/1.9	4.60/0.00	7	259/2.7	263/2.7	1.54/0.00
211	92/2.1	96/2.1	4.35/0.00	49	266/3	270/3	1.50/0.00
214	92/2.5	96/2.4	4.35/4.00	160	133/1.9	135/1.7	1.50/10.53
168	117/3.3	122/3.3	4.27/0.00	66	68/2.3	69/2.2	1.47/4.35
101	118/5	123/4.9	4.24/2.00	150	141/1.9	143/1.9	1.42/0.00
171	119/3.5	124/3.5	4.20/0.00	202	71/2.6	72/2.5	1.41/3.85
208	97/2.4	101/2.4	4.12/0.00	205	71/2.7	72/2.7	1.41/0.00
116	25/1.3	26/1.3	4.00/0.00	147	143/1.6	145/1.6	1.40/0.00
11	253/2.6	263/2.6	3.95/0.00	204	73/3.1	74/3	1.37/3.23
52	156/1.9	162/1.9	3.85/0.00	206	79/3.3	80/3.3	1.27/0.00
74	78/1.7	81/1.7	3.85/0.00	42	80/1.8	81/1.7	1.25/5.56

(table A.1 continued)

	CE	SLIPSA			CE	SLIPSA	
No.	n/r	n/r	n^+/r^-	No.	n/r	n/r	n^+/r^-
132	80/3.3	81/3.3	1.25/0.00	121	55/1.5	55/1.5	0.00/0.00
12	252/2.2	255/2.2	1.19/0.00	144	152/0.3	152/0.3	0.00/0.00
123	336/1.6	340/1.6	1.19/0.00	145	152/0.5	152/0.5	0.00/0.00
39	177/2.8	179/2.8	1.13/0.00	146	153/0.6	153/0.6	0.00/0.00
26	280/2.2	283/2.2	1.07/0.00	148	141/1.6	141/1.5	0.00/6.25
207	101/2.3	102/2.3	0.99/0.00	152	137/1.6	137/1.5	0.00/6.25
79	108/3.6	109/3.3	0.93/8.33	155	138/1.6	138/1.6	0.00/0.00
83	219/3.8	221/3.8	0.91/0.00	156	137/1.7	137/1.6	0.00/5.88
167	111/3.1	112/3.1	0.90/0.00	157	136/1.7	136/1.6	0.00/5.88
140	115/4.1	116/4.1	0.87/0.00	161	144/2.2	144/2.2	0.00/0.00
14	252/2.4	254/2.4	0.79/0.00	164	141/2.2	141/2.2	0.00/0.00
22	383/2.8	386/2.8	0.78/0.00	175	117/3.2	117/3.2	0.00/0.00
159	138/2	139/2	0.72/0.00	184	105/0.4	105/0.4	0.00/0.00
149	141/1.8	142/1.8	0.71/0.00	185	105/0.7	105/0.7	0.00/0.00
153	141/1.8	142/1.8	0.71/0.00	186	105/1.3	105/1.3	0.00/0.00
163	146/2.6	147/2.6	0.68/0.00	187	105/1.3	105/1.3	0.00/0.00
40	154/3	155/3	0.65/0.00	190	105/1.5	105/1.5	0.00/0.00
33	157/3.1	158/3.1	0.64/0.00	191	105/1.5	105/1.5	0.00/0.00
45	415/3.2	417/3.2	0.48/0.00	193	104/1.6	104/1.6	0.00/0.00
60	214/2.4	215/2.4	0.47/0.00	194	104/1.6	104/1.6	0.00/0.00
15	257/3.7	258/3.7	0.39/0.00	198	101/2.1	101/1.9	0.00/9.52
9	259/2.7	260/2.7	0.39/0.00	199	89/3.4	89/3.3	0.00/2.94
50	261/2.4	262/2.4	0.38/0.00	212	96/2.4	96/2.4	0.00/0.00
58	296/3	297/3	0.34/0.00	213	100/3.5	100/3.3	0.00/5.71
1	336/0.3	336/0.3	0.00/0.00	217	96/3.2	96/3.1	0.00/3.13
2	336/0.4	336/0.4	0.00/0.00	223	73/2.6	73/2.5	0.00/3.85
3	336/0.5	336/0.5	0.00/0.00	16	253/3.4	252/3.4	-0.40/0.00
5	254/1.4	254/1.4	0.00/0.00	72	147/3.7	146/3.4	-0.68/8.11
21	128/1.9	128/1.9	0.00/0.00	93	266/2.3	264/2.3	-0.75/0.00
24	69/2.3	69/2.1	0.00/8.70	53	121/2.6	120/2.6	-0.83/0.00
25	96/3.3	96/3.3	0.00/0.00	54	117/1.9	116/1.9	-0.85/0.00
30	346/1.8	346/1.8	0.00/0.00	192	105/1.5	104/1.4	-0.95/6.67
34	68/2	68/1.9	0.00/5.00	188	104/1.5	103/1.4	-0.96/6.67
35	169/3.5	169/3.5	0.00/0.00	189	104/1.5	103/1.4	-0.96/6.67
57	157/3.5	157/3.5	0.00/0.00	195	103/1.8	102/1.6	-0.97/11.11
59	197/3.2	197/3.2	0.00/0.00	196	103/1.8	102/1.7	-0.97/5.56
63	91/1.8	91/1.8	0.00/0.00	197	102/1.6	101/1.5	-0.98/6.25
67	252/4.6	252/4.6	0.00/0.00	218	97/3	96/3	-1.03/0.00
75	249/2.5	249/2.5	0.00/0.00	220	96/3.2	95/3.1	-1.04/3.13
78	84/3.4	84/3.3	0.00/2.94	27	93/2.6	92/2.5	-1.08/3.85
81	81/2.3	81/2.1	0.00/8.70	38	77/3.1	76/2.9	-1.30/6.45
85	87/1.9	87/1.8	0.00/5.26	203	75/3	74/2.9	-1.33/3.33
86	116/2.9	116/2.9	0.00/0.00	222	73/2.6	72/2.6	-1.37/0.00
91	354/0.7	354/0.7	0.00/0.00	55	143/3	141/2.9	-1.40/3.33
102	93/4.5	93/4.5	0.00/0.00	109	64/3.5	63/3.5	-1.56/0.00
105	72/3.5	72/3.4	0.00/2.86	215	97/3	95/2.9	-2.06/3.33
106	65/3	65/3	0.00/0.00	216	97/3	95/2.9	-2.06/3.33
108	46/1.7	46/1.7	0.00/0.00	201	74/2.3	72/2.1	-2.70/8.70
110	38/1.3	38/1.3	0.00/0.00	41	237/3	230/3	-2.95/0.00
112	25/0.8	25/0.8	0.00/0.00	32	201/2.2	195/2.2	-2.99/0.00
113	30/0.9	30/0.9	0.00/0.00	17	256/2.8	247/2.8	-3.52/0.00
114	24/1	24/1	0.00/0.00	118	31/1.5	29/1.5	-6.45/0.00
115	24/0.6	24/0.6	0.00/0.00	68	n/a	/	/
117	32/1.6	32/1.6	0.00/0.00	124	n/a	/	/
119	33/0.6	33/0.6	0.00/0.00	126	n/a	/	/
120	92/1.1	92/1.1	0.00/0.00	128	n/a	/	/

Table A.2: Comparing SLIPSA with DaliLite

	DaliLite	SLIPSA			DaliLite	SLIPSA	
No.	n/r	n/r	n^+/r^-	No.	n/r	n/r	n^+/r^-
102	75/10.3	124/10.3	65.33/0.00	64	74/1.8	77/1.8	4.05/0.00
36	37/5.3	55/4.6	48.65/13.21	221	77/2.7	80/2.7	3.90/0.00
88	61/3.8	88/3.8	44.26/0.00	79	103/3	107/3	3.88/0.00
133	47/6.7	67/6.7	42.55/0.00	143	104/3.2	108/3.2	3.85/0.00
179	79/4.6	108/4.6	36.71/0.00	125	81/2.8	84/2.8	3.70/0.00
180	71/4.9	93/4.9	30.99/0.00	112	27/2.3	28/2.3	3.70/0.00
19	99/4.9	129/4.9	30.30/0.00	73	82/2.8	85/2.6	3.66/7.14
95	101/5.9	131/5.9	29.70/0.00	175	113/3.2	117/3.2	3.54/0.00
124	180/5.2	227/5.2	26.11/0.00	86	114/3.1	118/3.1	3.51/0.00
176	83/4.4	103/4.4	24.10/0.00	40	147/2.7	152/2.7	3.40/0.00
100	89/6.6	109/6.5	22.47/1.52	111	59/3.1	61/3.1	3.39/0.00
118	27/2.5	33/2.4	22.22/4.00	217	89/2.8	92/2.7	3.37/3.57
20	56/3.4	67/3.4	19.64/0.00	57	149/3	154/3	3.36/0.00
71	82/3.3	98/3.3	19.51/0.00	172	120/3.8	124/3.8	3.33/0.00
181	83/6.7	98/6.6	18.07/1.49	174	121/3.9	125/3.9	3.31/0.00
177	84/7	99/7	17.86/0.00	45	401/3.1	414/3.1	3.24/0.00
109	53/3.2	62/3.2	16.98/0.00	130	62/2.9	64/2.8	3.23/3.45
183	66/6.3	76/6.2	15.15/1.59	141	63/3	65/2.8	3.17/6.67
104	81/5.2	93/5.2	14.81/0.00	107	65/3.7	67/3.7	3.08/0.00
131	67/3.3	76/3.3	13.43/0.00	106	66/3.4	68/3.3	3.03/2.94
61	206/4.1	233/4.1	13.11/0.00	202	69/2.4	71/2.4	2.90/0.00
142	71/3.2	80/3.2	12.68/0.00	222	71/2.8	73/2.7	2.82/3.57
98	64/2.2	72/2.2	12.50/0.00	119	36/3	37/2.8	2.78/6.67
96	116/5.3	130/5.3	12.07/0.00	69	80/2.1	82/2.1	2.50/0.00
132	69/2.8	77/2.8	11.59/0.00	168	121/3.5	124/3.5	2.48/0.00
77	94/3.3	104/3.3	10.64/0.00	84	291/3.3	298/3.3	2.41/0.00
224	50/3.8	55/3.8	10.00/0.00	18	250/3.4	256/3.4	2.40/0.00
182	71/3.7	77/3.7	8.45/0.00	83	211/3.5	216/3.5	2.37/0.00
135	72/3.9	78/3.9	8.33/0.00	127	85/2.9	87/2.9	2.35/0.00
105	65/3.1	70/3.1	7.69/0.00	200	86/3.4	88/3.4	2.33/0.00
62	71/3.2	76/3.2	7.04/0.00	87	131/3.1	134/3.1	2.29/0.00
56	114/3.6	122/3.5	7.02/2.78	48	89/2	91/1.9	2.25/5.00
89	114/3.6	122/3.5	7.02/2.78	63	94/2.6	96/2.3	2.13/11.54
78	75/3	80/3	6.67/0.00	70	188/2.5	192/2.5	2.13/0.00
101	106/4.3	113/4.3	6.60/0.00	59	189/2.9	193/2.9	2.12/0.00
129	47/2.8	50/2.8	6.38/0.00	49	261/2.7	266/2.7	1.92/0.00
199	83/3.2	88/3.2	6.02/0.00	93	262/2.6	267/2.6	1.91/0.00
72	133/3	141/3	6.02/0.00	167	106/2.7	108/2.7	1.89/0.00
103	67/2.8	71/2.8	5.97/0.00	166	106/2.7	108/2.7	1.89/0.00
43	57/2.6	60/2.6	5.26/0.00	138	55/2.9	56/2.9	1.82/0.00
67	260/6.7	273/6	5.00/10.45	123	340/2	346/2	1.76/0.00
178	80/4.1	84/4.1	5.00/0.00	65	285/3.4	290/3.4	1.75/0.00
173	121/4	127/4	4.96/0.00	54	114/2	116/2	1.75/0.00
139	82/3.3	86/3.3	4.88/0.00	58	292/3	297/3	1.71/0.00
92	105/3.1	110/3.1	4.76/0.00	170	117/3.2	119/3.2	1.71/0.00
218	87/2.5	91/2.5	4.60/0.00	75	248/2.6	252/2.6	1.61/0.00
25	88/2.8	92/2.8	4.55/0.00	10	253/2.5	257/2.5	1.58/0.00
220	88/2.7	92/2.7	4.55/0.00	9	253/2.5	257/2.5	1.58/0.00
216	90/2.8	94/2.7	4.44/3.57	22	395/3.5	401/3.5	1.52/0.00
204	68/2.7	71/2.7	4.41/0.00	31	135/2	137/1.8	1.48/10.00
134	70/3	73/2.8	4.29/6.67	158	138/2	140/1.9	1.45/5.00
205	70/2.9	73/2.8	4.29/3.45	37	374/3.5	379/3.5	1.34/0.00
82	97/3.2	101/3.2	4.12/0.00	33	154/3	156/3	1.30/0.00
208	97/2.4	101/2.4	4.12/0.00	206	78/3.2	79/3.2	1.28/0.00
223	73/3	76/3	4.11/0.00	51	412/3.6	417/3.6	1.21/0.00
97	74/1.8	77/1.8	4.05/0.00	35	168/3.6	170/3.6	1.19/0.00

(table A.2 continued)

	DaliLite	SLIPSA			DaliLite	SLIPSA	
No.	n/r	n/r	n^+/r^-	No.	n/r	n/r	n^+/r^-
68	168/2.6	170/2.6	1.19/0.00	120	97/1.9	97/1.9	0.00/0.00
12	256/2.4	259/2.4	1.17/0.00	66	67/1.9	67/1.8	0.00/5.26
85	86/1.9	87/1.8	1.16/5.26	47	56/1.9	56/1.8	0.00/5.26
6	261/2.8	264/2.8	1.15/0.00	74	85/1.9	85/1.9	0.00/0.00
39	174/2.6	176/2.6	1.15/0.00	160	135/1.9	135/1.7	0.00/10.53
13	266/3	269/3	1.13/0.00	30	347/1.9	347/1.9	0.00/0.00
215	90/2.5	91/2.5	1.11/0.00	121	58/1.8	58/1.8	0.00/0.00
212	94/2.2	95/2.2	1.06/0.00	34	67/1.8	67/1.7	0.00/5.56
213	95/2.5	96/2.5	1.05/0.00	108	46/1.7	46/1.7	0.00/0.00
214	95/2.5	96/2.4	1.05/4.00	194	105/1.7	105/1.7	0.00/0.00
209	96/2.3	97/2.3	1.04/0.00	42	80/1.7	80/1.6	0.00/5.88
210	96/2.3	97/2.3	1.04/0.00	147	145/1.7	145/1.6	0.00/5.88
29	96/1.6	97/1.6	1.04/0.00	4	323/1.7	323/1.7	0.00/0.00
32	194/2.3	196/2.3	1.03/0.00	157	136/1.7	136/1.6	0.00/5.88
23	97/1.9	98/1.9	1.03/0.00	156	137/1.7	137/1.6	0.00/5.88
41	224/2.8	226/2.8	0.89/0.00	193	104/1.6	104/1.6	0.00/0.00
53	118/2.5	119/2.5	0.85/0.00	191	105/1.5	105/1.5	0.00/0.00
169	119/3.3	120/3.3	0.84/0.00	24	67/1.4	67/1.4	0.00/0.00
171	120/3.3	121/3.3	0.83/0.00	186	105/1.3	105/1.3	0.00/0.00
8	243/2.6	245/2.6	0.82/0.00	187	105/1.3	105/1.3	0.00/0.00
94	126/3	127/2.8	0.79/6.67	113	31/1.1	31/1.1	0.00/0.00
15	254/3.6	256/3.6	0.79/0.00	185	105/0.7	105/0.7	0.00/0.00
5	255/1.6	257/1.6	0.78/0.00	146	153/0.6	153/0.6	0.00/0.00
7	263/2.8	265/2.8	0.76/0.00	145	153/0.6	153/0.6	0.00/0.00
11	265/2.9	267/2.9	0.75/0.00	144	153/0.5	153/0.5	0.00/0.00
152	135/1.4	136/1.4	0.74/0.00	184	105/0.4	105/0.4	0.00/0.00
55	140/2.9	141/2.9	0.71/0.00	2	336/0.4	336/0.4	0.00/0.00
148	140/1.6	141/1.5	0.71/6.25	1	336/0.3	336/0.3	0.00/0.00
149	142/1.9	143/1.9	0.70/0.00	162	145/2.5	144/2.4	-0.69/4.00
26	286/2.5	288/2.5	0.70/0.00	155	139/1.7	138/1.6	-0.72/5.88
151	144/2.2	145/2.1	0.69/4.55	28	125/2	124/1.9	-0.80/5.00
52	162/2.1	163/2	0.62/4.76	192	105/1.5	104/1.4	-0.95/6.67
3	334/0.5	336/0.5	0.60/0.00	188	104/1.5	103/1.4	-0.96/6.67
16	246/3.1	247/3.1	0.41/0.00	189	104/1.5	103/1.4	-0.96/6.67
17	246/2.8	247/2.8	0.41/0.00	190	104/1.3	103/1.3	-0.96/0.00
14	261/2.9	262/2.9	0.38/0.00	196	103/1.8	102/1.7	-0.97/5.56
50	261/2.4	262/2.4	0.38/0.00	195	103/1.8	102/1.6	-0.97/11.11
203	72/2.8	72/2.7	0.00/3.57	198	102/2.1	101/1.9	-0.98/9.52
137	62/2.7	62/2.7	0.00/0.00	197	102/1.7	101/1.5	-0.98/11.76
165	111/2.7	111/2.7	0.00/0.00	136	62/3.4	61/3.4	-1.61/0.00
76	60/2.6	60/2.6	0.00/0.00	44	58/2.3	57/2.2	-1.72/4.35
27	92/2.6	92/2.5	0.00/3.85	201	73/2.2	71/2.1	-2.74/4.55
163	147/2.6	147/2.6	0.00/0.00	21	n/a	/	/
60	216/2.5	216/2.5	0.00/0.00	46	n/a	/	/
211	97/2.4	97/2.3	0.00/4.17	80	n/a	/	/
90	46/2.4	46/2.4	0.00/0.00	91	n/a	/	/
161	147/2.4	147/2.4	0.00/0.00	99	n/a	/	/
81	81/2.3	81/2.1	0.00/8.70	110	n/a	/	/
207	102/2.3	102/2.3	0.00/0.00	114	n/a	/	/
38	72/2.2	72/2.1	0.00/4.55	115	n/a	/	/
164	141/2.2	141/2.2	0.00/0.00	116	n/a	/	/
219	91/2.1	91/2.1	0.00/0.00	117	n/a	/	/
154	140/2.1	140/1.9	0.00/9.52	122	n/a	/	/
159	139/2	139/2	0.00/0.00	126	n/a	/	/
153	143/2	143/1.9	0.00/5.00	128	n/a	/	/
150	143/2	143/1.9	0.00/5.00	140	n/a	/	/

Table A.3: Comparing SLIPSA with SSM

	SSM	SLIPSA			SSM	SLIPSA	
No.	n/r	n/r	n^+/r^-	No.	n/r	n/r	n^+/r^-
77	44/2.49	92/2.46	109.09/1.20	215	83/2.39	90/2.39	8.43/0.00
116	20/2.15	31/2.04	55.00/5.12	56	108/3.11	117/3.06	8.33/1.61
102	45/3.08	68/3.03	51.11/1.62	89	108/3.11	117/3.06	8.33/1.61
178	50/3.03	72/3.03	44.00/0.00	216	84/2.52	91/2.48	8.33/1.59
180	56/3.82	77/3.8	37.50/0.52	18	204/2.19	221/2.19	8.33/0.00
96	75/2.92	103/2.89	37.33/1.03	59	170/2.36	184/2.33	8.24/1.27
183	33/1.66	43/1.58	30.30/4.82	165	99/2.43	107/2.42	8.08/0.41
67	185/4.02	236/4.02	27.57/0.00	65	251/2.85	271/2.84	7.97/0.35
134	59/3.23	74/3.08	25.42/4.64	171	113/3.39	122/3.39	7.96/0.00
103	58/2.85	72/2.78	24.14/2.46	204	66/2.73	71/2.71	7.58/0.73
179	87/4.6	106/4.53	21.84/1.52	162	132/2.27	142/2.22	7.58/2.20
181	55/3.2	67/3.11	21.82/2.81	51	375/3.07	403/3.04	7.47/0.98
88	65/3.26	79/3.25	21.54/0.31	143	109/3.93	117/3.88	7.34/1.27
167	90/2.81	109/2.79	21.11/0.71	64	69/1.69	74/1.65	7.25/2.37
177	65/4.41	78/4.37	20.00/0.91	72	130/2.94	139/2.94	6.92/0.00
135	60/3.27	72/3.24	20.00/0.92	111	58/3.25	62/3.23	6.90/0.62
182	67/3.88	80/3.88	19.40/0.00	98	73/2.93	78/2.93	6.85/0.00
85	73/2.1	87/1.82	19.18/13.33	45	371/2.69	396/2.68	6.74/0.37
73	69/2.28	82/2.22	18.84/2.63	149	134/1.91	143/1.87	6.72/2.09
101	101/4.75	119/4.73	17.82/0.42	90	45/2.6	48/2.55	6.67/1.92
203	62/2.86	73/2.86	17.74/0.00	210	90/2.11	96/2.11	6.67/0.00
83	188/3.81	221/3.78	17.55/0.79	87	121/2.69	129/2.67	6.61/0.74
176	83/3.98	97/3.93	16.87/1.26	141	61/2.94	65/2.82	6.56/4.08
36	43/4	50/3.94	16.28/1.50	17	219/2.29	233/2.27	6.39/0.87
92	93/2.97	108/2.96	16.13/0.34	160	126/1.75	134/1.69	6.35/3.43
107	56/3.67	65/3.54	16.07/3.54	166	97/2.46	103/2.43	6.19/1.22
57	131/2.99	152/2.95	16.03/1.34	127	82/2.97	87/2.88	6.10/3.03
109	57/4.29	66/4.08	15.79/4.90	170	115/3.45	122/3.45	6.09/0.00
71	70/1.98	81/1.95	15.71/1.52	8	214/1.98	227/1.96	6.07/1.01
20	61/4.14	70/3.91	14.75/5.56	105	66/3.12	70/3.11	6.06/0.32
94	109/2.67	125/2.61	14.68/2.25	35	153/3.15	162/3.13	5.88/0.63
95	89/3.13	102/3.06	14.61/2.24	147	137/1.64	145/1.62	5.84/1.22
139	71/2.74	81/2.67	14.08/2.55	136	52/2.39	55/2.28	5.77/4.60
124	146/3.28	166/3.21	13.70/2.13	211	87/1.88	92/1.85	5.75/1.60
37	323/2.99	365/2.98	13.00/0.33	117	35/3.57	37/3.13	5.71/12.32
217	77/2.43	87/2.37	12.99/2.47	214	88/2.17	93/2.17	5.68/0.00
202	64/2.51	72/2.48	12.50/1.20	82	89/2.33	94/2.27	5.62/2.58
218	80/2.39	90/2.39	12.50/0.00	38	72/3.19	76/2.91	5.56/8.78
22	329/2.47	370/2.47	12.46/0.00	43	54/2.25	57/2.15	5.56/4.44
19	108/4.36	121/4.3	12.04/1.38	209	90/2.11	95/2.06	5.56/2.37
205	67/3.36	75/3.24	11.94/3.57	84	271/2.84	286/2.83	5.54/0.35
50	227/2.01	253/2.01	11.45/0.00	10	236/2.12	249/2.11	5.51/0.47
199	79/3.36	88/3.2	11.39/4.76	132	73/2.88	77/2.83	5.48/1.74
61	202/4.05	225/4.04	11.39/0.25	208	92/2.12	97/2.11	5.43/0.47
33	137/2.74	152/2.74	10.95/0.00	142	74/3.16	78/3.1	5.41/1.90
213	84/2.14	93/2.05	10.71/4.21	174	113/3.39	119/3.32	5.31/2.06
130	58/2.88	64/2.86	10.34/0.69	164	132/2.12	139/2.1	5.30/0.94
133	49/3.74	54/3.69	10.20/1.34	6	227/1.95	239/1.94	5.29/0.51
220	80/2.39	88/2.33	10.00/2.51	58	267/2.45	281/2.44	5.24/0.41
97	70/1.85	77/1.84	10.00/0.54	150	136/1.92	143/1.9	5.15/1.04
221	72/2.59	79/2.54	9.72/1.93	80	79/2.41	83/2.38	5.06/1.24
63	87/2.4	95/2.01	9.20/16.25	60	198/2.05	208/2.05	5.05/0.00
131	66/2.94	72/2.87	9.09/2.38	76	60/2.87	63/2.77	5.00/3.48
15	212/2.7	231/2.68	8.96/0.74	25	83/2.35	87/2.35	4.82/0.00
200	81/3.71	88/3.39	8.64/8.63	75	230/2.24	241/2.21	4.78/1.34
104	70/2.97	76/2.96	8.57/0.34	79	105/3.4	110/3.34	4.76/1.76

(table A.3 continued)

	SSM	SLIPSA			SSM	SLIPSA	
No.	n/r	n/r	n^+/r^-	No.	n/r	n/r	n^+/r^-
7	231/2	242/1.99	4.76/0.50	159	134/1.79	136/1.77	1.49/1.12
48	85/1.71	89/1.69	4.71/1.17	158	136/1.82	138/1.82	1.47/0.00
106	65/3.33	68/3.3	4.62/0.90	81	79/2.12	80/2.02	1.27/4.72
120	88/1.12	92/1.12	4.55/0.00	93	241/1.45	244/1.44	1.24/0.69
68	155/2.27	162/2.25	4.52/0.88	125	82/2.68	83/2.63	1.22/1.87
151	133/1.85	139/1.83	4.51/1.08	39	169/2.39	171/2.35	1.18/1.67
123	311/1.13	325/1.12	4.50/0.88	27	89/2.36	90/2.33	1.12/1.27
172	115/3.45	120/3.42	4.35/0.87	212	92/1.98	93/1.98	1.09/0.00
169	115/3.38	120/3.27	4.35/3.25	198	98/1.88	99/1.86	1.02/1.06
222	69/2.43	72/2.41	4.35/0.82	197	98/1.34	99/1.27	1.02/5.22
223	69/2.43	72/2.39	4.35/1.65	188	102/1.42	103/1.36	0.98/4.23
62	70/2.95	73/2.89	4.29/2.03	189	102/1.42	103/1.38	0.98/2.82
9	238/2.13	248/2.11	4.20/0.94	190	102/1.3	103/1.28	0.98/1.54
11	240/2.2	250/2.2	4.17/0.00	186	102/1.09	103/1.05	0.98/3.67
140	98/3.24	102/3.21	4.08/0.93	187	102/1.07	103/1.05	0.98/1.87
207	98/2.38	102/2.31	4.08/2.94	184	103/0.39	104/0.37	0.97/5.13
112	25/1.27	26/1.21	4.00/4.72	53	110/1.88	111/1.88	0.91/0.00
206	76/3.26	79/3.23	3.95/0.92	152	134/1.37	135/1.35	0.75/1.46
42	76/1.5	79/1.47	3.95/2.00	154	137/1.83	138/1.8	0.73/1.64
5	241/1.29	250/1.29	3.73/0.00	155	137/1.74	138/1.58	0.73/9.20
49	246/2.28	255/2.26	3.66/0.88	161	141/2.19	142/2.06	0.71/5.94
40	141/2.26	146/2.23	3.55/1.33	129	49/2.8	49/2.8	0.00/0.00
137	57/2.24	59/2.16	3.51/3.57	74	85/2.14	85/1.9	0.00/11.21
168	120/3.65	124/3.61	3.33/1.10	69	81/2.08	81/2.04	0.00/1.92
16	225/2.61	232/2.6	3.11/0.38	219	90/2.06	90/2.02	0.00/1.94
66	65/1.94	67/1.77	3.08/8.76	28	124/2.04	124/1.88	0.00/7.84
196	98/1.59	101/1.56	3.06/1.89	115	29/1.93	29/1.78	0.00/7.77
195	98/1.5	101/1.44	3.06/4.00	153	143/1.91	143/1.87	0.00/2.09
31	132/1.75	136/1.7	3.03/2.86	21	127/1.83	127/1.8	0.00/1.64
30	333/1.75	343/1.72	3.00/1.71	157	136/1.66	136/1.63	0.00/1.81
156	134/1.63	138/1.63	2.99/0.00	194	103/1.62	103/1.57	0.00/3.09
163	138/2.21	142/2.17	2.90/1.81	47	53/1.39	53/1.26	0.00/9.35
54	109/1.56	112/1.54	2.75/1.28	34	63/1.38	63/1.32	0.00/4.35
70	182/2.28	187/2.26	2.75/0.88	192	102/1.35	102/1.26	0.00/6.67
201	73/3.14	75/2.62	2.74/16.56	110	38/1.27	38/1.25	0.00/1.57
32	184/1.9	189/1.88	2.72/1.05	113	31/1.11	31/1.11	0.00/0.00
175	113/3.14	116/3.13	2.65/0.32	114	24/1	24/1	0.00/0.00
78	78/3.08	80/2.99	2.56/2.92	119	33/0.61	33/0.61	0.00/0.00
100	85/3.33	87/3.31	2.35/0.60	146	153/0.56	153/0.56	0.00/0.00
55	131/2.31	134/2.31	2.29/0.00	145	152/0.47	152/0.47	0.00/0.00
26	269/2.03	275/2.02	2.23/0.49	3	336/0.46	336/0.46	0.00/0.00
108	45/1.77	46/1.74	2.22/1.69	1	336/0.33	336/0.33	0.00/0.00
148	138/1.53	141/1.53	2.17/0.00	4	311/1.38	310/1.38	-0.32/0.00
29	93/1.4	95/1.38	2.15/1.43	14	236/1.94	235/1.93	-0.42/0.52
13	244/2.2	249/2.2	2.05/0.00	2	336/0.36	333/0.36	-0.89/0.00
144	149/0.31	152/0.3	2.01/3.23	121	56/1.5	55/1.45	-1.79/3.33
193	101/1.44	103/1.43	1.98/0.69	23	93/1.31	91/1.25	-2.15/4.58
52	153/1.56	156/1.53	1.96/1.92	122	31/2.85	30/2.73	-3.23/4.21
191	102/1.35	104/1.35	1.96/0.00	118	32/1.49	28/1.44	-12.50/3.36
185	103/0.73	105/0.73	1.94/0.00	224	47/3.16	35/2.97	-25.53/6.01
41	209/2.36	213/2.34	1.91/0.85	44	n/a	/	/
173	107/2.91	109/2.87	1.87/1.37	46	n/a	/	/
138	54/2.75	55/2.68	1.85/2.55	91	n/a	/	/
86	114/2.85	116/2.84	1.75/0.35	99	n/a	/	/
12	246/2.04	250/2.04	1.63/0.00	126	n/a	/	/
24	65/1.35	66/1.32	1.54/2.22	128	n/a	/	/

Table A.4: Results of the 88 test cases

#	Our				SSM				#	Our				SSM			
	Q_1	Q_2	Q_3	Q_4	Q_1	Q_2	Q_3	Q_4		Q_1	Q_2	Q_3	Q_4	Q_1	Q_2	Q_3	Q_4
1	29	63	79	79	15	61	79	79	45	1	1	238	445	1	1	97	498
2	49	108	117	153	40	50	51	74	46	28	70	122	613	28	71	86	408
3	52	118	124	258	34	36	36	36	47	23	23	24	25	23	23	24	24
4	1	5	83	89	1	3	58	59	48	21	21	22	22	21	21	22	22
5	446	1199	1404	1449	436	1134	1309	1315	49	42	288	468	661	70	263	414	552
6	4	386	1396	1475	4	297	1276	1290	50	92	137	153	155	93	137	140	147
7	265	1233	1409	1439	4	297	1276	1290	51	1	1	2	36	1	1	6	36
8	72	307	636	700	63	240	562	637	52	92	132	152	155	93	114	142	146
9	25	26	26	31	20	20	20	26	53	115	180	204	278	123	180	202	747
10	6	1056	1400	1428	6	711	1057	1068	54	61	180	204	439	25	180	180	522
11	34	56	87	225	34	61	110	239	55	50	84	162	476	50	83	98	243
12	11	13	13	56	10	12	13	36	56	279	1179	1220	1229	276	1169	1199	1229
13	265	1229	1408	1442	266	1147	1268	1278	57	18	18	92	160	18	18	26	50
14	1	1120	1401	1421	1	411	1020	1047	58	3	4	4	35	3	4	4	33
15	4	6	6	50	4	6	6	30	59	10	18	131	156	10	16	96	117
16	15	22	25	28	15	22	23	26	60	55	65	66	71	53	59	59	61
17	15	16	46	126	15	16	31	242	61	99	370	495	696	98	321	423	566
18	2	26	44	100	2	18	27	59	62	9	9	74	153	9	9	81	152
19	415	1246	1418	1478	403	1116	1354	1397	63	51	117	185	190	51	110	143	143
20	442	1204	1399	1454	431	1113	1344	1387	64	2	2	11	200	2	2	18	144
21	7	467	1402	1453	7	363	1376	1389	65	10	10	10	10	10	10	10	10
22	265	1231	1407	1454	265	590	628	629	66	14	51	51	85	9	43	43	52
23	3	21	25	973	3	21	24	105	67	252	326	350	377	259	335	348	380
24	5	275	614	856	5	208	515	586	68	11	20	20	26	10	20	20	20
25	16	35	103	125	13	29	88	101	69	260	260	366	375	102	102	108	155
26	1	1	1	23	1	1	1	17	70	3	3	3	3	3	3	3	3
27	23	31	39	78	25	30	88	122	71	1	1	1	1	1	1	1	1
28	199	560	918	1238	180	511	902	1361	72	116	162	186	191	98	140	152	152
29	617	628	630	632	597	608	608	609	73	188	262	294	295	188	247	278	279
30	9	20	21	43	1	2	2	4	74	92	96	99	99	91	96	98	98
31	1	419	669	1243	2	389	624	1070	75	3	15	19	19	3	4	5	6
32	154	370	646	1116	145	286	505	532	76	2	13	62	68	2	33	60	62
33	5	15	75	1158	4	12	77	525	77	15	47	138	138	10	34	127	128
34	3	27	547	1037	5	21	92	160	78	4	6	6	6	4	6	6	6
35	43	84	84	321	48	50	50	50	79	120	158	192	197	120	161	175	175
36	48	315	321	370	56	272	312	320	80	8	12	13	13	3	6	6	6
37	3	5	5	5	2	4	4	4	81	8	92	172	218	9	77	160	219
38	38	38	38	102	32	32	32	59	82	72	155	171	171	72	103	103	103
39	7	22	23	36	6	17	22	22	83	45	50	62	73	45	51	59	65
40	7	40	546	1439	6	35	471	857	84	4	4	4	4	4	4	4	4
41	2	24	695	1597	1	16	681	1243	85	77	152	169	273	79	153	159	159
42	4	5	15	97	4	5	12	144	86	43	169	202	202	42	156	200	200
43	9	38	41	42	9	36	36	36	87	102	104	108	157	102	105	107	115
44	8	8	223	421	8	8	66	298	88	186	251	294	295	186	251	288	289

Q_1: number of similar proteins found with Q-score ≥ 0.8.
Q_2: number of similar proteins found with Q-score ≥ 0.6.
Q_3: number of similar proteins found with Q-score ≥ 0.4.
Q_4: number of similar proteins found with Q-score ≥ 0.2.

Proof: Let $n = 2h$ and $t = \text{dist}(p_1, p_2)$. We construct a t-sequence of n points as follows: Let $1) q_1 = p_1 + t$, $2) q_s = q_{s-1} + t$ for $s = 2, \cdots, h$, and $3) q_s = q_{s-1} - t$ for $s = h+1, h+2, \cdots, 2h-2$. It is easy to see that $S = p_1 q_1 q_2 \cdots q_{2h-2} p_2$ is a t-sequence of $n = 2h$ points in R^1 and diameter$(S) = ht = \frac{nt}{2}$. ∎

Proof: (of Theorem 5.6) Assume that C is a randomized $(1-\epsilon)$ approximate algorithm for computing the approximate diameter for all of the t-sequences of diameter at least $m \cdot t$. Let $h = 2(\lceil \frac{\epsilon m}{1-\epsilon} \rceil + 2)$, $g = 2h$ and $n = m + kg$, where k is a parameter that is flexible. Since $m = o(n)$, we always assume that $m < \frac{n}{2}$. We have $k = \frac{n-m}{g} = \frac{n-m}{4(\lceil \frac{\epsilon m}{1-\epsilon} \rceil + 2)} \leq \frac{n}{4(\lceil \epsilon m \rceil)}$. On the other hand, $k \geq \frac{(n-m)}{4(\lceil \frac{\epsilon m}{1-\epsilon} \rceil + 2)} > \frac{(n-m)}{4(\frac{\epsilon m}{1-\epsilon} + 3)} \geq \frac{(n-m)}{4(\frac{\epsilon m + 3(1-\epsilon)}{1-\epsilon})} \geq \frac{(1-\epsilon)(n-m)}{4(\epsilon + 3(1-\epsilon))m} \geq \frac{(1-\epsilon)(n-m)}{4(3-2\epsilon)m} \geq \frac{(1-\epsilon)n}{8(3-2\epsilon)m}$. Let constant $c_0 = 0.09 \cdot \frac{(1-\epsilon)}{8(3-2\epsilon)}$. Let t be a constant greater than 0.

Since each path queries $o(\frac{n}{m})$ points, we assume that every path of C queries at most $\frac{c_0 n}{m}$ points in every t-sequence A. Let A be the t-sequence of points $q_1, q_2, \cdots, q_{m+1}, p_1, p_2, \cdots, p_{n-m}$, where $q_i = (i-1)t$ for $i = 1, 2, \cdots, m+1$, $p_i = (m-1)t$ for

odd number $i = 1, 3, \cdots$, and $p_i = mt$ for even number $i = 2, 4, \cdots$. Clearly, A is a t-sequence in one dimensional axis of diameter $m \cdot t$.

Partition the points $p_1 p_2 \cdots p_{n-m}$ sequentially into $P_1 P_2 \cdots P_k$ with $|P_i| = g$. In the next phase, we will show that there exists some P_i such that no more than $10\%G$ paths of C query the points in P_i, where G is the number of total paths in C. Assume that for every P_i, there are at least $10\%G$ paths of C to query the points in P_i. Thus, the total number of queries is at least $k \cdot 10\%G > \frac{c_0 n}{m} G$ among all paths. On the other hand, since every path of C queries at most $\frac{c_0 n}{m}$ points, the total number of queries by all paths of C is at most $\frac{c_0 n}{m} G$. This is a contradiction. Therefore, we have a P_i that no more than 10% paths of C query the points in P_i.

We can arrange the points in P_i so that it has greatly different diameters. Since P_i has at least $2h$ points, we can make diameter(P_i) as large as ht and as small as t without changing the positions of first and last points of P_i. Formally, assume that P_i has the sequence of points $p_u, p_{u+1}, \cdots, p_{u+g-1}$.

Clearly, $\text{dist}(p_u, p_{u+g-1}) = t$ and $p_u < p_{u+g-1}$ by the definition of A. We replace $p_{u+1}, \cdots, p_{u+g-2}$ by $p'_{u+1}, \cdots, p'_{u+g-2}$, where $\text{unfolding}_{R^1}(p_1, p_2, g) = p_u p'_{u+1} p'_{u+2} \cdots, p'_{u+g-2} p_{u+g-1}$.

If the sequence A' is derived from A that P_i is replaced by $P'_i = p_u p'_{u+1} p'_{u+2} \cdots, p'_{u+g-2} p_{u+g-1}$. $C(A, B)$ and $C(A', B)$ will be the same at 90% paths B. On the other hand, the diameter of A is $m \cdot t$ and the diameter of A' is at least $mt + ht - t > \frac{1}{(1-\epsilon)} mt$ by Lemma A.2. Thus, C is not an $(1-\epsilon)$-approximation to the diameter of a t-sequence of n points in R^1 with diameter at least mt. A contradiction. ∎

Definition A.3. For a sequence of points $S = p_1 p_2 \cdots p_n$, and integers i and j with $1 \leq i \leq j \leq n$, define $S[i, j] = p_i p_{i+1} \cdots p_j$ and $S[i] = p_i$.

Proof: (of Theorem 5.12) Assume that A is a deterministic algorithm that computes the $(1-\epsilon)$-approximate diameter for an g-reliable rearrangement sequence B of a t-sequence S of diameter at least $m \cdot t$, where $h = 2(\left\lceil \frac{\epsilon m}{1-\epsilon} \right\rceil + 2)$ and $g = 4h$. We are going to construct a counter example for this algorithm. Our target is to design two sequences of points S, S', and S''. The sequence S' is a g-reliable rearrangement sequence B of a t-sequence S of diameter at least $m \cdot t$. When S' and S'' are the inputs to the algorithm, the same output will be returned by the algorithm. On the other hand, we make the difference between diameter(S') and diameter(S'') more than factor $(1-\epsilon)$. This brings a contradiction.

Let S be the t-sequence of points $q_1 q_2 \cdots q_{m+1} p_1 p_2 \cdots p_{n-m-1}$, where $q_i = (i-1)t$ for $i = 1, 2, \cdots, m+1$, $p_i = (m-1)t$ for odd $i = 1, 3, \cdots$ and $p_i = mt$ for even $i = 2, 4, \cdots$. Clearly, S is a t-sequence in R^1 of diameter $m \cdot t$.

Let $n = m+1+kg$. Let $u_1 = m+2$ and $v_1 = u_1 + g - 1$. Let $u_i = u_{i-1} + g$, $v_i = u_i + g - 1$ for $i = 2, \cdots, k$. Partition the sequence S into multiple subsequences $Q P_1 P_2 \cdots P_k$, where $Q = q_1 \cdots q_{m+1}$, $P_i = S[u_i, v_i]$ for $i = 1, \cdots, k$. Clearly, each P_i contains g consecutive points from S.

Each P_i is partitioned into two equal half parts $P_{i,1} = S[u_{i,1}, v_{i,1}]$ and $P_{i,2} = S[u_{i,2}, v_{i,2}]$, where $u_{i,1} = u_i, v_{i,1} = u_{i,1} + \frac{g}{2} - 1, u_{i,2} = v_{i,1} + 1, v_{i,2} = u_{i,2} + \frac{g}{2} - 1$.

A query made by the algorithm is represented by an integer j, which means to request the j-th point in the input sequence. For an input of a sequence of n points, each query

made by the algorithm is represented by an integer j with $1 \leq j \leq n$. We will construct another sequence S', which is a permutation of points in S. The sequence S' has the format $S' = QP'_1P'_2 \cdots P'_k$, where each P'_i is a permutation of points in P_i.

For two integers a and b with $1 \leq a \leq b \leq n$, define $Q_z(S', a, b)$ as the set of queries j with $j \in [a, b]$ among the first z queries made by the algorithm when the input is the sequence of n points S'.

Initially $S' = QP^*_1 \cdots P^*_k$, where each P^*_i is a sequence of special point $*$, which means the position (with $*$) has not been assigned a point and will be replaced by a real point in R^1.

In order to construct P'_i, we design sequences T_i and W_i of length n. Let T_i be a sequence of n points such that $T_i[j] = *$ for all $i = 1, 2, \cdots, n$. Let W_i be a sequence of n points such that $W_i[j] = S[j]$ if $j \in [u_{i,2}, v_{i,2}]$, and $W_i[j] = *$ otherwise. We construct S' by simulating the queries made by the algorithm one by one. Each query made by the algorithm A has a stage for it as follows.

Stage z

Let j_z be the z-th query made by the algorithm. If $S'[j_z] \neq *$, reply to the query with answer $S'[j_z]$ and simulate the next query by entering Stage $z + 1$.

Assume that the algorithm makes the z-th query $j_z \in [u_i, v_i]$ for some $i \in [1, k]$ (If $j_z \in [1, m + 1]$ the answer to the query will be $S'[j_z] = q_{j_z}$), and $S'[j_z] = *$.

- Case 1: $|Q_z(S', u_i, v_i)| < \frac{|P_i|}{2}$. Do the following cases
  - Subcase 1: $j_z \in [u_{i,1}, v_{i,1}]$ and $S'[j_z] = *$. Assume that r is the first position with $W_i[r] \neq *$, and $r \in [u_{i,2}, v_{i,2}]$. Let $S'[j_z] = S[r]$, $T_i[j_z] = S[j_z]$ and $W_i[r] = *$. The purpose of the last a few assignments is to swap two points in the regions $[u_{i,1}, v_{i,1}]$ and $[u_{i,2}, v_{i,2}]$ in the sequence S.
  - Subcase 2: $j_z \in [u_{i,2}, v_{i,2}]$ and $S'[j_z] = *$. Assume that r is the first position with $W_i[r] \neq *$ and $r \in [u_{i,2}, v_{i,2}]$. Let $S'[j_z] = S[r]$ and $W_i[r] = *$. In this case, we find the position j_z that has been used for swapping from a point in the region $[u_{i,1}, v_{i,1}]$. Therefore, find another point in the same region $[u_{i,2}, v_{i,2}]$ to fill it.

    Reply to the query with answer $S'[j_z]$.
- Case 2: $|Q_z(S', u_i, v_i)| = \frac{|P_i|}{2}$. In this case, the region $[u_i, v_i]$ has been queried more than $\frac{|P_i|}{2}$ times. We do not do any swapping and just fill those undetermined positions marked with '*' by some points.

  For each position $r \in [u_i, v_i]$ with $S'[r] = *$

  If there exists s such that $T_i[s] \neq *$

  Then let $S'[r] = S[s]$ and $T_i[s] = *$

  Else find an s with $W_i[s] \neq *$, and let $S'[s] = S[s]$ and $W_i[s] = *$.

  After this case, $S'[u_i, v_i]$ has no any *. Reply to the query with answer $S'[j_z]$.
- Case 3: $|Q_z(S', u_i, v_i)| > \frac{|P_i|}{2}$. Before reaching Case 3, the construction has met Case 2 because $|Q_z(S', u_i, v_i)| \leq |Q_{z-1}(S', u_i, v_i)| + 1$. Therefore, $S'[u_i, v_i]$ has no $*$. The

answer to the query will be $S'[j_z]$ for the query j_z made by the algorithm with input S'.

End of Stage z

When we have simulated all the queried, we fill those positions, which are not queried by the algorithm, with the points in S, which are not used for replying the queries before.

Stage $z_0 + 1$ (z_0 is the total number of queries)

For each P_i and each position $r \in [u_i, v_i]$ with $S'[r] = *$,

If (there exists s such that $T_i[s] \neq *$)

Then let $S'[r] = S[s]$ and $T_i[s] = *$.

Else find an s with $W_i[s] \neq *$, and let $S'[s] = S[s]$ and $W_i[s] = *$

End of Stage $z_0 + 1$

For every computation that makes at most $(n - m - 1)/2$ queries, there exists an integer i such that the number of points queried in P_i is no more than $g/2 = 2h$ times. Assume P_i is queried no more than $\frac{|P_i|}{2}$ times by the algorithm. By the construction at Case 1, we have that no element of $P_{i,1}$ is queried by the algorithm.

The sequence S' is a $g = 4h$-reliable rearrangement of S since each P'_j is a permutation of P_j and $|P_j| = g$. We have diameter(S) = diameter(S'). In order to construct S'', We adjust the points in $P_{i,1}$ so that it has greatly different diameters from diameter(S'). We can make diameter$(P_{i,1})$ as large as $h \cdot t$ or as small as t. This makes diameter(S) more than $(\frac{m}{1-\epsilon}) \cdot t$ and also as small as $m \cdot t$ without changing the positions of first and last points of P_i. Formally, assume that $P_{i,1}$ has the sequence of points $p_u, p_{u+1}, \cdots, p_{u+2h-1}$, where $1 \leq i \leq k - 1$.

Clearly, $\text{dist}(p_u, p_{u+g-1}) = t$ and $p_u < p_{u+g-1}$ by the definition of sequence S. We replace $p_{u+1}, \cdots, p_{u+2h-2}$ by $p'_{u+1}, \cdots, p'_{u+2h-2}$, where $p_u p'_{u+1}, \cdots, p'_{u+2h-2} p_{u+2h-1}$ is unfolding$(p_u, p_{u+2h-1}, 2h)$ defined in Lemma A.2.

The diameter of P'_i is at least ht by Lemma A.2. The points in S'' are the same as those in S' except that $P_{i,1}$ is replaced by the subsequence $P'_{i,1} = p_u, p'_{u+1}, \cdots, p'_{u+2h-2}, p_{u+2h-1}$, the diameter of P'_i is at least $(m + h - 1)t$. Therefore, diameter$(S'') \geq (m + h - 1)t > \frac{mt}{1-\epsilon}$.

The algorithm A has the same output x when its input is S' or S''. Since the diameter of S' is $m \cdot t$, we have $x \leq mt$. The diameter of S'' is greater than $\frac{mt}{1-\epsilon}$. Therefore, x is not a $(1 - \epsilon)$-approximation for the diameter of S''. A contradiction. ∎

Proof: (of Theorem 5.14) Assume that A is an NQ$((n - m)/4)$ computation such that it computes the $(1 - \epsilon)$-approximate diameter for a rearrangement sequence S' of a t-sequence S of diameter at least $m \cdot t$ for some $t > 0$. We are going to construct a counter example for this algorithm.

Let $g = 4\left\lfloor \frac{m}{1-\epsilon} \right\rfloor$ and $n = m + kg$. Let S be the t-sequence of points $q_1, q_2, \cdots, q_m, p_1, p_2, \cdots, p_{kg}$, where $q_i = (i - 1)t$ for $i = 1, 2, \cdots, m$, $p_i = mt$ for odd $i = 1, 3, \cdots$ and $p_i = (m - 1)t$ for even $i = 2, 4, \cdots$. Clearly, S is a t-sequence in R^1 and has diameter $m \cdot t$.

Assume that B is a path of A with input S such that path B gives an output x with $(1 - \epsilon)$diameter$(S) \leq x \leq$ diameter(S). Each query is represented by an integer j, which

expects to receive the j-th point in the input sequence. Assume that $j_1, j_2, \cdots, j_z$ are all the queries in the path B when the input of A is S.

Consider the last g points $p_{(k-1)g+1}p_{(k-1)g+2}\cdots p_{kg}$ in the sequence S. Let unfolding$(p_{(k-1)g+1},\ p_{kg}, g) = p_{(k-1)g+1}u_1u_2\cdots, u_{g-2}, p_{kg}$. Let S' be the sequence S by replacing $p_{(k-1)g+2}p_{(k-1)g+3}\cdots p_{kg-1}$ by $u_1u_2\cdots, u_{g-2}$.

Let S'' be a permutation of S' such that no point in $p_{(k-1)g+1}q_1q_2\cdots, q_{g-2}, p_{kg}$ will be queried in the path B when A has S'' as input and $S''[j_i] = S[j_i]$ for $i = 1, 2, \cdots, z$. Such a permutation S'' from S' exists when m is large enough since $z \leq \frac{n-m}{4}, m = o(n)$ and $g = O(m)$. Therefore, the path B outputs the same result x when the input of A is S''. The diameter of unfolding$(p_{(k-1)g+1}, p_{kg}, g) \geq \frac{g}{2}$ by Lemma A.2. Therefore, diameter$(S'') \geq \frac{g}{2} > \frac{mt}{(1-\epsilon)}$. Thus, $x \geq (1-\epsilon)$diameter(S'') does not hold. A contradiction. ∎

Proof: (of Theorem 5.18) Assume that β and c with $0 < \beta < c$ are two constants, and A is a deterministic algorithm such that given a $\Lambda_{R^1}(1, \beta m, cm, 0, m, n)$-sequence S', it computes a $(1-\epsilon)$-approximate diameter for $E(S')$. We are going to construct a counter example for this algorithm. We will construct two $\Lambda_{R^1}(1, cm, \beta m, 0, m, n)$ sequences S' and S'' such that the algorithm has same output when S' and S'' are the inputs, but the diameters of $E(S')$ and $E(S'')$ have difference more than factor $\frac{1}{1-\epsilon}$. This brings a contradiction.

Let integer m be enough such that

$$1 < \frac{(c-\beta)m}{4}, \text{ and} \tag{A.1}$$

$$\frac{cm}{4} < \left\lfloor \frac{cm}{3} \right\rfloor - 1 \tag{A.2}$$

Let t be an arbitrary real nubmer > 0 (e.g. $t = 1$). Let $h = 2(\left\lceil \frac{\epsilon}{1-\epsilon} \right\rceil + 2)$ and

$$g = 4hm. \tag{A.3}$$

Let S be the t-sequence of points $q_1, q_2, \cdots, q_{m+1}, p_1, p_2, \cdots, p_{n-m}$, where $q_i = (i-1)t$ for $i = 1, 2, \cdots, m+1$, $p_i = (m-1)t$ for odd $i = 1, 3, \cdots$ and $p_i = mt$ for even $i = 2, 4, \cdots$. Clearly, S is a t-sequence in R^1 and has diameter $m \cdot t$. S' will be constructed as a permutation of the sequence of tuples $(S[1], 1)(S[2], 2)\cdots(S[n], n)$.

Let $n = m + 1 + kg$. We have $k = \Omega(\frac{n}{m})$ since $m = o(n)$ and $g = O(m)$. Partition the points $p_1p_2\cdots p_{n-m}$ evenly into PQ, where $|P| = |Q| = kg/2$ (both P and Q contain $\frac{kg}{2}$ points). The sequence S has three regions Q_0, P, and Q, where $Q_0 = q_1q_2\cdots q_{m+1}$.

Region P is partitioned into $P_1P_2\cdots P_{k/2}$ with $|P_i| = g$ for $i = 1, \cdots, k/2$. The region P has another partition into $P'_1\cdots P'_s$ such that $\left\lfloor \frac{cm}{3} \right\rfloor - 1 \leq |P'_i| \leq \left\lfloor \frac{cm}{3} \right\rfloor$ for $i = 1, 2, \cdots, s$. Let $P'_i = S[u_i, v_i]$ for $i = 1, 2, \cdots, s$ and $Q = S[u, v]$. Let

$$r = \left\lfloor \frac{cm}{3} \right\rfloor - 1. \tag{A.4}$$

Let w be the least integer such that

$$\frac{1}{w} < \frac{(c-\beta)}{4c}. \tag{A.5}$$

Let Q' be a sequence of length n such that $Q'[j] = (S[j], j)$ if $j = 0(\text{mod } w)$ and $j \in [u, v]$, and $Q'[j] = (*, *)$ otherwise. An important target of the construction is to make S' such that for some P_i, no point (p, k) with $p \in P_i$ is queried by the algorithm with S' as input. This can make S'', which is derived from S' by adjusting some tuples, have large diameter when the points in P_i are unfolded according to Lemma A.2, contradicting our assumption that the algorithm outputs a $(1 - \epsilon)$ approximation to the diameter of $E(S')$ as well as a $(1 - \epsilon)$ approximation to the diameter of $E(S'')$. We use $(*, *)$ in Q' to mark the tuple $Q'[i]$ not available to swap.

In the following construction, we will make sequence S' that consists of n tuples (p, a), where p is a point in R^1 and $a \in \{1, 2, \cdots, n\}$. We use $S'[i]$ to represent the i-th tuple in S'. Initially we let $S'[i] = (S[i], i)$ for $1 \leq i \leq n$. Then we let $S'[i] = (*, i)$ for $i \in [u, v]$ and $i = 0(\text{mod}w)$. The positions i with $S'[i] = (*, i)$ represent those elements that may be moved to other regions and we put the star "*" mark here. We will change the tuples in S' in the construction below.

Each query is represented by an integer j, which requests to fetch the j-th tuple in the input sequence. Assume that the z-th query is j_z. Let $Query_z(S', a, b)$ be the set of all queries $j \in [a, b]$ among the first z queries when the input is S'. Let T be a sequence of n tuples and $T[i] = (*, *)$ for $i = 1, 2, \cdots, n$. Let

$$y = \frac{(c - \beta)g}{64h}. \tag{A.6}$$

Stage z

Case 1: $j_z \in [u_i, v_i]$ and $|Query_z(S', u_i, v_i)| \leq y$.

- Subcase 1.1: $S'[j_z] = (p, j_z)$ for some p. Let j be the first position in Q' such that $Q'[j] \neq (*, *)$. Let $T[j_z] = S'[j_z]$, $S'[j_z] = Q'[j]$, and $Q'[j] = (*, *)$. Reply to the query with answer $S'[j_z]$, which is just updated.

- Subcase 1.2: $S'[j_z] = (q, j)$ for some q and $j \neq j_z$. By our construction, $S'[j_z]$ has been updated by the subcase 1.1 some time before. We will not change the content of $S'[j_z]$. Reply to the query with answer $S'[j_z]$.

Case 2: $j_z \in [u_i, v_i]$ and $|Query_z(S', u_i, v_i)| > y$. The region $[u_i, v_i]$ has been queried more than y times. We do not do any swap between the elements in this region and the elements in region Q. Reply to the query with answer $S'[j_z]$.

Case 3: $j_z \in [u, v]$.

- Subcase 3.1: $S'[j_z] = (*, j_z)$. Let j be the first position in Q' such that $Q'[j] \neq (*, *)$. Let $S'[j_z] = Q'[j]$, and $Q'[j] = (*, *)$. Reply to the query with answer $S'[j_z]$.

- Subcase 3.2: $S'[j_z] \neq (*, j_z)$. Reply to the query with answer $S'[j_z]$.

End of Stage z

Stage $z_0 + 1$ is the final stage of construction. It assigns tuples to those locations of S' where contain tuples with '*'.

Stage z_0+1(z_0 is the total number of queries)
For each $S'[j]=(*,j)$,
if there is a position k with $Q'[k]\neq(*,*)$
then let $S'[j]=Q'[k]$ and $Q'[k]=(*,*)$.
else find a position k with $T[k]\neq(*,*)$, then let $S'[j]=T[k]$ and $T[k]=(*,*)$.
End of Stage z_0+1

The following fact is easy to verify from the construction. It means that each region $[u_i,v_i]$ has got at most y swaps.

Fact: For every $[u_i,v_i]$ region for P_i, there are at most y indices $j\in[u_i,v_i]$ such that $S'[j]\neq(s,j)$ for any s.

As the algorithm makes $o(n)$ queries, there exists an integer i such that no point (p,j) with $p\in P_i$ is replied to the algorithm among all queries. Otherwise, each P_i intersects a P'_j that contains at least y elements replaced. Since the lengths of both P'_j and P_i are $\Omega(m)$, each P'_j can intersect at most $x=\Theta(1)$ P_is and each P_i can intersect at most $x'=\Theta(1)$ P'_js. The total number of queries is at least $y\frac{k}{x}=\Omega(n)$ since $x=\Theta(1), y=\Omega(m)$, and $k=\Omega(\frac{n}{m})$. It is a contradiction of the fact that the number of queries is $o(n)$.

Assume that there exists $[u_i,v_i]$ such that no point (p,j) with $j\in[u_i,v_i]$ is replied to the algorithm among all queries. Assume that $S[u_i,v_i]$ has the sequence of points $(s_{u_i},u_i),(s_{u_i+1},u_i+1),\cdots,(s_{v_i},v_i)$.

We replace

$$(s_{u_i},u_i),(s_{u_i+1},u_i+1),\cdots,(s_{v_i-1},v_i-1),(s_{v_i},v_i)$$

by

$$(s_{u_i},u_i),(s'_{u_i+1},u_i+1),\cdots,(s'_{v_i-1},v_i-1),(s_{v_i},v_i),$$

where

$$\text{unfolding}(s_{u_i},s_{v_i},g)=s_{u_i},s'_{u_i+1}\cdots,s'_{v_i-1}s_{v_i}.$$

Let S'' be converted from S' such that let $S''=S'$, and then if $S'[j]=(p,t)$ for some $t\in(u_i,v_i)$, then let $S''[j]=(s'_t,t)$. We now verify that S'' is a $\Lambda_{R^1}(1,cm,\beta m,0,m,n)$-sequence.

In the Q region, we only move points after every w consecutive points. For every x consecutive points in the region Q, there are at most $\frac{x}{w}$ points that are moved. Assume that we have cm consecutive points such that x_1 of them are in the P region, and x_2 of them are in the Q region. Each P_i contains at least r points. The number of moved points among the x_1 points is at most $\left\lceil\frac{x_1}{r}\right\rceil\cdot y$. The number of moved points among the x_2 points in the Q region is at most $\left\lceil\frac{x_2}{w}\right\rceil$. By equations (A.2), (A.3), (A.4), (A.6), and (A.5), we have at most

$$\begin{aligned}\left\lceil\frac{x_1}{r}\right\rceil\cdot y+\left\lceil\frac{x_2}{w}\right\rceil &\leq \left\lceil\frac{x_1}{\frac{cm}{4}}\right\rceil\cdot\frac{c-\beta}{64h}\cdot g+\frac{c-\beta}{4c}\cdot x_2+1 &\text{(A.7)}\\ &\leq \frac{2cm}{\frac{cm}{4}}\cdot\frac{c-\beta}{64h}\cdot 4h+\frac{c-\beta}{4c}\cdot cm+1 &\text{(A.8)}\\ &\leq \frac{3(c-\beta)m}{4}+1. &\text{(A.9)}\end{aligned}$$

$$\leq \quad (c-\beta)m \quad \text{(by inequality (A.1))}. \tag{A.10}$$

points among cm points moved. Therefore, at least βm of them remain still.

The algorithm A has the same result for the inputs S' and S''. The diameter of $E(S')$ is $m \cdot t$ and the diameter of $E(S'')$ is at least $(m+h) \cdot t > \frac{1}{1-\epsilon} m \cdot t$. Therefore, A is not a $(1-\epsilon)$-approximation algorithm. ∎

Proof: (of Theorem 5.25) We will assume the existence of an approximate algorithm with small number of queries. A contradiction will be derived by this assumption. We will use the fact that when the number of queries is small, there are a large number of consecutive points in the input sequence not queried by an algorithm. If those points are packed tightly, the diameter is small. Those points can also be arranged so that the diameter is large.

Select positive constants δ, c_0 and c_1 such that

$$\sqrt{d} \quad < \quad c\frac{c_1^{d-1}}{4}(1-\delta), \text{ and} \tag{A.11}$$

$$c_0 \quad < \quad \frac{1}{c_1^{d-1}}. \tag{A.12}$$

Assume that algorithm A makes at most $c_0 n^{1-\frac{1}{d}}$ queries to the input points and computes c-approximate diameter for a self-avoiding sequence in R^d. We consider an integer m and an integer

$$k = \left\lceil c_1 m^{\frac{1}{d-1}} \right\rceil. \tag{A.13}$$

We let k and n depend on m. Let integer m be large enough such that

$$c_1 m^{\frac{1}{d-1}} \geq 10. \tag{A.14}$$

We define a self-avoiding sequence S_1 which is shown in Figure A.1 and another self-avoiding sequence S_2 which is shown in Figure A.2 and has the same length as S_1 ($|S_1| = n$). They are constructed as follows.

We consider two types of cubes. The type 1 cube is of size $(k+10)^d$ grids (each grid is of size 1^d). The type 2 cube is of size m^d type 1 cubes. Consider a type 2 cube B that contains m^d type 1 cubes. When the self-avoiding sequence S_1 is inside the box B, the diameter of S_1 is at most the length of the diagonal of B, which is at most $\sqrt{d}m(k+10)$. We put two colors in the grid points of a self-avoiding sequence. Each type 1 cube contains exactly k^d blue points. The rest points are red for connecting two consecutive cubes. The number of grid points in the self-avoiding sequence is $n = m^d k^d + O(m^d k)$ as the number of red grid points connecting the two consecutive type 1 cubes is $O(m^d k)$.

For the type 1 box that has no points queried, the points in it can be rearranged so that the diameter is at least $\frac{k^d}{2}(1-\delta)$ and still keeps the self-avoiding grid point sequence in R^d. See Figure A.1 and Figure A.2. In Figure A.1, each black box contains k^d grid points in the self-avoiding sequence. In Figure A.2, the k^d grid points in one black box spread from left to right and make the diameter at least $\frac{k^d}{2}(1-\delta)$. The self-avoiding sequence S_2 is constructed like that in Figure A.2 such that there is a black box that has no points queried

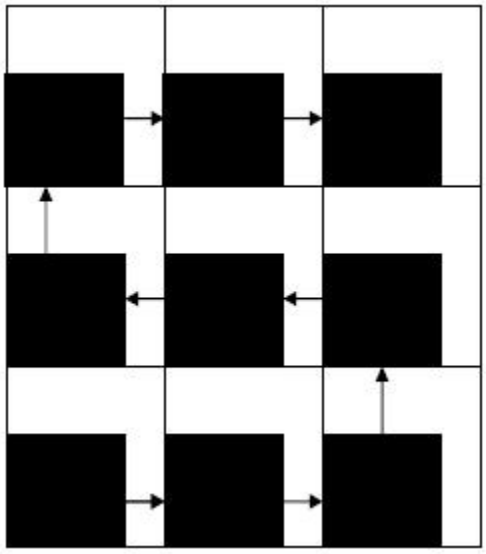


Figure A.1: Every black box has k^d grid points in the sequence S_1

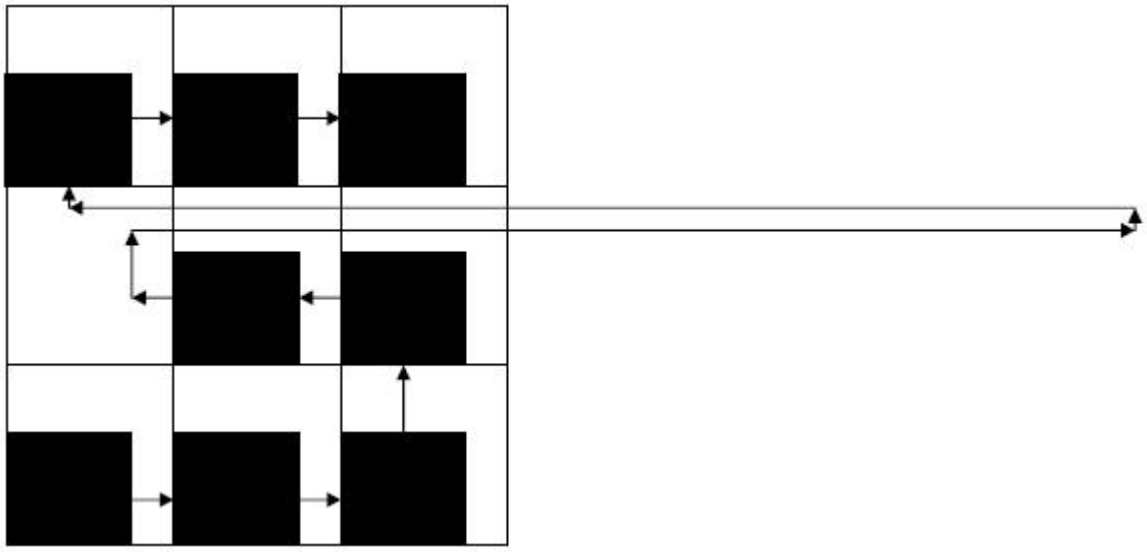


Figure A.2: The points in one black box stretch out to greatly increase the diameter of the sequence S_2

by $A(S_2)$. Then both $A(S_1)$ and $A(S_2)$ will have the same output.

By the setting of m, k and n, we have $n = m^d k^d(1+o(1)) = c_1^d m^{\frac{d^2}{d-1}}(1+o(1))$. Since $n^{\frac{d-1}{d}} = c_1^{d-1}m^d(1+o(1))$, we have $m^d = \frac{1}{c_1^{d-1}} n^{\frac{d-1}{d}}(1-o(1))$. Since $n^{\frac{1}{d}} = c_1 m^{\frac{d}{d-1}}(1+o(1)) = \frac{1}{c_1^{d-1}}(c_1^d m^{\frac{d}{d-1}})(1+o(1)) = \frac{k^d}{c_1^{d-1}}(1+o(1))$, we have $k^d = c_1^{d-1} n^{\frac{1}{d}}(1-o(1))$. Since the algorithm A with input S_1 makes only $c_0(n^{1-\frac{1}{d}})$ queries, there are $m^d = \frac{1}{c_1^{d-1}} n^{\frac{d-1}{d}}(1-o(1)) > c_0(n^{1-\frac{1}{d}})$ type 1 cubes. There exists at least one type 1 cube that has no points queried by the algorithm. The self-avoiding sequence S_2 has consecutive points that are in a black box of S_1 and are not queried by $A(S_2)$. Those points will make the diameter large, as shown in Figure A.2. The result is that $A(S_1)$ and $A(S_2)$ produce the same output, yet there is a large difference between the diameters of S_1 and S_2.
By (A.13), we have $k \geq c_1 m^{\frac{1}{d-1}}$. Therefore,

$$m \leq (\frac{k}{c_1})^{d-1}. \tag{A.15}$$

By inequality (A.15), we have the inequalities

$$\begin{aligned}
\sqrt{d}m(k+10) &\leq \sqrt{d}(\frac{k}{c_1})^{d-1}(k+10) && \text{(A.16)}\\
&\leq c\frac{c_1^{d-1}}{4}(1-\delta)\cdot(\frac{k}{c_1})^{d-1}(k+10) && \text{(A.17)}\\
&< c\cdot\frac{k^{d-1}(k+10)}{4}(1-\delta) && \text{(A.18)}\\
&< c\cdot\frac{k^{d-1}\cdot 2k}{4}(1-\delta) && \text{(A.19)}\\
&= c\cdot\frac{k^d}{2}(1-\delta) && \text{(A.20)}
\end{aligned}$$

The transition from (A.18) to (A.19) is due to (A.13) and (A.14), which imply $k > 10$. By (A.16) through (A.20), we have $\sqrt{d}m(k+10) < c\cdot\frac{k^d}{2}(1-\delta)$. Since algorithm A is a c-approximation to the diameter of the self-avoiding sequence, $A(S_1)$ should output an approximate diameter r with $\sqrt{d}m(k+10) \geq r$ (recall that the diameter of S_1 is at most $\sqrt{d}m(k+10)$). This implies $c\frac{k^d}{2}(1-\delta) > r$. We have a contradiction by the definition of a c-approximate algorithm in Definition 5.1. Therefore, there is no algorithm for c-approximation to the diameter of the self-avoiding sequence. ∎

A.4 Some Tables in Chapter 6

Table A.5: Results for $m = 100, n = 20, \beta = 20\%$ and $\alpha_2 = 20\%$

α_1 (%)	Time (ms)	Reconstruction Rate (%)
1	3.460	100.00
2	3.706	100.00
3	3.983	99.99
4	4.188	99.96
5	4.390	99.95
6	4.553	99.90
7	4.697	99.77
8	4.943	99.58
9	5.183	99.39
10	5.412	98.94

Table A.6: Results for $m = 100, \beta = 20\%$ and $\alpha_2 = 20\%$

	$n = 10$		$n = 30$	
α_1 (%)	Time (ms)	Reconstruction Rate (%)	Time (ms)	Reconstruction Rate (%)
1	2.444	99.91	4.744	100.00
2	2.568	99.78	5.046	100.00
3	2.674	99.58	5.261	100.00
4	2.774	99.36	5.605	99.99
5	2.851	99.01	6.045	100.00
6	2.925	98.60	6.302	99.97
7	3.028	98.03	6.567	99.96
8	3.121	97.54	6.870	99.85
9	3.213	96.81	7.307	99.70
10	3.314	95.85	7.635	99.56

Table A.7: Results for $m = 100, n = 20$ and $\alpha_2 = 20\%$

	$\beta = 10\%$		$\beta = 30\%$	
α_1 (%)	Time (ms)	Reconstruction Rate (%)	Time (ms)	Reconstruction Rate (%)
1	3.425	100.00	3.564	100.00
2	3.687	100.00	3.736	100.00
3	3.904	99.90	3.925	99.99
4	4.175	99.83	4.154	99.96
5	4.422	99.52	4.337	99.95
6	4.606	99.25	4.528	99.91
7	4.826	98.68	4.704	99.83
8	4.998	98.14	4.920	99.73
9	5.190	97.69	5.096	99.61
10	5.355	96.90	5.295	99.39

Table A.8: Results for $m = 100, n = 20$ and $\beta = 20\%$

	$\alpha_2 = 10\%$		$\alpha_2 = 30\%$	
α_1 (%)	Time (ms)	Reconstruction Rate (%)	Time (ms)	Reconstruction Rate (%)
1	3.225	100.00	3.575	99.98
2	3.551	99.99	3.792	99.98
3	3.712	100.00	3.990	99.94
4	3.980	99.98	4.184	99.88
5	4.162	99.98	4.369	99.76
6	4.324	99.97	4.592	99.54
7	4.550	99.94	4.761	99.09
8	4.733	99.92	4.968	98.52
9	4.911	99.82	5.191	97.70
10	5.116	99.72	5.401	96.75

Zhiyu Zhao

From: hra@cs.uga.edu
Sent: Friday, May 09, 2008 5:03 PM
To: silva.zhao@gmail.com
Subject: Re: Permission Request

Categories: Green Category

Dear soon-to-be Dr. Zhiyu Zhao:

Thank you for your kind email.

You wrote:
> Dear Professor Arabnia,
>
> My name is Zhiyu Zhao. I am a graduate student at the University of
> New Orleans. As an author of a paper published and another one which
> is going to be published by the CSREA Press, I would like to obtain
> the company's permission for including the content of the following
> papers in my PhD dissertation entitled "Algorithms for Comparing
> Protein Structure and Genome
> Similarities":
>
> (1) "A Flexible Algorithm for Pairwise Protein Structure
> Alignment" in Proceedings of BIOCOMP'07.
>
> (2) "New Algorithm and Web Server for Finding Proteins with
> Similar 3D Structures" accepted by BIOCOMP'08.
>
> Would you please kindly let me know if I am permitted to do so? Thank
> you very much in advance.

As the editor, general chair, and coordinator of the BIOCOMP'07 and BIOCOMP'08 conferences as well as an elected officer of CSREA Press, you have my permission to include the above mentioned papers in your PhD dissertation.

I wish you best with your PhD.

With kindest regards,
Hamid

Hamid R. Arabnia, PhD.
Professor, Computer Science
Chair, WORLDCOMP'08 (http://www.world-academy-of-science.org/) Editor-in-Chief, The Journal of Supercomputing (Springer) Member, Advisory Board, IEEE Technical Committee on Scalable Computing (TCSC) Director, Graduate Programs The University of Georgia Department of Computer Science
415 Graduate Studies Research Center
Athens, Georgia 30602-7404, USA

Tel: (706) 542-3480
Fax: (706) 542-2966
email: hra@cs.uga.edu
url: http://www.cs.uga.edu/~hra/

Dear Zhiyu

Thanks for the info. I am pleased to grant you the permission provided that you acknowledge the original source properly.

With regards,
Tu Ning

Sylvia Zhao wrote:
> Dear Ning,
>
> The title of my dissertation is "Algorithms for Comparing Protein
> Structure and Genome Similarities". If you need any more information
> please just let me know. Thank you!
>
> Regards,
> Zhiyu
>
> -----Original Message-----
> From: Tu Ning [mailto:ntu@wspc.com]
> Sent: Wednesday, May 07, 2008 8:05 PM
> To: silva.zhao@gmail.com
> Subject: Re: Fw: Permission Request
>
> Dear Dr. Zhao
>
> Kindly let me know the full title of your Ph.D dissertation in order
> to process your permission request.
>
> With regards
> Tu Ning
>
>> *---------- Forwarded Message ----------*
>> From: "Sylvia Zhao" <silva.zhao@gmail.com>
>> To: <edit@icpress.co.uk>
>> Sent: Tue, 6 May 2008 23:12:11 -0600
>> Subject: Permission Request
>>
>> Dear Sir/Madam,
>>
>> My name is Zhiyu Zhao. I am a graduate student at the University of
>> New Orleans, USA. As an author of a paper published by your company,
>> I would like to obtain the permission from your company for including
>> the content of the following paper in my PhD dissertation:
>>
>> "Linear Time Probabilistic Algorithms for theSingular Haplotype

From: Werf van der, Nel, Springer SEM NL [mailto:Nel.vanderWerf@springer.com] On Behalf Of Permissions Europe/NL
Sent: Thursday, July 31, 2008 7:24 AM
To: Zhiyu Zhao
Subject: RE: Permission request (Sylvia Zhao)

Dear Zhiyu Zhao,

Thank you for your email.

We think that 100 copies to defend your thesis will be enough.
The number of copies for ProQuest is not included in this number, 100 copies is only for you to defend your thesis.

We give you permission to post your dissertation on the e-library of your university.

Best regards,

Nel van der Werf (Ms)
Rights and Permissions/Springer

Van Godewijckstraat 30 | P.O. Box 17
3300 AA Dordrecht | The Netherlands
tel +31 (0) 78 6576 298

fax +31 (0)78 65 76-300
Nel.vanderwerf @springer.com
www.springeronline.com

Vita

Zhiyu Zhao received her B.E. (1997) and M.E. (2000) degrees in Computer Engineering from the Huazhong University of Science and Technology, China. In the year 2004, with a governmental scholarship, she initialized her Bioinformatics research in DNA micro-array data analysis at the Politecnico di Milano University, Italy. In the year 2005, she started her Ph.D. study in the Department of Computer Science at the University of New Orleans.

She received a M.S. degree in Computer Science in 2006. After the completion of her Ph.D. study, she will be working in the College of Sciences at the University of New Orleans as a Computational Scientist.