QUERY AND MINING IN BIOLOGICAL DATABASES

TAN ZHENQIANG

NATIONAL UNIVERSITY OF SINGAPORE

2006

QUERY AND MINING IN BIOLOGICAL DATABASES

TAN ZHENQIANG

MASTER OF COMPUTER SCIENCE
WUHAN UNIVERSITY, CHINA

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

SCHOOL OF COMPUTING

NATIONAL UNIVERSITY OF SINGAPORE

2006

Acknowledgement

I owe my thanks for contributions to this thesis to many persons. First of all, I would like to thank my Ph.D. advisor, Professor Anthony K.H. Tung, for his many suggestions and support during this research. He has taught me how to establish valuable research directions and how to constantly move forward towards the target. The training that I have received from him is the most valuable thing during the days in National University of Singapore. I have learned a lot from him about the way to conduct qualified research. This thesis is the result of his inspiring and thoughtful guidance and supervision. I would like also to thank Professor Ooi Beng Chin and Professor Kian-Lee Tan for their valuable suggestions. I am highly indebted to Ms. Cao Xia and Mr. Zeyar Aung for sharing their knowledge and experience in computational biology with me. I am grateful to Mr. Chen Jin and Mr. Liu Tiefei for their very helpful ideas and discussions. I also thank Ms. Xia Chenyi and Mr. Jing Qiang for their help and support. Many thanks are due to Dr. Cui Bin and Dr. Ng Wee Siong for their assistances. Many thanks go to School of Computing, National University of Singapore, for accepting me to carry out substantial work with the facilities. Thanks are also due to the management of School of Computing, Ms. Loo Line Fong and Mr. Tan Poh Suan. Finally, I would like to thank my parents and my wife for their patience and love. Without their support, this work would never have come into existence.

Zhenqiang Tan

Jan 12, 2006

CONTENTS

Acknowledgement **iii**

Summary **xvi**

1 Introduction **1**

1.1 DNA Sequences And Proteins 2

1.1.1 DNA Sequences 2

1.1.2 From DNA Sequences to Proteins 3

1.1.3 Amino Acid Sequences And Protein Structures 4

1.1.4 Our Study on Computational Approaches to DNA Sequences and Proteins 5

1.2 Database Techniques for Biological Datasets 6

1.3 Homology Search in DNA Sequences 7

1.3.1 Motivations 8

1.3.2 Our Research Problem 9

1.3.3 Contributions: The *ed-tree* 10

1.4 Mining Sequential 3D Patterns in Protein Structures 10
1.4.1 Motivations 10
1.4.2 Our Research Problem 11
1.4.3 Contributions: sCluster And MSP 12
1.5 Remote Homology Detection Based on Sequential 3D Patterns . . . 13
1.5.1 Motivations 14
1.5.2 Our Research Problem: Protein Classification Based on 3D Structures 14
1.5.3 Our Research Problem: Finding Coding DNA Regions for Similar 3D Protein Structures 14
1.5.4 Contributions: Deterministic Binary Classification Tree . . . 15
1.5.5 Contribution: FCDR System 15
1.6 Outline of This Thesis 15

2 State of Arts **17**
2.1 Homology Search in DNA Sequence Datasets 17
2.1.1 Sequential-scan-based Approaches 17
2.1.2 Suffix Tree Based Approaches 22
2.1.3 Index-based Approaches 25
2.2 Subspace Clustering And Pattern Mining 28
2.2.1 Subspace Clustering 28
2.2.2 Graph Pattern Mining 35
2.3 Remote Homology Detection 38

3 Homology Search in Large DNA Sequence Datasets **44**
3.1 Introduction 44
3.2 The *ed-tree* 50

3.2.1 Definitions 50

3.2.2 Algorithm to Build The *ed-tree* 52

3.3 Homology Search with The *ed-tree* 53

3.3.1 Theories 54

3.3.2 The Algorithm - *Probe Search* 58

3.3.3 Analysis And Experimental Evaluation of Pruning Effect . . 61

3.3.4 Detecting Proper Setting 62

3.4 Performance Study 64

3.4.1 Datasets 64

3.4.2 Comparing The *ed-tree* with *Blastn* 65

3.4.3 Pruning Cost Analysis 67

3.4.4 Effect of Parameters 68

3.5 Summary 70

4 Substructure Clustering in Sequential 3D Object Datasets 72

4.1 Introduction 72

4.2 Definition And theory 74

4.2.1 Sequential 3D object 74

4.2.2 Similarity Evaluation 75

4.2.3 *sCluster* 78

4.3 Algorithms 83

4.3.1 Mining Pairwise *Maximal sCluster* 83

4.3.2 Query Related *sClusters* 88

4.4 Experiments 90

4.4.1 Effect of Parameters 91

4.4.2 Query *Maximal sCluster*s Related to New Object 93

4.4.3 Mining *sCluster*s in Synthetic Datasets 94

4.4.4 Comparison with *rmsd*-based Clustering 95
4.4.5 Results of *sCluster* 96
4.4.6 Application in HIV Protein 3D Structures 99
4.5 Summary 101

5 Mining 3D Sequential Patterns With Constraints 103

5.1 Introduction 103
5.2 Definitions 105
5.2.1 Pattern And Hit 106
5.3 Algorithm 107
5.3.1 Generating Seeds: Pairwise Pattern Mining 111
5.3.2 Vertical Extension: Depth-first Search to Detect Hits 111
5.3.3 Horizontal Extension: Extend Pattern Length without Loss of Hits 115
5.3.4 Detection of Proper Settings 117
5.4 Experiments 120
5.4.1 Parameters 121
5.4.2 Comparing MSP with sCluster 126
5.5 The Applications of MSP 129
5.5.1 MSP for Binary Classification in Protein Structures 129
5.5.2 MSP for PhysioNet/CinC Challenge 2002 Dataset 131
5.6 Summary 133

6 Remotely Homology Detection Based on Protein 3D Structures 134

6.1 Introduction 134
6.2 Preliminary 136
6.2.1 Definitions 136

6.2.2 Mining Motifs with Gaps 138

6.2.3 Mining Motifs as Specified 141

6.3 Binary Classification Rule Group 142

6.4 Binary Classification Tree 144

6.4.1 Family Structural Difference 145

6.4.2 Deterministic Binary Classification Tree 145

6.5 Experiments 148

6.5.1 Dataset 149

6.5.2 Accuracy of Binary Classifier 149

6.5.3 Confidence 151

6.5.4 Precision And Recall 152

6.6 Summary 153

7 FCDR: Finding Coding DNA Regions for Similar 3D Protein Structures 155

7.1 Introduction 155

7.2 Problem Description 156

7.3 System Architecture 156

7.3.1 Translate DNA to Protein Sequence 157

7.3.2 Build $ed - tree$ on Protein Sequences 158

7.3.3 DPS & sCluster to Mine Similar 3D Protein Structures . . . 159

7.3.4 Search Coding DNA Regions for 3D Protein Structures . . . 160

7.4 Experiments 161

7.4.1 Datasets 161

7.4.2 Preprocessing on DNA Sequence Dataset 161

7.4.3 Preprocessing on Protein 3D Structure Dataset 162

7.4.4 Visualization And Query 162

7.5 Summary 164

8 Conclusions **165**

8.1 Thesis Findings 165

8.2 Future Works 168

LIST OF FIGURES

1.1 DNA dual-helix structure 2
1.2 From DNA to protein 3
1.3 Architecture of amino acid 4
1.4 Connection between two amino acids 4
1.5 Task classification 8
1.6 Growth of DNA sequences in GenBank 9
1.7 Example of DNA similarity search 9
1.8 Example of subspace clustering 11

2.1 Word tables in Blastn and SENSEI 21
2.2 Lemma of QUASAR 23
2.3 Shift pattern and scaling pattern in *pCluster* 30
2.4 The architecture of AnMol 32
2.5 Example of sequence patterns with noise 33
2.6 Sample result of common structures 33
2.7 The architecture of GraphMiner 37

3.1 Sensitivity(64 bps) 48
3.2 Sensitivity(128 bps) 48
3.3 An example of *ed-tree* 52
3.4 Building an *ed-tree* 53
3.5 The 3-level *ed-tree* index 54
3.6 Cardinality of Cover Generator 57
3.7 Segmenting P=GGTAGCGGCTTACTTCAG 58
3.8 Homology search in *ed-tree*(w, s, H) 59
3.9 Processing for the example in step 4 60
3.10 Pruning Rate 61
3.11 *ed-tree* Index Sizes, $w = 18$ 65
3.12 Speed vs DB Size (Query length=250) 66
3.13 DB:est_human 1.55Gbps 67
3.14 DB:est_other 2.07Gbps 67
3.15 Level 1,2 Pruning time vs DB Size 68
3.16 Level 3 Pruning time vs DB Size 69
3.17 Level 1,2 Pruning time vs Query Length 70
3.18 Level 3 Pruning time vs Query Length 70

4.1 Example of sequential 3D objects 74
4.2 Features on $S[i]$: $l[i]$, $a[i]$ and $t[i]$ 76
4.3 Comparison of fds, ald and $rmsd$ in D_1 77
4.4 Comparison of fds, ald and $rmsd$ in D_2 78
4.5 Example of *maximal sCluster* 81
4.6 Sample of Lemma 4.2.1 82
4.7 Example of pairwise *maximal sCluster*s 84
4.8 Example of Algorithm 4.3.1 85

4.9 Example of Algorithm 4.3.2 89
4.10 Object length VS. Clustering time 91
4.11 Number of objects VS. Clustering time 91
4.12 ε VS. Clustering time 92
4.13 w VS. Clustering time 92
4.14 Object length VS. Query response time 94
4.15 Number of objects VS. Query response time 94
4.16 Object length VS. Clustering time on synthetic datasets 95
4.17 Number of objects VS. Clustering time on synthetic datasets 95
4.18 $sCluster$ VS $rmsd - based\ clustering$ on object length 96
4.19 $sCluster$ VS $rmsd - based\ clustering$ on number of objects 97
4.20 Cardinality VS. Number of $sCluster$s in 5 cases 97
4.21 $d1mma\_2[150 : 182]$ 98
4.22 $d1d0xa2[151 : 183]$ 98
4.23 $d1d1aa2[157 : 189]$ 98
4.24 $d1d1ca2[159 : 191]$ 98
4.25 $d1b71a1[7 : 61]$ 98
4.26 $d1bcfa\_[6 : 60]$ 98
4.27 $d1euma\_\_[2 : 56]$ 98
4.28 $d1jgca\_\_[5 : 59]$ 98
4.29 $d1c0ua1[431 : 470]$ 100
4.30 $d1c1ca1[431 : 470]$ 100
4.31 $d1jlga1[431 : 470]$ 100
4.32 $d1rt1a1[431 : 470]$ 100
4.33 $d1hiia\_[1 : 40]$ 101
4.34 $d1idaa\_[1 : 40]$ 101

4.35 $d1idbb\_[1:40]$ 101

4.36 $d1idab\_[1:40]$ 101

5.1 Framework of MSP 108

5.2 Example of vertical extension 108

5.3 MSP Algorithm 110

5.4 Example of horizontal extension 115

5.5 DPS Algorithm 119

5.6 Example of DPS Algorithm 120

5.7 Number of objects and ε VS. Processing time 121

5.8 Number of objects and object length VS. Processing time 122

5.9 Object length and ε VS. Processing time 122

5.10 Object length and number of objects VS. Processing time 123

5.11 Seed length and number of objects VS. Processing time 123

5.12 ε and number of objects VS. Processing time 123

5.13 min_sup and number of objects VS. Processing time 125

5.14 min_conf and number of objects VS. Processing time 125

5.15 Number of patterns VS. Number of hits 126

5.16 MSP VS. sCluster on number of objects 126

5.17 MSP VS. sCluster on ε 127

5.18 MSP VS. sCluster on object length 128

5.19 MSP VS. sCluster on number of objects in synthetic data 128

5.20 MSP VS. sCluster on object length in synthetic data 128

5.21 Sample pattern - 1: $\{d1b71a1[7:61], d1bcfa\_[6:60], d1euma\_[2:56], d1jgca[5:59]\}$ 131

5.22 Sample pattern - 2: $\{d1bmr\_[2:26], d1cn2\_[3:27], d1i6ga\_[2:26], d1nrb[1:25]\}$ 131

6.1 Sample motif: $\{(R[4:7], P[3:6], Q[2:5])\}$ 137

6.2 Example of hits: $\{(m_1, P[2:7]), (m_2, P[10:15])\}$ 138

6.3 Left-hand extension on pairwise motifs 140

6.4 Sample motif: $\{d1dm2a[141:170], d1ckpa[144:173], d1b38a[157:186], d1aq11\_[144:173]\}$ 141

6.5 Create $BCRG$s 144

6.6 $DBCT(\{C_1, C_2, C_3, C_4, C_5\})$ 146

6.7 Create $DBCT$ 148

7.1 Architecture of FCDR System 156

7.2 Main interface of FCDR System 157

7.3 Interface of building $ed-tree$ on proteins 158

7.4 Interface of mining protein 3D patterns 159

7.5 Interface of searching DNA sequences for protein 3D structures . . . 160

7.6 Sample pattern in FCDR System 163

Summary

In the last decade, biologists experienced a fundamental revolution from traditional researches involving DNA sequence search and protein structure pattern mining. The biological data is complex, and both the quantity and the size are growing exponentially. Data evolves more quickly than the technologies developed to interpret the data. This motivated us to conduct researches on the query and mining in biological databases. The DNA sequence and the protein structure are the two types of the most important biological data. The former can be represented by strings of four characters and the later can be represented by a sequential 3D structure together with the amino acid sequence information. In this thesis, we focused on the problems raised in these two types of sequential biological data.

First, we studied the index and similarity search in large DNA sequence databases on desktop PC. We proposed an index structure called the *ed-tree* [82] for supporting fast and effective homology searches on DNA databases. The *ed-tree* is a probe-based homology search algorithm similar with the popular Blastn [7] which generates short probe strings from the query sequence and matches them against the sequence database in order to identify the potential regions of high similarity

to the query sequence. Compared to Blastn, *ed-tree* adopts more flexible probe detection model which allows insertion, deletion and replacement. Meanwhile, the query speed on large DNA sequence datasets is significantly enhanced by a factor of 3 to 6. Moreover, the index size of *ed-tree* is modest. For example, the index for a dataset of 2Gbps is about 3GB which is much smaller than the other index strategies such as suffix tree and etc.

Second, we investigated substructure clustering in sequential 3D object datasets, especially protein structures. This problem was not well studied but applicable in many important applications such as protein 3D structure pattern mining, track mining on moving objects and so on. We presented a distance measurement, $Feature\ Difference\ Summation$ (fds), for evaluating the dissimilarity of two sequential 3D structures. The fds is effective on protein structure comparisons but more efficient compared to the traditional structural distance measurement, *Root Mean Square Distance* (*rmsd*). Mining *maximal sCluster*s was described for modelling the problem of finding non-trivial substructure cluster where every two substructures are similar and the cluster cannot be further extended in terms of both the cardinality of cluster and the length of substructures. We proposed sCluster algorithm [83, 85], a modified-apriori approach for efficiently mining *maximal sCluster*s on given sequential 3D object datasets. Additionally, we extend the algorithm to query *maximal sCluster*s which are related to given new objects. Experiments show that our approach significantly outperforms the alternative algorithm and the sample result on protein chains shows the effectiveness.

Third, as an improvement of sCluster, MSP [86] was designed for mining maximal sequential 3D patterns with the constraints of minimum support and mining confidence based on a seed-and-extension strategy. MSP includes three stages, generating pairwise patterns as seeds, vertical extension to detect all the hits with a depth-first search and horizontal extension to extend the pattern length without loss of hits. In order to adapt MSP to various datasets, we created a method to automatically detect proper settings according to the given dataset. The experiments on protein chains and synthetic data show MSP significantly outperforms the sCluster method.

Fourth, we utilized protein 3D structure patterns as the features in classifications for remotely homologous proteins where the similarities of their amino acid sequences to known proteins are ambiguous. Without considering sequences, sCluster were adopted to find structural motifs for building binary classification rule groups. Deterministic Binary Classification Tree ($DBCT$) [84] was proposed to incorporate multiple binary classifiers to multi-class classification. $DBCT$ avoids the tremendous number of binary classifiers. Experimental study shows both the precision and the recall of our approach are high, and $DBCT$ exponentially enhances the response speed of protein family prediction.

Furthermore, we applied $ed-tree$ on protein sequences and built a FCDR System to search DNA regions which code conserved 3D protein structures mined by sCluster. A well-designed GUI was provided for researchers to view 3D protein structures and to query the coding DNA regions. The hit protein sequence and the corresponding DNA coding sequence, annotation, position, translation open reading frames and directions would be described in the query results. It is a comprehensive and intuitive tool to understand the relationship between DNA sequences and conserved protein 3D structures.

In all, we have addressed some important and valuable issues about sequential biological data including DNA sequences and protein chains and proposed our solutions in this thesis. The *ed-tree* could be applied for similarity search in large DNA sequence databases on desktop PC. sCluster and MSP are two generic approaches

for mining sequential structural patterns with respect to 3D coordinates. Both the problem and the approaches are new compared to the existing works. sCluster and MSP could be adopted to find the frequent 3D patterns in proteins. The obtained 3D patterns are further used for classifications in remotely homologous proteins with the $DBCT$ mechanism. Finally, FCDR System integrates $ed - tree$ on protein sequences with sCluster to find coding DNA regions for conserved protein 3D structures.

CHAPTER 1

Introduction

With the development of molecular biology in the last decades, both the volume and the complexity of biological data is growing exponentially. Classical approaches and standard relational database systems are not efficient to produce effective information. To understand and conduct analysis on the data and the correlations between them, computational biological methods are required.

DNA sequences and protein structures (mainly protein chains) are two types of the most important biological data. They are sequential objects which can be represented as strings of characters and sequential 3D structures respectively. In this thesis, we mainly investigated several important issues on DNA sequences and protein structures.

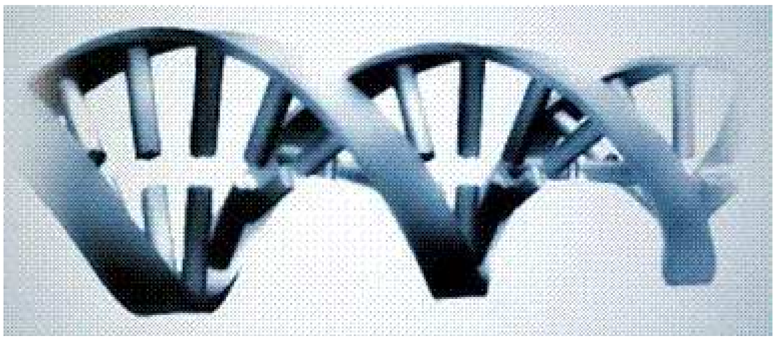


Figure 1.1: DNA dual-helix structure

1.1 DNA Sequences And Proteins

The DNA-protein system is a simple but extremely powerful system for creating all biological features and structures. By varying the code words of DNA sequences, innumerable different proteins with disparate functions are generated. The proteins are consequently incorporated together to build all biological organisms [73].

1.1.1 DNA Sequences

The structure, type and functions of a cell are all determined by chromosomes which are composed of DNA. As shown in Figure 1.1, DNA sequence is arranged into a double-helix structure where the spirals are intertwined with one another continuously bending in on itself and nucleic acids are the building blocks [51]. There are four different nucleic acids, adenine (A), thymine (T), guanine (G), and cytosine (C). The number of nucleic acids in genome is normally very large. For example, a yeast has 12 million and the human genome is made of roughly three billion of nucleic acids. The genome is like a library of instructions that provide the instructions for a single protein component of an organism. Billions of nucleic acids and the variations of permutations result in the uniqueness of the individuals.

1.1.2 From DNA Sequences to Proteins

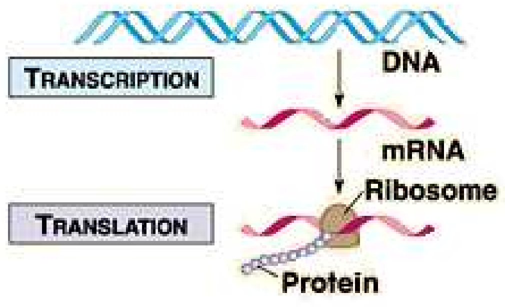


Figure 1.2: From DNA to protein

Each cell contains all the DNA sequences. However, its functions and structures are composed according to the fractions of the DNA sequences which are used. Proteins are essential to our body in a variety of ways. They are the results from a series of transformations on the genetic information in DNA sequences.

Figure 1.2 illustrates the processes for transforming DNA sequences to proteins [51]. Transcription is the creation of messenger RNA (mRNA) using the DNA as a template. Translation is the creation of protein in the ribosome. The double helix structure of DNA uncoils in order for messenger ribonucleic acid (mRNA) to replicate the genetic sequence responsible for the coding of a particular protein. At the beginning, mRNA moves in and transcribes the genetic information. Uracil (U) bases in mRNA replace all thymine bases (T) in DNA . When the genetic information responsible for creating substances is available on the mRNA strand, the mRNA moves out from the DNA towards the ribosome. Ribosomes are special cell structure which are the sites for translation. Finally, the synthesis of proteins is done in ribosomes. During the translation, every three nucleic acids in DNA code one amino acid in protein. The human genome makes about 30,000 proteins, each of which contains a few hundred amino acids [72].

1.1.3 Amino Acid Sequences And Protein Structures

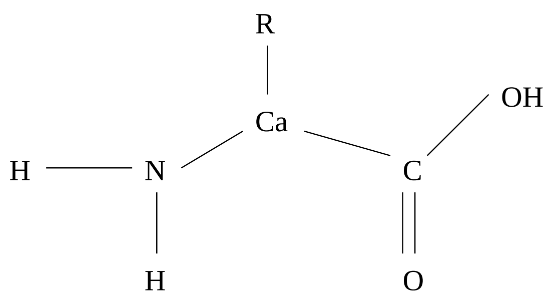


Figure 1.3: Architecture of amino acid

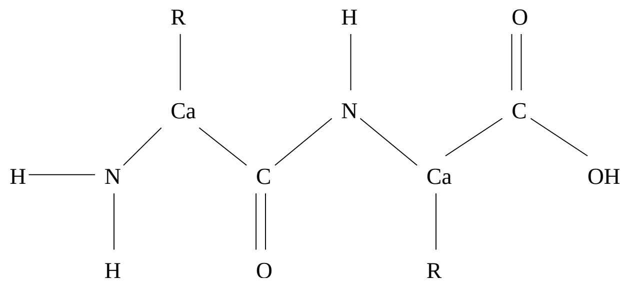


Figure 1.4: Connection between two amino acids

There are twenty amino acids found in proteins. The architecture of an amino acid is depicted in Figure 1.3. R denotes any one of the 20 possible side chains [14]. The different side chains R determine the chemical properties of the amino acid or residue (the residue is the amino acid side chain plus the peptide backbone). The amino acids are encoded using 3-letter code such as ALA (Alanine), LYS (Lysine) and TYR (Tyrosine) and etc. They are combined and connected by the condensation reactions as illustrated in 1.4.

The amino acid sequence is considered as the primary structure of protein. However, the sequence is folded into a complicated 3D structures. Secondary structure is defined as "local" ordered structure brought about via hydrogen bonding mainly within the peptide backbone. Tertiary structure is the "global" folding of a single polypeptide chain. Quaternary structure involves the association of two or more polypeptide chains into a multi-subunit structure [14].

Every protein has either chemical or structural functions to fulfill. It means that the protein functions are determined by the sequence and structure. The protein structure is one of the most important biological data in real-life applications. For example, in pharmaceutics, the protein substructure pattern is extremely valuable for binding site detection which is the basis of the structure-based drug design.

1.1.4 Our Study on Computational Approaches to DNA Sequences and Proteins

During the evolution, the DNA sequence and the protein varied with mutations and natural selection. Consequently, the DNA sequence, the protein sequence and structure are conserved with variations in an extent. To investigate the homology on DNA sequences and protein structures is an important approach to better understanding the evolution.

In this thesis, we firstly studied the homology search in DNA sequences at first. As a result, we proposed the $ed-tree$. Secondly, we discussed the homology mining in protein structures and contributed sCluster [83, 85] and MSP [86]. For proteins which are remotely homologous to the existing annotated protein collection, 3D structures are conserved better than sequences. Therefore, we created the DBCT [84] to apply the structure patterns which are obtained in sCluster and MSP to remote homology detection for proteins. Moreover, we built $FCDR$ System which integrates the visualization of 3D structures and sequence searches in order to further trace DNA regions which code frequent protein 3D patterns.

1.2 Database Techniques for Biological Datasets

Indexing, clustering and mining technology on biological databases are essential to summarize the information of biological data, to efficiently discover knowledge that may be impossible by the traditional methodologies, and to find unexpected patterns which may be meaningful for drug design and some important biological applications such as protein interaction predictions.

A database index is meant to improve the efficiency of data lookup at rows of a table by a key access retrieval method. In practice, large databases must be indexed to meet performance requirements [26]. DNA sequence databases are normally as large as billions of bps (base pairs). For example, the human genome, is about 3Gbps. On the other hand, the DNA sequences are mainly consisted of 4 types of nucleic acids, A (Adeninine), C (Cytosine), G (Guanine) and T (Thymine). Approximate matches are sometimes more important to detect mutation and homology. Special indices [45, 62, 82, 91] are designed according to the characteristics of DNA sequences to address the efficiency and the effectiveness of the results.

Clustering is an unsupervised process to group similar objects together based on the principle of maximizing the intra-class similarity and minimizing the inter-class similarity [23, 32, 34]. Subspace clustering is an extension of traditional clustering that finds clusters in different subspaces within a dataset [67]. Protein chains are sequential 3D objects which comprise linked amino acids ranging from tens to thousands. Subspace clustering on protein chains is to find out frequent 3D motifs which could be very useful.

Classification is a process to find the models or functions to describe and distinguish data classes for the purpose of predicting the class of objects whose class labels are unknown [74]. Nearly all proteins have structural similarities with other proteins and, in some of these cases, share a common evolutionary origin [63].

Many works such as SCOP [9, 25, 63], CATH [66, 68, 69] and Dali [35] and etc. for protein classifications have been contributed to illustrate the structural and evolutionary relationships between the proteins whose structures are known. It could provide a broad survey of all known protein folds, detailed information about the close relatives of any particular protein, and a framework for future research and classification. Extensive researches focused on protein homology detection based on significant or weak sequence similarities [3, 7, 8, 18, 36, 43, 52, 54, 80, 81, 89]. Because protein 3D structure can elucidate its function, in both general and specific terms as well as its evolutionary history [15, 53]. Besides, protein 3D structures in the same family conserve in a more significant extent than sequences. Frequent structural patterns in terms of 3D coordinates could be a new way to facilitate the detection of remote homologies.

Overall, since the biological data becomes tremendous with the growing research interests and the revolution of research approaches, it becomes more and more important and necessary to analyze and understand biological data and the relationships between various data sets using computational approaches.

1.3 Homology Search in DNA Sequences

Homology search on DNA sequences is to find similar local alignments among the query and the sequences in databases according to a similarity scoring system, for example edit distance. It is an important function in genomic research. Different from the previous works, our study in this thesis is to develop a system to enable biologists to build large DNA databases and to conduct fast and effective queries on their own desktop PC.

1.3.1 Motivations

Homology search on DNA sequence databases is an important function in genomic research. R. Stevens et. al. [31] conducted a survey of the tasks in bioinformatics in 2001. These include the tasks of obtaining a sequence, finding what exists that is similar and what patterns are present that might indicate sequence function. The result in Figure 1.5 depicts 35.2% [31] of the tasks are sequence similarity search where 33.3% of them are related to DNA sequence search. Due to the recent interest

Question Class	Percentage(%)
Sequence similarity searching	35.2
Functional motif searching	11.1
Sequence retrieval	8.57
Multiple sequence alignment	6.7
Restriction mapping	6.03
Secondary and tertiary structure prediction	4.4
Other DNA analysis including translation	4.4
Primer design	3.8
ORF analysis	3.5
literature searching	3.17
Phylogenetic analysis	2.86
Protein analysis	3.17
Sequence assembly	2.54
Location of expression	2.22
Miscellaneous	2.22

Figure 1.5: Task classification

in genomic research, the size of DNA sequence databases is growing exponentially in the past few years. For example, the popular GenBank 's nucleotide sequence database is doubling its size every 15-16 months [11, 12] as shown in Figure 1.6. As many existing search methods are based on sequential scanning on databases, the growth in database size will adversely affect the efficiency of these search methods. Due to limited PC memory and the sequential-scan schema of the existing approaches, the query speed on large DNA sequence databases is not satisfied. This motivated us to either develop new and more efficient methods or enhance existing methods to be more scalable to the size of databases. Consequently, we designed the *ed-tree* [82] to speed up the query process on desktop PC.

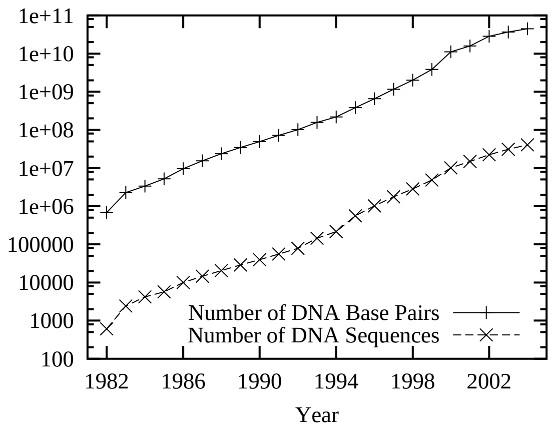


Figure 1.6: Growth of DNA sequences in GenBank

1.3.2 Our Research Problem

Given a query sequence Q and a target sequence database T, find a set of subsequences of Q such that each subsequence Q' in the set is highly similar to some subsequence T' of T. The similarity between Q' and T' is computed as a function of the **edit distance**, $edit(Q', T')$, which is defined as the minimum number of edit operations (insert, delete, replace) that transform Q' into T'.

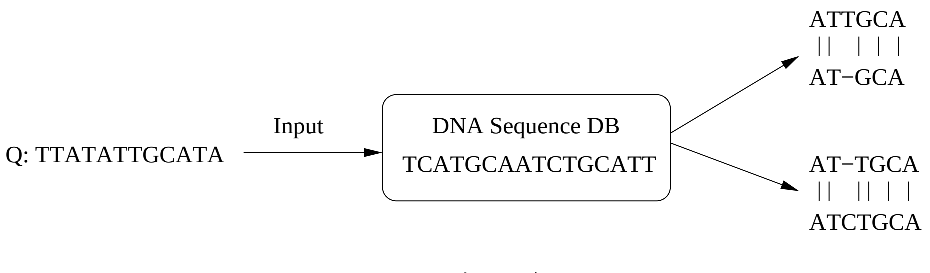


Figure 1.7: Example of DNA similarity search

As shown in Figure 1.7, Q : $ATTGCA$ is a short DNA sequence and we are going to find the similar sequence alignments in the target DNA sequence database $TCATGCAATCTGCATT$. Two pairs are found as below:

$$(ATTGCA, ATGCA) \text{ and } (ATTGCA, ATCTGCA)$$

1.3.3 Contributions: The *ed-tree*

The *ed-tree* is an index structure specially designed for DNA sequences which mainly include four kinds of nucleic acids: A, C, G and T. We also presented the algorithm to index DNA sequences with *ed-tree* and the search algorithm on *ed-tree*. Compared to the popular Blastn method [7], the *ed-tree* supports more flexible probe model with longer probes and more relax matching. The query of our method is up to 6 times faster than Blastn. Moreover, to index a DNA database of 2 giga base pairs(Gbps), the *ed-tree* only takes less than 3Gb hard disk storage which is easily handled by a desktop PC.

1.4 Mining Sequential 3D Patterns in Protein Structures

Life science data are complicated. In real-life biological applications, many datasets such as protein chains could be represented by sequential 3D structures. Existing subspace clustering methods mainly process value-based patterns which are located on same dimension group.

1.4.1 Motivations

Traditional relational database may not be applicable for modeling and analyzing such complex data especially protein structures. Protein chains can be represented as sequential 3D objects. The frequent protein 3D patterns are very meaningful in

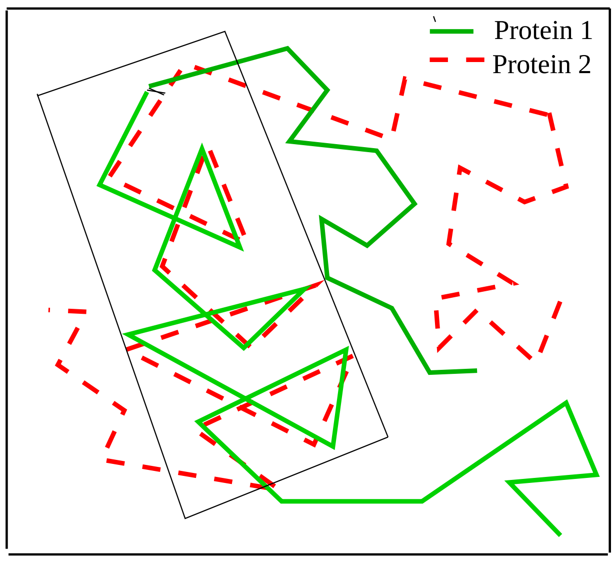


Figure 1.8: Example of subspace clustering

many biological and pharmaceutical applications. However, most of the existing subspace clustering methods are based on value similarity and pattern similarity instead of 3D structure similarity where translation and rotation should be considered. We studied subspace clustering method in terms of 3D coordinates and the method was applied to discover 3D structural motifs in protein families.

1.4.2 Our Research Problem

Sequential 3D objects appear in many real-life applications. To find out all the frequent substructures in the sequential 3D object dataset is a common and meaningful problem. The maximal pattern is defined as a group of substructures which cannot be extended either in terms of length or in terms of occurrences. As shown in the area specified by the rectangle in Figure 1.8, one substructure of protein 1 exhibits the similarity with one substructure of protein 2. The purpose of the study in this thesis is to find out the frequent patterns in sequential 3D dataset.

Additionally, because datasets often include objects from various classes and it is possible for a pattern to appear in different classes, the constrains of the minimum support and minimum confidence should be considered during pattern mining. These constraints form the basis of applications such as classification and prediction [55].

1.4.3 Contributions: sCluster And MSP

According to our knowledge, we are the first to investigate mining subspace clusters with respect to 3D coordinates in sequential structural datasets. Motivated by the lack of suitable mining approaches to protein chains, we started to work on clustering substructures in sequential 3D object dataset. We proposed sCluster [83, 85] for mining sequential structural subspace clusters in terms of 3D coordinates. The obtained clusters are the non-trivial clusters, *maximal sCluster*, which cannot be contained by another cluster. sCluster is an extended apriori [5] algorithm to expand the pairwise *maximal* sClusters with respects to both the cardinality and the length. We also extended the approach to support query, i.e., to incrementally generate the *maximal* sClusters only related to the given new object. Due to the absence of existing subspace clustering methods on sequential 3D objects, we compared sCluster with an *rmsd*-based clustering to evaluate the performance. Experiments showed that sCluster was faster than the *rmsd*-based method by magnitudes. Furthermore, randomly selected *sCluster*s in protein chains illustrated the effectiveness of our results.

As an improvement of sCluster, MSP [86] was proposed for mining maximal sequential 3D patterns with the constraints of minimum support and mining confidence based on a seed-and-extension strategy. MSP includes three stages. First, short patterns with fixed length appearing in two 3D objects are produced as the seeds. Second, the vertical extension, a novel depth-first search algorithm is adopted to enumerate the hits of seeds in all objects with the constraints of minimum support and minimum confidence. Third, the horizontal extension is to extend every pattern to be the longest without loss of hits. Furthermore, a dual-level binary-search algorithm, DPS, is implemented to automatically identify the proper settings to produce the number of patterns specified by users. Comparison experiments showed that MSP was faster and more scalable than sCluster. We applied MSP to protein family classification, and the obtained patterns correctly classified the protein families on all the tested binary-class datasets. We also applied MSP to PhysioNet/CinC Challenge 2002 dataset and achieved both good precision and recall in the classification event.

1.5 Remote Homology Detection Based on Sequential 3D Patterns

Remote homology detection is to find out the evolution relationship between various proteins where the sequence similarities are ambiguous, i.e., to classify new protein chains to the known families. Protein sequences and their corresponding structures may change due to mutations during natural select. High sequence similarity implies that the proteins be descendants of the same ancestry family. At the same time, the similar structure occurrences also provide evidence of evolutionary relationship. The results can be applied to drug discovery, phylogenetic analysis and etc [3]. Naturally, amino acid sequences are conformed into 3D shapes which are highly conserved in the evolution process.

As protein sequences are translated from DNA sequences, it would be helpful to further study DNA regions which code the frequent protein 3D structures.

1.5.1 Motivations

Many researchers proposed methods [7, 8, 18, 43, 52, 54, 80, 81] focused on the analysis on amino acid sequences. However, the 3D structure of proteins are more resilient to mutations than sequences due to the conformation and functional constraints [3]. Remotely homologies are statistically undetectable using traditional classification methods which are mainly based on sequence identity [36]. This promotes us to study family classification for remotely homologous proteins based on 3D structural motifs to be a complement of the sequence based methods. Moreover, a convenient visualization tool should be provided to better understand 3D structures.

1.5.2 Our Research Problem: Protein Classification Based on 3D Structures

A protein chain can be represented as a sequential 3D object where the vertices are the C_α atoms with coordinates and the edges are the links between neighboring C_α atoms.

Given a new protein structure, q, we are going to predict the most possible protein family which q belongs to based on the 3D structural pattern comparison.

1.5.3 Our Research Problem: Finding Coding DNA Regions for Similar 3D Protein Structures

Given a DNA sequence dataset and a protein 3D structure dataset of an organism, we are going to find the DNA sequences which code similar protein 3D structures.

1.5.4 Contributions: Deterministic Binary Classification Tree

We proposed a classification approach that was purely based on 3D structural features. We aimed to find an accurate classification method for remotely homologous protein whose sequence identity to known protein is less than 30% but the functionalities are similar. Our method generated the discriminative frequent 3D patterns for each two groups of proteins, i.e., the patterns appear in only one group. A mechanism, called deterministic binary classification tree ($DBCT$) [84] was proposed to incorporate the pattern groups for multi-class classification. Our method can be a good compliment of the existing sequence based methods.

1.5.5 Contribution: FCDR System

We built a FCDR System to preprocess DNA sequences and protein 3D structures, to interactively visualize 3D structures and to search DNA regions which code similar 3D structures.

1.6 Outline of This Thesis

This thesis would be organized as follows. In Chapter 2, we described the existing works for all the topics that would be discussed in this thesis. In Chapter 3, the *ed-tree* was presented together with the algorithms for building *ed-tree* over the given DNA sequence database, the search process with the *ed-tree* and the experimental evaluation results. In Chapter 4, sCluster was proposed for mining non-trivial subspace clusters in sequential 3D dataset. Hence, MSP was presented in Chapter 5 for mining maximal sequential 3D patterns with the constraints of minimum support and minimum confidence based on a seed-and-extension strategy. Both sCluster and MSP were evaluated on protein structures. In Chapter 6, we described a classification approach that was purely based on 3D structural features. FCDR System was introduced in Chapter 7. This thesis was concluded in Chapter 8.

CHAPTER 2

State of Arts

2.1 Homology Search in DNA Sequence Datasets

With the increasing interest on genomic research, various DNA sequence searching systems [7, 16, 17, 30, 41, 45, 59, 79, 91] have been developed to support different objectives. Some methods locate similar regions in the sequence database by sequential scan while others index the databases using novel data structures which can speed up homology search processes where homology means the similarity in different DNA sequences.

2.1.1 Sequential-scan-based Approaches

There have been many proposals on performing a full scan on the sequence database for homology search. The most fundamental method is the Smith-Waterman algorithm [77], which performs sequence alignment between query sequence and target sequence using a dynamic programming algorithm in $O(mn)$ time with m and n

being the lengths of two sequences.

Blastn

Blastn [7] is the most widely used DNA homology search system since 1990. It considers exact match of w contiguous bases as candidates which are extended greedily towards both left and right side to obtain the final alignments. However, Blastn faces a difficulty in the choice of w since increasing w decreases sensitivity whereas decreasing w slows down computation.

BLAST includes three algorithmic steps, compiling a list of high scoring words, scanning the database for hits and extending hits. These stages vary somewhat depending on whether the database contains proteins or DNA sequences. For proteins, the list consists of all words that score more than a threshold T. For DNA, given a query sequence Q, Blastn moves a sliding window of size w along the sequence Q one alphabet at time generating a total of $|Q| - w + 1$ seeds where $8 \leq w \leq 15$. It encodes every database sequence into bit representation. And it employs a finite state machine [10] to scan the entire sequence database to see if the sequence contains a k-tuple that can match with one of the query k-tuples to produce a seed with a score no less than a pre-determined threshold. These seeds are then use to query the **target sequence** R and any portion of R that match any of the seeds **exactly** are extended to check for local alignment. A full scan through the target sequence is to identify matching positions. Dynamic programming is used to find a locally maximal segment pair containing the hit. The similarity score between two sequences is determined by scoring identities +5, and mismatches -4. The extension is along both the left and the right sides until the score cannot grow any more through either extending it or cutting it short.

w is a major factor that affects the tradeoff between finding too many random matches and having fasle drop. A large value for w can result in matching regions and to be missed while a small value of w means that there could be too many random hits which slow down the computation. The root of the problem here is the requirement for exact seed match which is rather rigid for homology search. And it consequently cannot detect homologous regions with deletions and insertions.

Blast becomes popular due to its fast speed. However with the growth of the database size, its memory requirement becomes large that makes it unsuitable for biologists to build index and conduct search in large sequence database on their desktop PC.

Pattern Hunter

Pattern Hunter[59] aims to find all approximate repeats or homologies in one DNA sequence or between two DNA sequences. It is an improvement on Blast both in speed and sensitivity by using non-consecutive k letters as model, where k is the weight of model. For example, in 110100110010101111 model 1-positions mean required matches while 0s are wild cards. The hits will be extended in a greedy manner to the left and right stopping when the score drops by a certain amount. Unlike Blast, Pattern Hunter scores matches +1, mismatches -1, gap open -5 and gap extension -1. According to their reported results, this system is powerful on handling homologous search with long query sequences.

Pattern Hunter is implemented in Java using the spaced seed model and various algorithmic improvements using advanced data structures which are the key to its fast speed.

The obvious improvement of Pattern Hunter is that it introduces wildcards during hit selections. Compared to Blast, replacements can be more likely detected by this system. When generating seeds, Pattern Hunter achieves better sensitivity since it can better detect replacements in the sequences than Blast. However, homologous sequence regions with insertions and deletions still cannot be detected sensitively. Pattern Hunter is still essentially a sequential scan method which may not scale up for very large sequence databases on desktop PC.

SENSEI

SENsitive SEarch Implmentation(SENSEI) [79] is another sequential scanning method which selects hits by exact match. It outperforms Blastn by using compactly encoded scoring tables for k-tuples, encoding bases with single bits, removing the simple sequence repeats, and masking some known repeats in the query sequence. It is a tool for computationally efficient identification of nucleic acid sequence similarities, and it is particularly optimized for the analysis of large sequences.

Similarly with BLAST, SENSEI search engines is based on a search algorithm in which words generated from the query sequence are indexed by the location of their occurrences in the query. It's based on a heuristic word search similar to that of Blastn, a component of the BLAST suite of programs, is used for searching DNA query sequences against a DNA sequence database or a DNA target sequence.

Thus, for each word or k-tuple, a list of all the locations in the query sequence containing that word is generated. The target sequence is then scanned sequentially to identify potential matches by finding words in common with the query. When a word hit occurs, the program attempts to extend it on both the left and the right by checking if additional matching nucleotides can be found. If this extended word forms a significant ungapped segment (in the BLAST nomenclature, high-scoring pair or HSP) and its score achieves statistical significance, the extended word is saved.

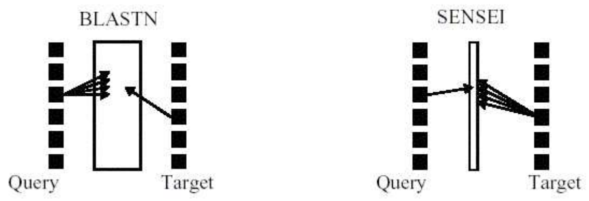


Figure 2.1: Word tables in Blastn and SENSEI

Compared to Blastn, SENSEI differentiates itself in four points including: First, Figure 2.1 shows that multiple words for each query address are stored in the word look-up table in Blastn, while only a single word for each query address is stored in the word table in SENSEI. Second, in Blastn, both the positive and the negative strands are considered. While SENSEI uses only the positive strand. Third, Blastn encodes each base into 2 bits. SENSEI offers two representations of a base. The default one is representing a base using 1 bit. Both A and G are encoded by bit(0). Both C and T are encoded by bit(1). Finally, during the extension of hit word, Blastn moves single base at a time compared to SENSEI that extends high score pair (HSP) scores 8 bases-pair at a time.

In summary, SENSEI is a variant of Blastn. It uses a logical exclusive-or (XOR) to encode the score table and extends HSP scores 8 base-pair at a time. And it does not find more homologous sequences than Blastn.

Locality Sensitive Hashing

LSH-ALL-PAIRS was proposed by Buhler in [16] for finding longer seeds to improve efficiency, while maintaining sensitivity for weak similarity by using the technique of locality-sensitive hashing(LSH). However false drops and false hits cannot be completely avoided because the result is sensitive to the hashing functions being used. Furthermore, it is possible to miss some short alignments in a collection of sequences.

A Whole Genome Alignment Method

In [44], the problem of finding local alignments for huge genome strings has been discussed. The authors construct a match table, a boolean matrix in which an entry is marked as true if the corresponding substrings may be similar, and false otherwise. The match table is used for pruning and partitioning search. After that, they compute scores in the frequency domain and find local alignments between two strings. This method addresses on comparing two long sequences such as whole genomes.

2.1.2 Suffix Tree Based Approaches

For a given string, suffix tree indexes every suffix that can be uniquely traced from the root to the corresponding leaf. Concatenating all characters along the path from the root to a leaf will produce the text of the suffix.

Suffix tree was well studied by many researchers [41, 60]. It's powerful to do exact matching which costs $O(n)$ time. Theoretically, for a string with n characters, its suffix tree requires $O(n)$ construction time and $O(n)$ space using suffix links.

However most of the suffix tree variant suffers from the memory bottleneck. The bottleneck is caused by not only the source string itself but the tree data structure. Suffix links also contribute substantially to the bottleneck. In all, suffix tree is powerful to handle exact matching. On the other hand, it has two obvious weaknesses.

1. Memory bottleneck: up to now, the memory requirement of suffix tree is still too high to be adopted to practical use for large DNA sequences. As we know, the longest sequence which can be indexed is 263Mbp under the limitation of the 2GB RAM.

2. Hard to handle mismatch: according to its structure, it's easy to know suffix tree is difficult to detect mismatch while for biologists distant homologous sequences are often biologically significant.

QUASAR

QUASAR [17] applies a modification of q-tuple filtering implemented on top of suffix array. It aims to search for sequences that are strongly similar to a given query sequence. This approach is based on an observation: if two sequences have an edit distance below a certain bound, it is guaranteed that they share a certain number of q-grams[87]. A filter is designed to select candidate positions from the database where the query sequence possibly occurs with a high level of similarity. Generally speaking, the approach is to solve approximate matching by reducing it to exact matching of short substrings of length q (called q-grams). It relies on the following lemma:

Lemma 2.1.1 *Let an occurrence of $S[1 : w]$ with at most k differences end at position j in D. Then at least $w + 1 - (k + 1)q$ of the q-grams in $S[1 : w]$ occur in the substring $D[j - w + 1 : j]$.*

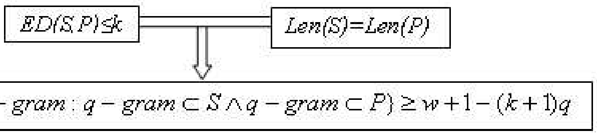

$ED(S,P)\leq k$
$Len(S)=Len(P)$
$\#\{q-gram : q-gram \subset S \wedge q-gram \subset P\} \geq w+1-(k+1)q$

Figure 2.2: Lemma of QUASAR

This lemma gives a necessary condition for a subsequence of D to be a candidate for an approximate match with $S[1 : w]$ occur in a substring of D with length w. Substrings of D with this property are potential approximate matches and will later be checked with an alignment algorithm. The suffix array[60] A built on database

D is an array of length $|D|$ storing the lexicographically ordered sequence of all suffixes of D. Entry $A[j]$ contains the text position where the j-th smallest suffix of D starts. In order to find all approximate matches between $S[1 : w]$ and D, all the substrings in D that share at least t q-grams with $S[1 : w]$ are identified. For each block, they maintain one counter.

QUASAR is good search tool for highly similar strings. The database is split into blocks of a fixed size. A window slides on the query sequence, and the number of common q-grams of the sub-query specified by the window and each of the database blocks are counted using the suffix array. Database blocks which contain less common q-grams than a given threshold are eliminated. Experimentally this approach is an order of magnitude faster than BLAST. This technique has three drawbacks. First, it has extensive memory requirement. The memory requirement is 9 times of the database size at construction phase, and 5 times of the database size at run time. Second, the performance deteriorates quickly for distant homologous sequences. Third, it leads to false drops. The substrings that span two consecutive blocks are not considered, and each block has only one counter.

Disk Based Suffix Tree[41] And In-memory Compressed Suffix Array [75]

To my knowledge, the most recent work in [41] disposes of suffix links. The authors reduce storage by not storing the suffix number and the right index into the string for each node. The suffix number is calculated during tree traversal. The right pointer into the string is looked up in the child node, or in the case of leaves, is equal to the size of the indexed string. Multiple passes over the sequence are performed to construct the suffix tree for a subrange of suffixes at each pass. Because of removing the suffix links, the construction of a new partition corresponding to a different subrange does not need to modify previously checkpointed partitions of the tree. Each tree node consists of two object references costing 4Beach(child, sibling). The observed space is about 28B per node in memory. With this method, the DNA of 263Mbps can be handled by a 2GB log and 18GB in files of 2GB each.

2.1.3 Index-based Approaches

There exist DNA sequence searching methods which pre-build indexes for searching the sequence database.

SST

In SST [30], each sequence is partitioned into fragments according to the window size, and each window is mapped into a vector. Tree structured vector quantization is used to create its tree-structured index by a *k*-means clustering technique. SST is much faster than Blastn when searching for highly similar sequences. Unfortunately, since the distance between sequences in vector space does not correspond well with the actual edit distance, there would be substantial false dismissals if the similarity between the query sequence and the target sequence is not sufficiently high.

CAFE

CAFE was proposed by Williams et al. as a searching algorithm in a research prototype system. CAFE[91] is based on techniques used in text retrieval and in approximate string matching used for databases of names. It contains two components.

1. Coarse search: Uses an inverted index to select a subset of sequences that display broad similarity to the query sequence.

2. Fine search: A computationally more expensive fine search mechanism ranks the resultant sequences from the coarse search in order of relevance to the query. The ranked results are presented to the user.

An inverted index has two components: a search structure and posting lists. The search structure consists of the set of unique searchable terms, i.e., the set of intervals. A posting list is associated with each term in the search structure. For example, consider the following postings list:

$$ACCC12, (3 : 144, 154, 962), 38, (2 : 47, 1045)$$

In which the indexed sequences, the 12th and the 38th, contain the interval $ACCC$, the interval appears three times in 12th sequence, at offsets 144, 154 and 962, and twice in 38th sequence, at offsets 47 and $1,045$.

FRAMES are the ranking structure. Frame-based metrics incorporate the relative positioning of matching intervals, as well as other calculated metrics, to give a model of likely homologous alignments.

A simple scoring metric that can be calculated using frames is to rank frames based on the number of intervals in each frame for two sequences s and t, so that

$$framecount(s, p) = max(|F(I(s) \bigcap I(t))|)$$

where $I(s)$ is the set of intervals in sequence s, $I(t)$ is the intervals in sequences t, and $F()$ is the frame function that returns one or more sets of intervals that are at the same relative offset.

The length and the coverage are considered together to be *Combined.*

$$Combined = coverage - k \star (length - coverage) \quad k << 1$$

Note: *length* is the total number of bases that lie between the two intervals that have the smallest and largest offsets. *Coverage* is a count of the actual number of residues or bases that match between two sequences.

According to the reported experiment results, CAFE can be over eighty times faster than FASTA and eight times faster than BLAST2. The response time depends on query length and the statistics distribution of the intervals in the query; intervals with longer inverted lists require more processing.

A Wavelet-based Method

In [45], a wavelet-based method was proposed to map the subsequence of the database into a 2σ dimensional integer vectors where σ is the alphabet size of the sequences. The number of the dimension is determined by the alphabet size and the number of wavelet coefficients. FD(Frequency Distance) is defined as a lower bound of the edit distance. A sliding window is used to translate a set of contiguous substrings into a MBR(Minimum Bounding Rectangle). This translations are repeated over all the strings to generate an array of MBRs of the database. This approximates to the database at different granularities, and produces a grid of MBRs. The index structure is quite compact and can be stored in memory. Typical size of this index structure ranges from 1% to 2% of the database size. For similarity search, range queries and nearest-neighbor queries are performed using this in-memory index structure with the lower-bound distance at first. Hence, the obtained candidate pages are then accessed from the disk to remove false hits (using the actual edit distance). Although this method avoids false dismissals, there are lots of false hits since the approximation of edit distance is not sufficiently tight. This increases the cost for refining the final result. In addition, this method tries to find the regions of the target sequence which are similar to the **whole** query sequence and **not part of the** query sequence.

2.2 Subspace Clustering And Pattern Mining

High dimensional dataset is increasingly common in many fields. As the number of dimensions increases, many clustering techniques begin to suffer from the curse of dimensionality. In high dimensional datasets, data becomes very sparse and distance measures become increasingly meaningless [67]. There should be appropriate data mining solutions for different applications. This encourages the extensive studies on the subspace clustering [1, 2, 19, 27, 39, 40, 42, 48, 64, 88, 95].

2.2.1 Subspace Clustering

CLIQUE

CLIQUE [4] is a subspace clustering algorithm combining density and grid based clustering and uses an APRIORI style search technique. Once the dense subspaces are found, they are sorted by coverage defined as the fraction of the dataset the dense units in the subspace. CLIQUE filters the subspaces with smallest coverage and expands the remained subspaces in a greedy-growth manner [67]. The progress of region growing and the density based approach for generating clusters enable CLIQUE to find clusters of arbitrary shape, in any number of dimensions.

CLTree

CLTree [56] - the cluster tree, is a supervised learning method. It uses a modified decision tree algorithm to adaptively partition each dimension into cluster and sparse regions at different levels of details. This method is based on one hypothesis:

If there are clusters in the data

Then the data points cannot be uniformly distributed in the entire space.

Non-existing points are introduced to the data space and a new purity function is designed to look ahead in determining the best partitioning. Assume Y is the class of the record existing in the dataset. By adding some uniformly distributed N points (non-existing points), CLTree can isolate the clusters because within each cluster region there are more Y points than N points. Consequently, the decision tree technique facilitates this clustering.

PROCLUS

PROCLUS [1] - PROjected CLUStering, is a top-down subspace clustering algorithm. It allows the selection of different sets of dimensions for different subsets of the data. Because normal feature selection algorithms do not work on all types of datasets, the authors proposed the projected clustering to process the correlation among various subsets of the given dataset. PROCLUS is an algorithm to partition data points together with the sets of dimensions on which points in each cluster are correlated. Compared to the previous clustering methods, PROCLUS generates not only the clusters but also the guaranteed partitions.

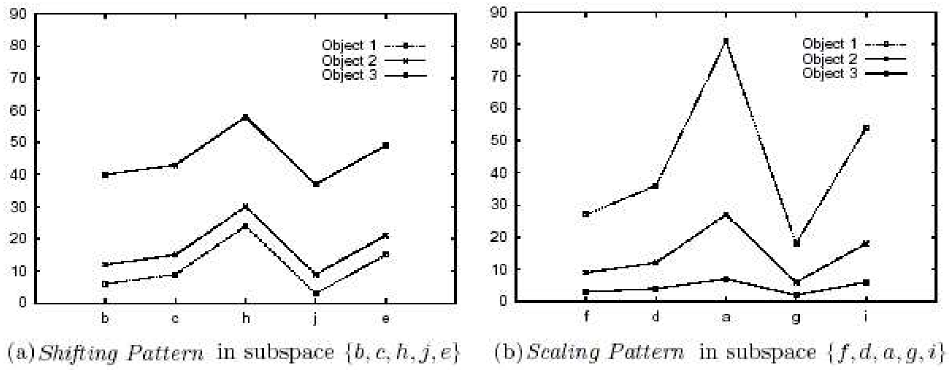
δ-Cluster:

The δ-cluster was introduced by J. Yang et. al. in [95]. It is considered as a generalization of the subspace cluster model for mining the cluster of points/objects that have coherent behaviors rather than points/objects that are physically close to each other. The type of coherence is common in many real-life applications such as recommendation systems and target marketing in E-commerce and analysis on DNA microarray datasets.

A metric of residue is introduced to measure the coherency of among points/objects in a given cluster. The main objective of this measurement is to capture a set of objects which exhibits strong coherence on the set of dimensions/attributes despite the fact that each object may bear a nonzero bias/offset. To find out all δ-clusters is a NP-hard problem. The authors presented a randomized moved-based algorithm, FLOC. FLOC is a two-phase process where the first phase is to construct k initial clusters and the second phase is to iteratively process to improve the quality of clusters until no improvement can be achieved. In the iteration, every row and every column is examined to determine its best actions towards reducing the average residue. As a result, FLOC discovers near optimal δ-clusters.

pCluster

pCluster [88] was proposed to capture not only the closeness of objects but also the similarity of the patterns exhibited by the objects. *pCluster* is a generalization of subspace clustering in the applications where pattern similarities among a set of objects carry significant meanings. The goal is to discover such shifting or scaling patterns from large-sized raw data sets as shown in 2.3.



(a) *Shifting Pattern* in subspace $\{b, c, h, j, e\}$ (b) *Scaling Pattern* in subspace $\{f, d, a, g, i\}$

Figure 2.3: Shift pattern and scaling pattern in *pCluster*

pScore is defined to evaluate the similarity between patterns. This paper focuses on the problem on the pattern similarity during clustering where most of the traditional value-similarity-based subspace clustering approaches are special cases in the *pCluster* model. Furthermore, to avoid redundancy, *Maximum Dimension Set*(*MDS*) is defined. A depth-first search process is designed to deterministically cluster all similar patterns without loss.

In this model, the value could be shifted on the same dimension groups. However the model and the algorithm could not discover the patterns appearing in different dimensions.

AnMol

A platform [78] called AnMol was proposed mainly for supporting similarity search over structural data of large biomolecules. The author targeted to create a system to answer the below questions:

1. Show me the set of all proteins that are similar to the hemoglobin molecule (PDB ID: 1a00).
2. Among the set of molecules that are similar to hemoglobin, which of them are likely to contain the given active region (structural fragment)?
3. Show me the set of all frequently occurring structural fragments among the given set of molecules.
4. What are the possible mutations of protein 1a8i?
5. What are the commonly occurring substructures between 1a2a and 1a2e?
6. Which molecule has a similar disulphide bonding structure as that of 1buea?

Figure 2.4 displays the architecture of AnMol platform. The structural information is represented using one or more vectors. Structure vectorization is one-way hashing function to represent a graph structure in the form of vectors. Graphs are hashed to vectors at different levels depicting different granularities. It is best suited

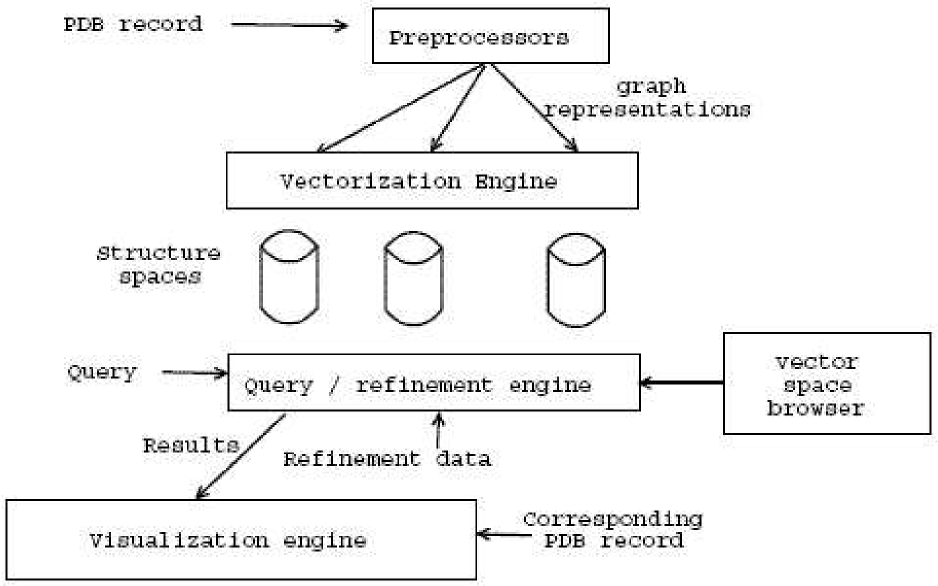
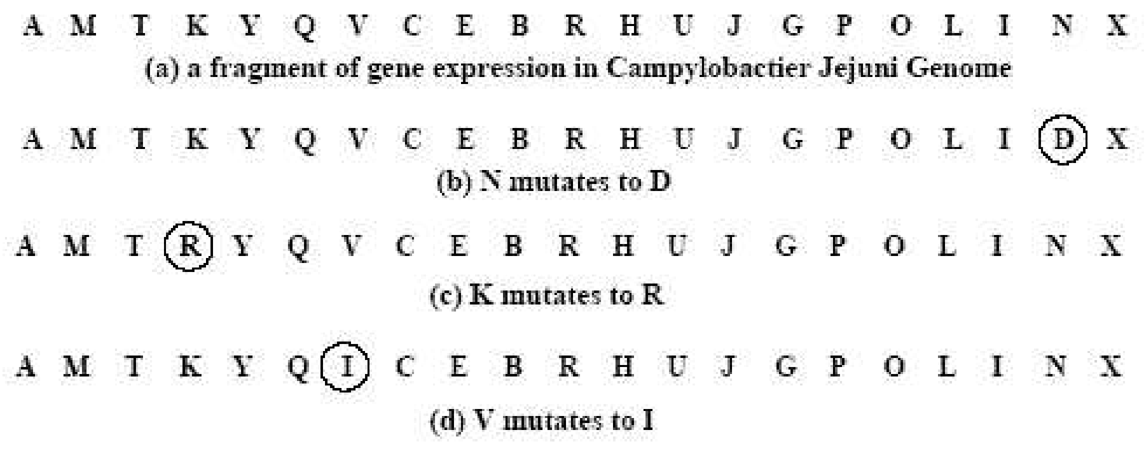


Figure 2.4: The architecture of AnMol

for labeled graphs where the number of labels is small and diversity in structures is large. AnMol enables the nearness queries, substructure queries and mutation queries. A new distance measurement based on the vectors, AnMol distance is given to avoid complex *rmsd* computation. The comparison experiment depicts the substantial difference between *rmsd* and AnMol distance but it reveals the similarity from the graph prospective.

Mining Long Sequential Patterns

Yang et. al. [96] presented an approach for mining sequential frequent patterns with noisy data. In pattern discovery in long sequences, due to the presence of noise, a symbol may be misrepresented by some other symbols. The substitution may prevent an occurrence of a pattern from being recognized and in turn slashes the support of the pattern. Figure 2.5 shows a fragment of gene expression that is found in campylabacter jejuni genome. The problem becomes critical when the pattern is long because a long pattern is much more vulnerable to the distortion



A M T K Y Q V C E B R H U J G P O L I N X

(a) a fragment of gene expression in Campylobactier Jejuni Genome

A M T K Y Q V C E B R H U J G P O L I D X

(b) N mutates to D

A M T R Y Q V C E B R H U J G P O L I N X

(c) K mutates to R

A M T K Y Q I C E B R H U J G P O L I N X

(d) V mutates to I

Figure 2.5: Example of sequence patterns with noise

caused by noise.

A compatibility matrix is introduced as a means to provide the connection from the observation to the underlying true value. The author used a novel statistical sampling method and a border collapsing algorithm to discover long patterns in minimum number of scans of sequence databases with sufficiently high confidence.

This paper mainly focuses on mining the frequent-occurred sequences in characters instead of sequential substructure patterns. However, it reveals the importance of mining sequential patterns in noisy environments.

Detection Of Common Geometric Substructure in Proteins

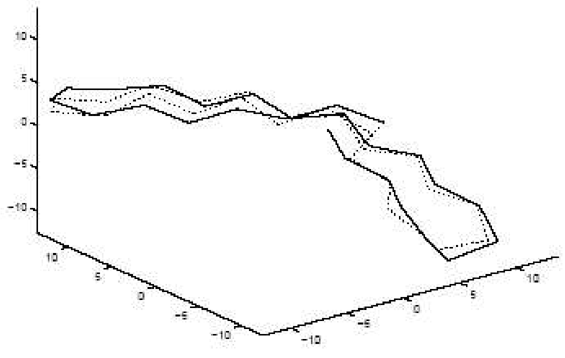


Figure 2.6: Sample result of common structures

Chew [22] et. al. proposed a genometric representation of protein chains which were represented as a sequence of unit vectors, defined by each adjacent pair of C_α atoms. All the algorithms developed so far which are based on calculating *rmsd* suffer from the running time which is high-degree polynomials in the number of atoms in the protein molecules. This caused them infeasible for practical applications. It motivated the author to define a practical measure of similarity between two protein structures based on *rmsd* between corresponding orientation vectors of two proteins.

The authors represented the protein backbone chains with vectors of C_α. Using an RMS (Root-Mean-Square) distance on the unit-vector representation $\{v_i\}$ helps to avoid the drawbacks while retaining the computational efficiency of the minimum RMS calculations. The representation $\{v_i\}$ is consisting of a sequence of points on the unit sphere in R^3. To compare two proteins, with sets of unit vectors $\{v_i\}$ and $\{w_i\}$, they computed the minimum RMS distance between the two sets $\{v_i\}$ and $\{w_i\}$, viewed as subsets of the unit sphere. To remain the origins of the vectors fixed in space, they minimized the RMS distance only over possible rotations of the vectors, not over translations. Henceforth they used the term position-RMS to refer to the standard position-based minimum RMS distance and used the term unit-vector RMS (URMS) to refer to the minimum RMS distance for a pair of sets in our unit-vector representation. In this paper, URMS was proposed as an approximate RMS distance measurement on protein structures. The computation cost for calculating the URMS between any two same length protein structures is $O(n \lg n)$ where n is the number of C_α atoms on backbone.

Figure 2.6 shows a sample result of this approach. The method is mainly for discovering common substructure between protein structure pair.

2.2.2 Graph Pattern Mining

Many studies [29, 38, 42, 49, 50, 90, 93, 92] about graph pattern mining have been contributed recently.

gFsg

gFsg [49, 50] is an algorithm for finding frequent patterns corresponding to geometric subgraphs in a large collection of geometric graphs. It is a level-by-level apriori algorithm which iteratively refines shapes in order to optimize the patterns. At the beginning, it generates the initial patterns using an enumeration approach. During this enumeration, the patterns with low frequencies are pruned. After that, it starts a heuristic candidate generation process. During the refinements, the pattern frequency and the number of patterns increase.

gSpan

gSpan [92] was proposed to mine frequent connected subgraphs in large graph databases by employing a depth-first search strategy. gSpan builds a new lexicographic order among graphs, and maps each graph to a unique minimum DFS code as its label. DFS lexicographic order and minimum DFS code form a novel canonical labelling system to support DFS search. It combines the growing and checking of frequent subgraphs into one procedure, thus accelerates the mining process. Compared to the previous methods, gSpan eliminated the cases that the same subgraph patterns were found repeatedly.

CloseGraph

As an improvement of gSpan, CloseGraph [93] has been presented in order to find the closed graph whose supgraphs cannot have the same support. The authors developed equivalent occurrence and early termination to enable CloseGraph to significantly prune the search space with modest additional costs. The pruning would lead to missing of some patterns while the authors presented another approach to eliminate these cases. The main enhancement is that the results of CloseGraph are non-trivial, i.e., the closed graphs avoid the trivial graph patterns which are the subgraphs of the other patterns with the same support.

Mining Closed Relational Graphs with Connectivity Constraints

Yan et al. [94] has studied the problem of mining frequent highly connected subgraphs in relational graphs where the label for each node is unique. Since this study aims for the applications related to biological networks and social networks where thousand of nodes and millions of edges would be involved. The nodes are unique, relational graphs may be large and the result patterns should be both frequent and satisfy user-specified connectivity constraints.

GraphMiner

Wang et al. [90] demostrated a prototype system - GraphMiner for mining frequent patterns from large disk-based graph databases with constrants based on an index structure, ADI. Motivated by the substantial growth of chemical compound databases, plan databases, XML documents, web logs, citation networks, and social networks, the authors produced GraphMiner by integrating index structure and the mining algorithms with a good implementation.

As shown in Figure 2.7, initially the graph mining engine builds the ADI index over the database. With the constraint composer, users can input their target constraints. These constraints would be transferred to the mining engine to start the pattern mining. The results can be viewed using the pattern browser. GraphMiner

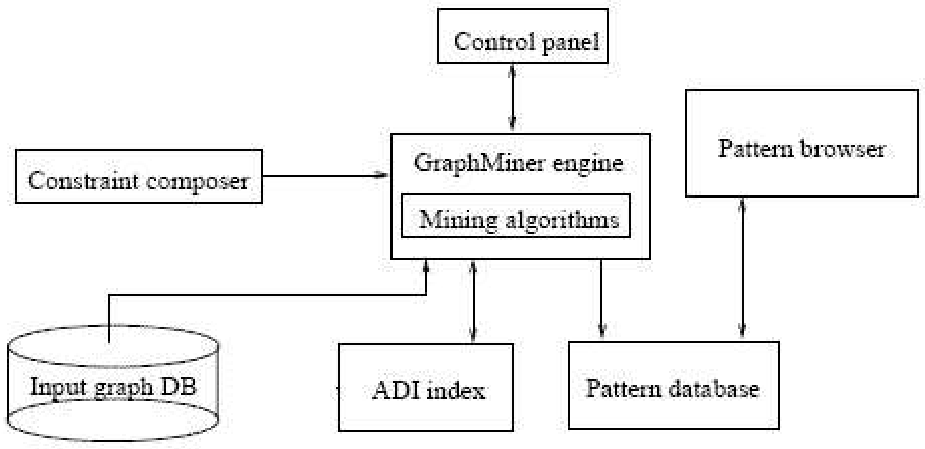


Figure 2.7: The architecture of GraphMiner

is a system leveraging on multiple research results. It can be a practical tool for real-life application.

SPIN

SPIN [40]is an algorithm to mine maximal frequent subgraphs from graph databases. Motivated by the overwhelming abundance of patterns whose supgraphs are frequent, the authors combined the technique of mining frequent subgraphs with the technique of mining frequent trees in forest because tree related operations are substantially simpler than the corresponding operation for graphs and many real-life graph data are trees. The approach firstly find all frequent trees from databases followed by the construction of the group of frequent subgraphs. After that, a bottom-up pruning process, a tail shrink process and an external-edge pruning process is employed for optimizations. Reported by the authors, SPIN outperforms FFSM and gSpan in terms of scalability on large graph databases.

Mining Protein Family Specific Residue Packing Patterns From Protein Structure Graphs

In paper [38], Huan et al. converted protein 3D structures using three graph representations: first, based on simple distance threshold between contact residues; second using the Delaunay tessellation from computational geometry, and third using the recently developed almost-Delaunay tessellation approach. The graphs are built by representing the coordinates of C_α atoms as the vertices which were labelled by the residue type. The edges were generated according to the representation method. After representing the protein structures by graphs in forms of CAMs (canonical adjacency matrixes), the authors created a CAM tree and carried out a frequent subgraph mining process recursively until all CAMs were handled [39]. A post-process was used to filter the subgraphs appearing across families and to select the subgraphs which have high distinguishing power between protein families.

With the obtained subgraph patterns from protein families, Support Vector Machine with the radius kernel was applied for classifications. In the later chapter, we would discuss the comparison results between MSP and this method.

2.3 Remote Homology Detection

In 1981, Smith et. al. [77] presented an alignment tool based on dynamic programming to identify common biological sequences. BLAST [8] and FASTA [70] were created and widely used to provide evidences for homology by matching a new sequence against the annotated proteins.

SCOP

On 1995, Murzin et. al. [9, 25, 63]constructed the Structural Classification of Proteins (SCOP) database. The classification is on hierarchical levels that embody the evolutionary and structural relationships. Family, superfamily, common fold and class are the four levels in the classification. Based on the secondary structures, there are four types of classes including all alpha, all beta, alpha and beta, alpha plus beta and multi-domain. Proteins which exhibit sequence identity larger than 30% or very similar functions or structures are classified to the same family. The SCOP database aims to provide a detailed and comprehensive description of the structural and evolutionary relationships between all proteins whose structure is known. Many researches on remote homology detection were evaluated according to SCOP database.

CATH

Similar with SCOP, CATH [66, 68] is a hierarchical classification of protein domain structures, which clusters proteins at four major levels, Class(C), Architecture(A), Topology(T) and Homologous superfamily (H). CATH is created based on the criteria different from SCOP. In this database, the class describes the domain of the protein. The architecture is the summary of the shapes. At the topology level, sequence connectivity is considered. Proteins of the same topology exhibit similar functions. On homologous superfamily level, the proteins are evolutionarily related.

Many remote homology detection methods were proposed based on support vector machine (SVM) [18, 36, 37, 43, 52, 54]. Generally speaking, the framework includes two steps: the first step is to convert protein information mainly related to their sequences and structures to feature vectors with different logic and the second step is to build SVM classifiers with different kernels.

SVM-Fisher

Jaakkola et. al. [43] proposed SVM-Fisher which was known as the first attempt to bring classifiers to remote homology detection in 1999. Hidden Markov Models (HMMs) are used to build statistical models on the sequences for protein families or superfamilies in SCOP database. HMMs provides a complete summary of the sequence in the parameter space of the model. For each family or superfamily, an HMM was trained as the model from the positive samples. Given a new sequence, HMMs would be used to map it to a fixed-length vector as the Fisher score and compute the distance between this vector to the score vectors of positive samples and negative samples. This method performs better than BLAST and it mainly relies on the sequence information.

SVM-Pairwise

Similarly with SVM-Fisher, SVM-Pairwise [54] represents the protein sequence using a fixed-length vector and applies SVM as the classifier. It differs from SVM-Fisher on the vectors. In this study, every vector is a list of pairwise sequence similarity scores computed with respect to all sequences in the training set. Besides, a large-margin SVM classifier is used. This method is not significantly better than SVM-Fisher but SVM-Pairwise is simpler to calculate compared to SVM-Fisher.

SVM with Profile-based Kernels

In [18], Busuttil et. al. computed a multiple alignment of the positive training set using ClustalW and built a profile from the multiple alignment using the position-based method. Given a sequence, it would be converted to a vector and compared to the profiles to produce the vector to measure the similarity between this given sequence and the profiles built from the pre-produced profile vectors for superfamilies. This method outperforms non-discriminative methods and it is comparable with or better than the other SVM based methods.

SVM with Mismatch Kernel

Remote homology detection should finally be applied for protein classification. SVM-Fisher and etc. involve extensive computations. The computational cost for Fisher kernel requires quadratic-time computation for each Fisher vector calculated. This motivated Leslie et. al. to propose mismatch kernel [52], a class of string kernels, to detect (as positives) amino acid sequences that are only remotely related to the positive training sequences. This kernel reduces the computational costs while the accuracy is comparable with the previous method.

SVM-I-sites

Hou et. al. proposed SVM-I-sites [36] system. Compared to the previous works, SVM-I-sites encode structure information into feature vectors instead of using sequence similarity. The local sequence-and-structure motifs are obtained from I-sites library. Given a protein sequence, it would be segmented to short subsequences. The possibility of each subsequence being one of the motifs in I-sites would be evaluated. Since there are 263 motifs in I-sites library, each protein sequence is transformed to a vector of length 263 where each component denotes the confidence value of the presence of the corresponding structure motif. After that, SVMs are trained on each protein class. Reported by the authors, the accuracy of this method is comparable with SVM-Pairwise but it is more efficient than SVM-Pairwise.

Protein Seer

Protein Seer [57] is a protein classification system which relies on the concept of protein family which is a group of sequences sharing same evolutionary origin. It built a statistical model for each family or superfamily. Given a new sequence, the system computes the probability according to every statistical model. A discriminative framework such as HMM and SVM is used to learn a boundary between two or more classes. In Protein Seer, the classifications were conducted on superfamily level in SCOP database. The proteins in each superfamily were converted to fixed dimension representation to be positive samples. The protein in different folds were selected as negative sames. SVM was used to separate the superfamily based on the positive and negative samples.

eKISS

Since the proteins are imbalanced distributed in the SCOP classification, eKISS [81] was proposed to generate one-against-others classifiers which are capable of learning over multi-class examples under the skewed normal distribution of the training examples. The common approach to multi-class learning is to transform the K classes into a set of two-class problems, which is also known as one-against-others method. Another approach is to generate all the possible pairwise two-class classifiers between K classes from the training examples, i.e. all-versus-all method. eKiss is a machine learning for imbalanced data.

TFASTX, TFASTY

W.R. Pearson et. al. presented TFASTX and TFASTY [71] for comparing a protein sequence to a DNA sequence database, translating each sequence in the DNA database in six frames and scoring alignment with gaps and frame shifts. TFASTX allows only frame shifts between codons while TFASTY allows substitutions or frame shifts within a codon.

Aligning a DNA Sequence with a Protein Sequence

Z. Zhang et. al. [99] have further implemented the algorithms for aligning a DNA sequence with a protein sequence. Based on their definition of an alignment of a DNA sequence and an amino acid sequence, the author gave a method for computing an optimal alignment. It has been incorporated with between-codon-frame-shift algorithm in FASTX and TFASTX. This method considers frame shift errors rather than the errors inside codons.

BLASTX

BLASTX is a tool to probe nucleotide sequence directly from the presence of protein coding regions by identifying segments that encode significant similarity to members of a protein sequence database. BLASTX allows protein-protein comparison to be considered when only uncharacterized nucleotide query sequence available.

Most existing tools for finding protein-coding genes are for similar amino acid sequences. While our FCDR System is to find DNA regions which code similar protein 3D structures. It can provide a better understanding from DNA to protein structures.

In this chapter, we briefly introduced the existing research work related to the topics which would be discussed in this thesis.

CHAPTER 3

Homology Search in Large DNA Sequence Datasets

3.1 Introduction

Homology search in large sequence databases becomes more and more important. The result of searching homologous sequences can help biologists do further analysis and detection on the protein structure and function. As many existing search methods are based on sequential scanning on the databases, this growth will adversely affect the efficiency. Besides, biologists often build index and do similarity search on their desktop PC. This motivated us to develop an efficient system to enable DNA homology search in large databases on desktop PC.

The problem of homology search on DNA databases can be described in a nutshell as follow. Given a query sequence Q and a target sequence database T, find a set of subsequences of Q such that each subsequence Q' in the set is highly similar to some subsequence T' of T. The similarity between Q' and T' is computed as a function of the **edit distance**, $edit(Q', T')$, which is defined as the minimum number of edit operations (insert, delete, replace) that transform Q' into T'. While the actual similarity function can vary depending on a biologist's preference model, if resemble to the simple one that we will use in this chapter. In short, it means that our proposed algorithm and index structure will be robust with respect to changes in similarity function, defined as:

Definition 3.1.1 *ED-Similarity*(Q', T')
Given two sequences Q' and T', we measure the similarity between the two sequences as

$$\text{ED-Similarity}(Q', T') = \frac{|T'| - edit(Q', T')}{|T'|}$$

Intuitively, the measure defined above tries to estimate the maximum number of matches that occur between Q' and T' as $|T'| - edit(Q', T')$ and then normalizes the measure by dividing this difference by $|T'|$.

This chapter focuses on probe-based algorithms like the Blastn[7] which is arguably the most popular homology search tool. The latest version of Blastn works in two phases. Firstly, Blastn moves a sliding window of size w along the **query sequence** Q one letter at a time, generating $|Q| - w + 1$ probes. Secondly, a sequential scan is performed on the sequence data T to identify any portion that matches any one of the probes **completely**. These portions are then extended in a greedy fashion in both directions to identify Q' and T' that have a high similarity score. There are two problems with such an approach. The first and more obvious is that the whole sequence database must be scanned which is unacceptable for very large databases. The second is in the choice of the probe length, w, which is the major factor that affects the tradeoff between sensitivity and speed. The root of the problem here is the requirement for exact probe match which is too rigid for homology search. To find alternative way of probing, we define a model of probing as follow:

Definition 3.1.2 *Probe Model, pmodel*(w, s, r)
Given a target sequence T, let $T[i]$ denote its i^{th} letter and $T[i, i + w - 1]$ denote a substring of length w located at its i^{th} letter. We say that a homology search of sequence Q is done on a target sequences T using **probe model,** *pmodel*(w, s, r) with **skip interval,** s and **edit distance range,** r if we have the following three conditions:

1. the length of the probe is w
2. each probe is compared again all subsequences of T that are of the form $T[\alpha s, \alpha s + w - 1]$,
   $\alpha \in \mathcal{I}^+$
3. all subsequences of the form $T[\alpha s, \alpha s + w - 1]$ are extended if $edit(T[\alpha s, \alpha s + w - 1], Q') \leq r$,
   Q' being any one of the probes generated from Q

□

As an example, the default probe model for Blastn will be *pmodel*$(11, 1, 0)$ since it uses probes of length 11 by default and extends all substrings of length 11 in T if they match any of the probes exactly. However, detecting replacements, insertions and deletions is important in DNA homology search because they occur frequently in DNA mutations causing diseases and natural selections. For example, deletion of three nucleotides in CF gene was convincingly proved to cause cystic fibrosis[72].

ED-Similarity is used to evaluate the similarity of two sequences with same length. Given a group of sequence pairs with *ED-Similarity* higher than a threshold, the sensitivity of a probe model can be evaluated according to the proportion of sequence pairs which are hit by the probe model.

To investigate the querying effectiveness for varying configuration of the probe model, we randomly select 120,000 sequence pairs of equal length and with more than 50% *ED-Similarity* from est_human genome. Given each sequence pair Q and T, we use Q as the query sequence and perform a homology search on T for each probe model to see whether at least one subsequence of T is extended (we will refer to this as a hit). Based on the result from the sequence pairs, we can now compute the probability of generating at least one hit from Q to T given a certain probe model. This probability will be used to evaluate the sensitivity of a probe model. Since typical homologous search is of length 20 to 200 bases [59], we select 64 and 128 as the fixed homology length.

Because Blastn has a probe length of 11 by default, we adopted it as a benchmark and will denote it as Blastn-11. Figure 3.1 and 3.2 illustrate the sensitivity for detecting 64-bases homologous sequence pair and 128-bases homologous sequence pair in different models [1]. For the sequences of 64 bases, we can see from Figure 3.1 that adopting $pmodel(18, 2, 3)$, $pmodel(18, 3, 3)$ will find all sequences that have ED-Similarity of 0.7 and above. This compares very well against Blastn-11 which only detects 80% of such sequences. Although, $pmodel(18, 2, 2)$ and $pmodel(18, 3, 2)$ are not as sensitive as Blastn-11 when detecting sequences with low ED-Similarity, their performance also becomes slightly better than Blastn-11 when we consider sequences with ED-Similarity of 0.75 and above. For sequences with 128 base pairs, Figure 3.2 essentially tells the same story. The only difference is that the curves level off when ED-Similarity is from 0.65 to 0.7 instead of from 0.75 onwards. This is due to the longer length of the sequences which make it less likely for Blastn to

[1]Note that while we have analyzed many pmodels for their sensitivity, we only show the more relevant ones here to avoid affecting the clarity of the graphs.

miss sequences that have higher ED-Similarity. From the experiments, we see that

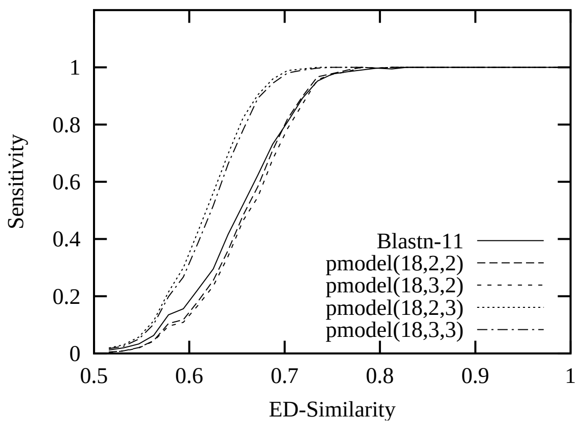


Figure 3.1: Sensitivity(64 bps)

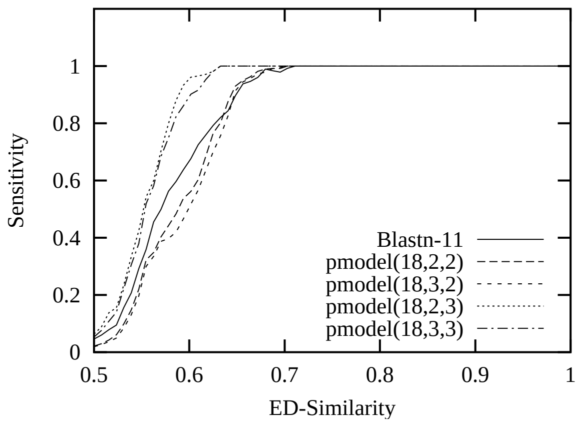


Figure 3.2: Sensitivity(128 bps)

adopting a probe model with longer probe length and more relaxed matching can result in higher sensitivity. Our aim in this chapter is to develop an index structure which will support such probe models efficiently.

Contribution:

In this chapter, we propose an index structure called the *ed-tree* which performs probe-based homology search on DNA databases with a longer probe length but more relaxed matching. The *ed-tree* has the following strengths:

Scalability. As we will see in the next section, previous work on indexing DNA sequences often results in an index that is much larger than the original sequence database. One contributing factor to this is that there is usually a need to store pointers into every position of the DNA sequences. For example, given a DNA sequence of 2 billion base pairs (i.e characters), each pointer will require around 4 bytes and the total size of the pointers alone will be 8GB. With the probe model adopted by *ed-tree*, this requirement is reduced by 2 to 3 times since not every position in the sequences needs to be indexed. Combined with some compression techniques which we will describe later, the index for 2 billion base pairs is around 2.28 to 2.97 GB which is acceptable consisting that the whole human genome is about 3 billion base pairs.

Efficiency. Unlike Blastn or other algorithms which perform a sequential scan on the database, the *ed-tree* allows us to directly access portions of the targets which are hit by probes. The *ed-tree* is a multi-layer index and the first two layers of the index are sufficiently small for storage in the main memory. This enables us to quickly prune off disk accesses in memory. As mentioned earlier, the disk-resident portion of the index is also smaller than other sequence indexes and this reduces I/Os. The use of a large skip interval also means that there are fewer comparisons to be made in the search.

Sensitivity. The sensitivity of the *ed-tree* does not short-change the scalability and efficiency. The probe model of *ed-tree* is more flexible and sensitive than Blastn since we allow certain mismatches between the target sequence and the probe. Experiments shown earlier have already illustrated this point.

3.2 The *ed-tree*

In this section, we present the structure of the *ed-tree* and relevant concepts.

3.2.1 Definitions

Definition 3.2.1 Segment Length Vector, $H = [h_1, ..., h_t]$

A segment length vector $H = [h_1, ..., h_t]$ is an integer vector in which each h_i is a positive integer. We say that a sequence S is segmented according to a segment length vector H (denoted as S^H) if we partition S into t segments $S^H[1]$,...,$S^H[t]$ such that each segment $S^H[i] = S[1 + \sum_{j=0}^{j=i-1} h_j, \sum_{j=0}^{j=i} h_j]$ with $h_0 = 0$ by default.

For example, if $H = [6, 6, 6]$ and S=AAAATTCGCGATAAGTAG, then we say that we segment S according to H by partitioning S into 3 parts, $S^H[1]$=AAAATT, $S^H[2]$=CGCGAT and $S^H[3]$=AAGTAG. As we can see, a sequence S can only be segmented by a segment length vector $H = [h_1, ..., h_t]$ if and only if $|S| = \sum_{j=1}^{t} h_j$.

Definition 3.2.2 *ed-tree*$(w, s, H), H = [h_1, ..., h_t]$

The structure of ed-tree(w, s, H) defined on a target sequence database, T, will be a tree with $t + 1$ levels [2] described as follow:

- *it contains a virtual root node at level 0 which represents a null sequence (recall that $h_0 = 0$ by definition);*
- *each node t_i, at level i represents a sequence of length h_i;*
- *a path from the root to a leaf, $\{t_1, ..., t_n\}$ (note that we leave out t_0 which represents a null sequence) represents a sequence S that is segmented based on H with each t_i representing $S^H[i]$;*

[2] For convenience of description, we consider the root of the tree to be at level 0. Thus we now have level 0 to level t for the *ed-tree*.

- *the leaf node t_n at the end of the path $\{t_1, ..., t_n\}$ consists of the pointers to all subsequences in T which match S* **exactly** *with the form $T[\alpha s, \alpha s + w - 1]$, $\alpha \in \mathcal{I}^+$.*

For example, Figure 3.3 illustrates an *ed-tree*$(18, 2, [6, 6, 6])$ for a length 42 target sequence T below:

T=AGGTAGGTAGGTAGGTAGGTAGGTAGGGCTTACATTCAGTAC

Since we have $w = 18$ and $s = 2$ in this case, all subsequences in T of length 18 and located at even positions are indexed by the *ed-tree*. As such, there are a total of $\lfloor (42-18)/2 \rfloor$ positions being indexed (In Fig 3.3, we begin indexing from the second letter). The subsequences which start at location 2, 6, 10, 14, 18 and 22 share the common prefix of "GGTAGG" which are represented by the same level 1 node. At level 2 however, the subsequence is separated into 3 portions depending on which of the subsequences "TAGGGC", "GCTTAC" or "TAGGTA" they match for their second segment. Finally, if we follow any path from the root to a leaf node, the leaf node contains the pointers to all subsequences in T that match the subsequence represented by the path.

Because the size of the leaf nodes could be rather huge for large sequence databases, to reduce this storage requirement we sort the pointers in each leaf node in increasing order and apply the frame-of-reference compression method to each set of pointers in the leaf nodes. We achieve a reduction in storage of around 40% to 50% with this approach.

This however brings a tradeoff between the selectivity of the index and the compression ratio of the leaf nodes. When the parameter w is high (i.e. more selective), there exist very few pointers in each leaf node which means that there is less space for compression because it is less likely to find a "run" of pointers which are very close in their values within the same leaf node. On the other hand,

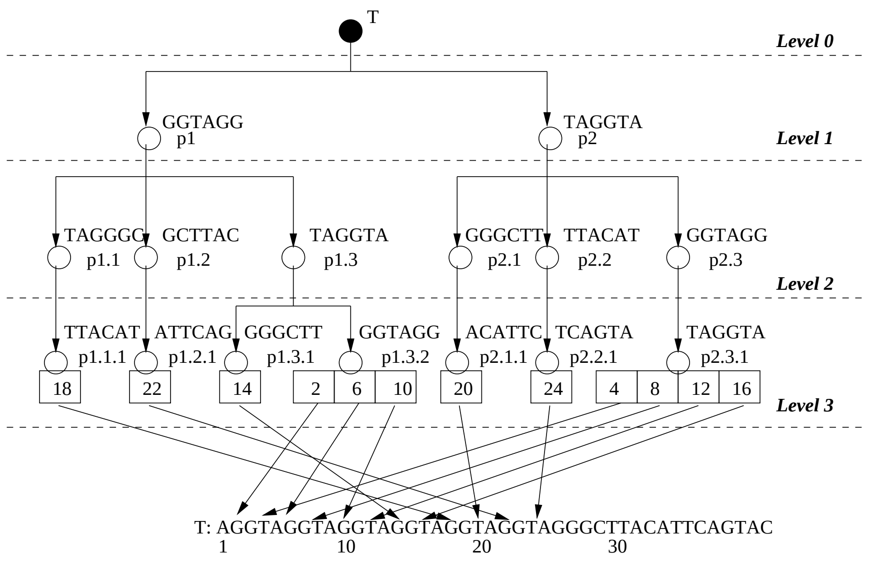


Figure 3.3: An example of *ed-tree*

if w is low, the number of pointers in one leaf node will be large which still results in large number of I/Os even after compression. Considering the memory of our targeting hardware platform, normal PC, and the four major different nucleic acids, we set $w = 18$ and $H = [6, 6, 6]$ for the DNA sequences such that its efficiency and effectiveness are good in practice.

3.2.2 Algorithm to Build The *ed-tree*

Figure 3.4 gives the general algorithm for building an *ed-tree*(w, s, H) with time complexity of $O(\frac{|D|}{s}t)$, $|D|$ is the sequence size and t is the number of the tree levels. There are two points to highlight for the actual implementation. First, for the values of H, level 1 and 2 of the *ed-tree* usually have a node that represents each possible sequence of length h_1 or h_2. As such, these two layers of the tree are implemented as a two-dimensional lookup table where the values along each dimension represent sequences and each element of the table is a pointer to the third level. To find the locations of a sequence Q, $Q^H[1]$ and $Q^H[2]$ will be used to map into a particular element in the lookup table and the third level of the tree can then be accessed. Second, for the last level of the tree, the pointers in the node are in fact **physically separated** from the node itself. This is because the pointers will only be accessed if the node is a matching one and we thus can avoid large amount of I/O by storing them separately. Figure 3.5 illustrates the physical implementation of a three level *ed-tree*.

```
Algorithm 3.2.2: Build ed-tree
input: Parameters $w$, $s$, $H = [h_1, ..., h_t]$ and the target sequence
database $T$
output: ed-tree$(w, s, H)$ for $T$
method:
1. $\mathcal{EDT}$ ← EmptyNode ; /* root of the tree */
2. Based on $w$ and $s$, slide along $T$ and for each generated probe $p$
do
  i. Segment $p$ based on $H$ into $p^H[1]$,...., $p^H[t]$;
  ii. $pt \leftarrow \mathcal{EDT}$;
  ii. i = 0;
  iii. while ( $i \leq t$ )
    If $pt$ has a child, $ch$, which represents $p^H[i+1]$ then
      { $pt \leftarrow ch$;
      $i = i + 1$ ;}
    Else
      { Add a child which sequence $p^H[i+1]$ to $pt$;
      $pt$ ← the new child; }
  iv. Add a pointer to location of $p$ in $pt$
```

Figure 3.4: Building an *ed-tree*

3.3 Homology Search with The *ed-tree*

Our algorithm consists of two phases. The first phase is to search for candidate region in the target sequence which match the probes and the second phase is to extend these matching regions. Since the second phase is essentially the same as the other probe-based algorithms like Blastn, we focus our discussion on the first phase.

3.3.1 Theories

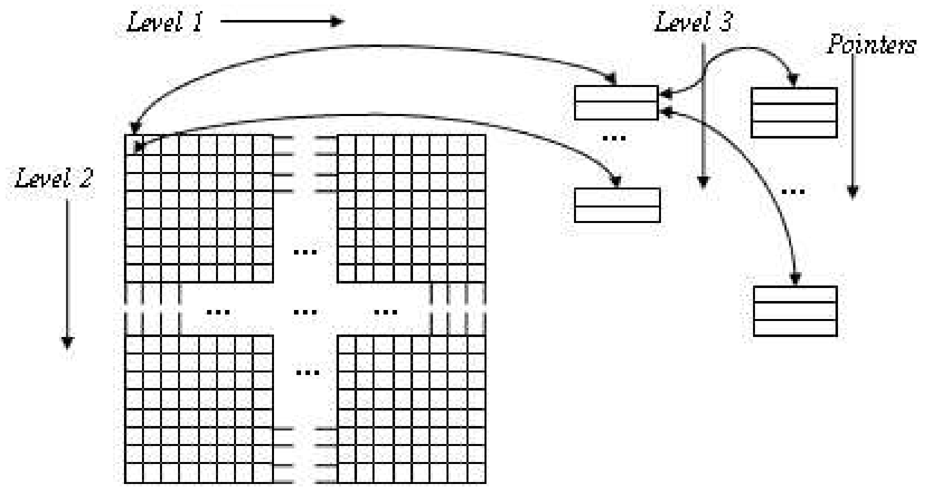


Figure 3.5: The 3-level *ed-tree* index

With a query sequence Q, a target sequence database T and the *ed-tree*(w, s, H) built on T, given a probe, P (Note: $|P| = w$) that is generated from Q, we want to find the locations of all subsequences in T that are within a distance of r from P. In other words, we want to look at how the *ed-tree* can be used to support a probe model, *pmodel*(w, s, r).

To compare P efficiently against the segmented sequences in the *ed-tree*, we introduce **matching segment** and **length difference vector** as follow.

Definition 3.3.1 *Matching Segment,* $MS(P, S^H[i])$

Let S and P be two sequences of length w and $H = [h_1, ..., h_t]$ be a segment length vector. Assume that $edit(P, S)$ is computed based on alignment $\mathcal{L}$ and that we segment S based on H. We say that $MS(P, S^H[i])$ is the matching segment of P

to $S^H[i]$ iff $MS(P, S^H[i])$ is the portion of P that can be transformed to $S^H[i]$ in the alignment $\mathcal{L}$.

Definition 3.3.2 *Length Difference Vector,* $\delta(P, S, H) = [\delta_1, ..., \delta_t]$

Let S and P be two sequences of length w and $H = [h_1, ..., h_t]$ be a segment length vector. We define the **length difference vector** between S and P based on H as an integer vector $\delta(P, S, H) = [\delta_1, ..., \delta_t]$ where $\delta_i = |MS(P, S^H[i])| - h_i$.

Example 3.3.1 Let us consider the two sequences

P=GGTAGCGGCTTACTTCAG and

S=GGTAGGGCTTACATTCAG.

Assume that $H = [6, 6, 6]$ is the segment length vector for sequence S. The alignment of sequence P and S is as follow:

```
P:GGTAGCG   GCTTAC   - TTCAG
S:GGTAG -G  GCTTAC   ATTCAG
```

Based on the alignment of S and P and the segment length vector H, P can be partitioned into 3 parts, $MS(P, S^H[1]) = GGTAGCG$, $MS(P, S^H[2]) = GCTTAC$, $MS(P, S^H[3]) = TTCAG$. Thus the corresponding length difference vector is $\delta(P, S, H) = [7 - 6, 6 - 6, 5 - 6] = [1, 0, -1]$.

Lemma 3.3.1 *Let S and P be two sequences of length w and H be a segment length vector. If $edit(S, P) \le r$, we have the following properties for $\delta(P, S, H) = [\delta_1, ..., \delta_t]$:*

1. $\sum_{i=1}^{t} \delta_i = 0$;
2. $\sum_{i=1}^{t} |\delta_i| \le r$;

Proof: Since P is transformed into S which is of the same length, the total change in length for P must be zero which implies that the sum of the elements in $\delta(P, S, H)$

must be zero. For the second property, we note that each edit operation increases or decreases the length of a sequence at most by 1. Since it takes r operations to transform P into S, this means that the absolute change in length of each segment which is computed as $\sum_{i=1}^{t} |\delta_i|$ should be less than or equal to r.

Theorem 3.3.1 Let S and P be two sequences of length w and $H = [h_1, ..., h_t]$ be a segment length vector. If $edit(S, P) \leq r$, then there exists another segment length vector $H' = [h_1 + \beta_1, ..., h_t + \beta_t]$ such all the followings are true:

1. $\sum_{i=1}^{t} \beta_i = 0$;
2. $\sum_{i=1}^{t} |\beta_i| \leq r$;
3. $\sum_{i=1}^{t} edit(P^{H'}[i], S^{H}[i]) \leq r$;

Proof: Let $\delta(P, S, H) = [\delta_1, ..., \delta_t]$. We set each value β_i to be δ_i which means that $P^{H'}[i]$ is in fact $MS(P, S^H[i])$. From Lemma 3.3.1, we will immediately know that the first two properties will be satisfied. For the third property, we should observe that $\sum_{i=1}^{t} edit(P^{H'}[i], S^H[i])$ is in fact the edit distance of P and S which is known to be less than r.

Based on Theorem 3.3.1, we derive a method for finding all the substrings of length w that are within an edit distance of r from a sequence P given *ed-tree*(w, s, H). Our approach is to generate all possible values of $\delta(P, S, H)$ such that the first two properties in Theorem 3.3.1 are satisfied and then use these values to segment P for comparison to the subsequences indexed in the *ed-tree*. We call the set of all such possible values the **cover generator** and denote it as *cover_gen*(r, t). Note that the size of *cover_gen*(r, t) is only dependent on r, the edit distance range and t, the number of segments. Combinatorial analysis shows the cardinality of the cover generator is:

$$|cover\_gen(r,t)| = 1 + \sum_{j=1}^{\lfloor \frac{r}{2} \rfloor} \sum_{i=1}^{t-1} \binom{t}{i} \binom{j-1}{i-1} \binom{j+1}{i-j+1}$$

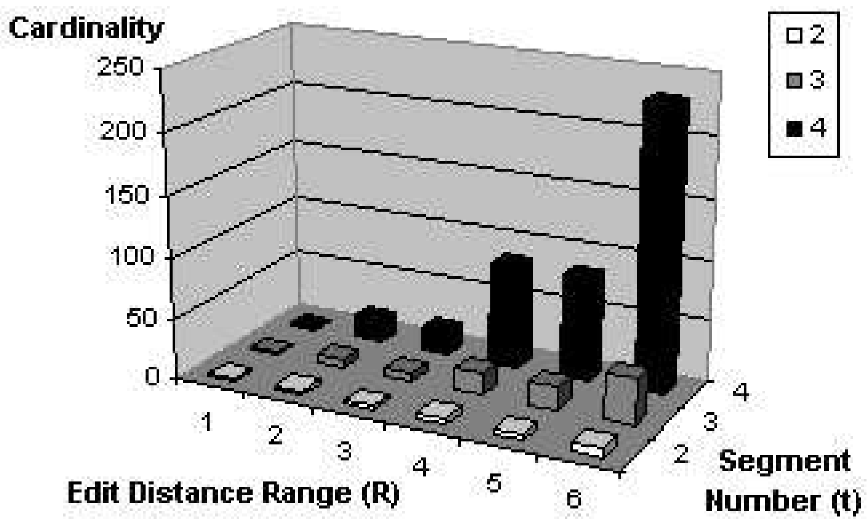


Figure 3.6: Cardinality of Cover Generator

Figure 3.6 shows how the cardinality of the cover generator varies with r and t. We note that the cardinality increases quickly with increasing t but only moderately for increasing r. For example, if t, the number of the segments is set to 3, then $|cover\_gen| = 7$ for both the cases where $r = 2$ and $r = 3$.

Example 3.3.2 Consider a sequence P=GGTAGCGGCTTACTTCAG and an *ed-tree* with segment length vector H=[6, 6, 6]. When r=2, searching the *ed-tree* will result in a cover generator, $cov\_gen(2, 3)$={[−1, 0, 1], [−1, 1, 0], [0, −1, 1],[0, 0, 0],[0, 1, −1],[1, 0, −1],[1, −1, 0]} . By adding each element in the cover generator to H, we will generate 7 different segment vectors $H_1, ..., H_7$ which will be used to generate 7 ways of segmenting P for searching in the *ed-tree*. Figure 3.3.1 shows how P is segmented based on the different elements in *cov_gen*. Let us now illustrate Theorem 3.3.1 by assuming that a sequence, S=GGTAGGGCTTACATTCAG is indexed by the *ed-tree*. In this case, since $edit(P, S)$=2, readers can verify that H_6 will be able to segment P such that all the three conditions in Theorem 3.3.1 are satisfied[3].

[3]edit(GGTAGCG,GGTAGG) + edit(GCTTAC,GCTTAC) + edit(TTCAG,ATTCAG) = 2

i	$[\beta_1, \beta_2, \beta_3]$	H_i	$P^{H_i}[1]$	$P^{H_i}[2]$	$P^{H_i}[3]$
1	$[-1, 0, 1]$	$[5, 6, 7]$	GGTAG	CGGCTT	ACTTCAG
2	$[-1, 1, 0]$	$[5, 7, 6]$	GGTAG	CGGCTTA	CTTCAG
3	$[0, -1, 1]$	$[6, 5, 7]$	GGTAGC	GGCTT	ACTTCAG
4	$[0, 0, 0]$	$[6, 6, 6]$	GGTAGC	GGCTTA	CATTCAG
5	$[0, 1, -1]$	$[6, 7, 5]$	GGTAGC	GGCTTAC	TTCAG
6	$[1, 0, -1]$	$[7, 6, 5]$	GGTAGCG	GCTTAC	TTCAG
7	$[1, -1, 0]$	$[7, 5, 6]$	GGTAGCG	GCTTA	CTTCAG

Figure 3.7: Segmenting P=GGTAGCGGCTTACTTCAG

3.3.2 The Algorithm - *Probe Search*

Figure 3.8 presents the homology search algorithm. For example, for a given input probe, P=GGTAGCGGCTTACTTCAG, we are going to find all the locations of the subsequences S indexed by *ed-tree*$(18, 2, [6, 6, 6])$ of the target sequence T, shown in Figure 3.3, such that $edit(P, S) \leq 2$.

Step 1 initializes a set, $PSet$, to store the information of **active nodes**. In *ed-tree*, **active nodes** are nodes not pruned off based on the search condition. Each tuple in $PSet$ includes 4 elements, $< v, \beta, l, d >$, where v denotes an active node in the tree, β is a member $cover\_gen(r, t)$, l denotes the level of the active node v, and d is the sum of the edit distances from the root to the level l of the active node v (i.e $\sum_{i=1}^{l} edit(a[i], P^{H+\beta}[i])$ where $a[i]$ is the ith level ancestor of v).

In Step 2, a tuple $(\mathcal{EDT}, \beta, 0, 0)$ is inserted into $PSet$ for each $\beta \in cover\_gen(r, t)\}$. Here $\mathcal{EDT}$ is the root of the input *ed-tree*. In the example, $PSet$ becomes $\{(\mathcal{EDT}, [-1, 0, 1], 0, 0)$, $(\mathcal{EDT}, [-1, 1, 0], 0, 0)$, $(\mathcal{EDT}, [0, -1, 1], 0, 0)$, $(\mathcal{EDT}, [0, 0, 0], 0, 0)$, $(\mathcal{EDT}, [0, 1, -1], 0, 0)$, $(\mathcal{EDT}, [1, 0, -1], 0, 0)$, $(\mathcal{EDT}, [1, -1, 0], 0, 0)\}$ after Step 2.

Step 3 initializes the *ResultSet* which will contain the final output.

Step 4 iteratively goes through the following four sub-steps until $PSet$ becomes empty. In each iteration, the next available tuple $< v, \beta, l, d > \in PSet$ is retrieved in Step 4(i) and all tuples of the form $< v, \beta', l', d' > \in PSet$ are moved from

Algorithm 3.3.2: *Probe_Search*
input: probe P, edit distance r, target sequence T, *ed-tree*$(w, s, H = [h_1, ..., h_t])$ built on T
output: All locations of the subsequences S indexed by *ed-tree*(w, s, H) such that $edit(P, S) \leq r$
method:

```
1. $PSet \leftarrow \emptyset$; /* a set for storing active nodes */
2. For each $\beta \in cover\_gen(r, t)$
      $PSet \leftarrow PSet + \{< \mathcal{EDT}, \beta, 0, 0 >\}$; /* $\mathcal{EDT}$ is the root of the input ed-tree */
3. $ResultSet \leftarrow \emptyset$;
4. while( $PSet \neq \emptyset$ )
   i.Select the first tuple $< v, \beta, l, d >$ from $PSet$;
   ii.$QSet \leftarrow \{< v', \beta', l', d' > | v' = v \bigwedge$
              $< v', \beta', l', d' > \in PSet\}$;
   iii. $PSet \leftarrow PSet - QSet$;
   iv. for each child node $y$ of $v$
          for each tuple $< v', \beta', l', d' >$ in $QSet$
            if $edit(y.sequence, P^{H+\beta'}[l'+1]) + d' + |\sum_{i=l'+2}^{t} \beta_i| \leq r$ then
              if $l' + 1 < t$ then
                $PSet \leftarrow PSet +$
                  $\{< y, \beta', l'+1, d' + edit(y.sequence, P^{H+\beta'}[l'+1]) >\}$;
              else
                for each pointer $pt$ in $y$ /*$y$ is $t$-th level node */
                  $ResultSet \leftarrow ResultSet + \{pt\}$;
5. Return $ResultSet$;
```

Figure 3.8: Homology search in *ed-tree*(w, s, H)

$PSet$ into $QSet$ from Step 4(ii) to 4(iii). In step 4(iv), each child y of the node v is compared against each tuple $< v, \beta', l', d' >$ in $QSet$ to determine whether y will lead to a leaf in the solution set with respect to $< v, \beta', l', d' >$. This is done by checking whether the total sum of the edit distance between the sequence represented by y and $P^{H+\beta'}[l'+1]$, i.e. $edit(y.sequence, P^{H+\beta'}[l'+1])$, d' and $|\sum_{i=l'+2}^{t} \beta_i|$ is less than or equal to r. If the condition fails, then the leaf nodes

Iteration	PSet	QSet	ResultSet
0	$(\mathcal{EDT},[-1,0,1],0,0),(\mathcal{EDT},[-1,1,0],0,0)$ $(\mathcal{EDT},[0,-1,1],0,0),(\mathcal{EDT},[0,0,0],0,0)$ $(\mathcal{EDT},[0,1,-1],0,0),(\mathcal{EDT},[1,0,-1],0,0)$ $(\mathcal{EDT},[1,-1,0],0,0)$	$\emptyset$	$\emptyset$
1	$(p1,[-1,0,1],1,1),(p1,[-1,1,0],1,1)$ $(p1,[0,-1,1],1,1),(p1,[0,0,0],1,1)$ $(p1,[0,1,-1],1,1),(p1,[1,0,-1],1,1)$ $(p1,[1,-1,0],1,1)$	$(\mathcal{EDT},[-1,0,1],0,0),(\mathcal{EDT},[-1,1,0],0,0)$ $(\mathcal{EDT},[0,-1,1],0,0),(\mathcal{EDT},[0,0,0],0,0)$ $(\mathcal{EDT},[0,1,-1],0,0),(\mathcal{EDT},[1,0,-1],0,0)$ $(\mathcal{EDT},[1,-1,0],0,0)$	$\emptyset$
2	$(p1.2,[1,0,-1],2,1)$	$(p1,[-1,0,1],1,1),(p1,[-1,1,0],1,1)$ $(p1,[0,-1,1],1,1),(p1,[0,0,0],1,1)$ $(p1,[0,1,-1],1,1),(p1,[1,0,-1],1,1)$ $(p1,[1,-1,0],1,1)$	$\emptyset$
3	$\emptyset$	$(p1.2,[1,0,-1],2,1)$	{22}

Figure 3.9: Processing for the example in step 4

under y is not a active node with respect to $< v, \beta', l', d' >$ and the next pair of node-tuple comparison will be processed. If the condition holds and y is a leaf node, then all the pointers at y will be added to the result set. Otherwise, a new tuple $< y, \beta', l'+1, d'+edit(y.sequence, P^{H+\beta'}[l'+1]) >$ will be inserted in the front of $PSet$ for the next round of processing where the children of y will be searched for the next level of active nodes. Figure 3.9 depicts the iterations in step 4 of the example. In iteration 1, node $p2$ is pruned and there exist 7 values of β which can make $p1$ a active node. In iteration 2, the only active node is $p1.2$ while $p1.1$ and $p1.3$ are pruned. This is because we can see that for node $p1.2$,

$$edit(p1.2.sequence, P^{H+\beta'}[2]) + d' + |\beta'_3|$$
$$= edit(\text{GCTTAC,GCTTAC}) + edit(p1.sequence, P^{H+\beta'}[1]) + 1$$
$$= edit(\text{GCTTAC,GCTTAC}) + edit(\text{GGTAGCG,GGTAGG})) + 1$$
$$= 0 + 1 + 1 = 2$$

In iteration 3, $p1.2.1$ is again verified to be a active node and the result leaf containing a pointer to location 22 of the sequence is appended to $ResultSet$. Finally, in Step 5, $ResultSet$ which contains all the valid locations will be returned.

Level	ed-tree(18,*,(6,6,6)) r=3	ed-tree(18,*,(6,6,6)) r=2
1	63.5082%	90.8696%
2	99.4152%	99.7760%
3	99.8828%	99.7555%

Note: skip interval does not influence the experiment result here.

Figure 3.10: Pruning Rate

3.3.3 Analysis And Experimental Evaluation of Pruning Effect

We carry out an experiment to analyze the effect of the sequence pruning in each level of the *ed-tree*. 2,000,000 sequences pairs with the length of 18 are randomly selected from the esthuman.z database. In the pruning of level l, sequence pair (S_1, S_2) will be remained only if there exists $\beta = [\beta_1, ..., \beta_t]$ to satisfy $\sum_{i=1}^{l} edit({S_1}^H[i], {S_2}^{H+\beta}[i]) \leq r$. For level l, $1 \leq l \leq 3$, the pruning rate, μ_l, can be formalized as follow:

$$\mu_l = \frac{\textit{the number of the sequence pairs remained in level } l}{\textit{the number of the sequence pairs checked in level } l}$$

Figure 3.10 shows the pruning rates in the different levels. After the processing of the first two levels, 1-(1-μ_1)*(1-μ_2) of sequence pairs are pruned. For example, we consider the pruning rate after the first two levels. Only fewer than 0.03%(0.22%) sequence pairs needed to be checked in the level 3 for $r = 2(r = 3)$. It greatly speeds up the homology searching. The time complexity is $O(\sum_{i=1}^{3}\prod_{j=0}^{i-1}\mu_j \cdot 4^{\beta_{j+1}})$ where $\mu_0 = 1$.

3.3.4 Detecting Proper Setting

The settings in *pmodel* and *ed-tree* significantly determine the efficiency and effectiveness of the index and search method. These settings include:

$$\begin{cases} w & \textit{length of probe} \\ s & \textit{skip interval} \\ r & \textit{edit distance range} \\ H = [h_1, ..., h_t] & \textit{segment length vector} \end{cases}$$

In order to determine a probe model with good sensitivity for a given computer, we firstly randomly select sequence pairs that with ED-Similarity greater or no less than 0.5 from the given DNA sequence database. Secondly we apply multiple probe models on the selected pairs and calculate the proportion of hits. Herein, a hit on a pair means that at least one probe on this pair generates edit distance less than the threshold. The higher hit rate, the more sensitive the probe model is. For example, we conduct sensitivity study on est_human genome with the 120,000 sequence pairs which are mentioned in the introduction. In order for clarity, we only presented the comparison among Blast-11, *pmodel*$(18, 2, 2)$ and *pmodel*$(18, 3, 2)$ in Figure 3.1 and 3.2. More results are observed and summarized in Table 3.3.4 where Blast-11 is used as a benchmark. With edit distance $r = 2$, we see *pmodel*$(18, 3, 2)$ and *pmodel*$(18, 2, 2)$ become more sensitive with the increasing of ED-similarity. Compared to Blast-11, they generate less trivial hits on the sequence pairs with ED-Similarity less than 0.6. While they produce more hits on the sequence pairs with ED-similarity higher than 0.7. We also notice that *pmodel*$(20, 1, 3)$ and *pmodel*$(20, 2, 3)$ are better choices than Blast-11 if users would like to detect more hits on sequence pairs with ED-similarity higher than 0.6. Hence, a proper *pmodel* for the dataset on the given computer can be identified

ED-Similarity	0.5	0.6	$[0.7, 1]$
Sensitivity			
1	$pmodel(18, 3, 2)$	$pmodel(18, 3, 2)$	Blast-11
2	$pmodel(18, 2, 2)$	$pmodel(18, 2, 2)$	$pmodel(18, 3, 2)$
3	$pmodel(20, 2, 3)$	Blast-11	$pmodel(18, 2, 2)$
4	$pmodel(20, 1, 3)$	$pmodel(20, 2, 3)$	$pmodel(20, 2, 3)$
5	Blast-11	$pmodel(20, 1, 3)$	$pmodel(20, 1, 3)$
6	$pmodel(18, 3, 3)$	$pmodel(18, 3, 3)$	$pmodel(18, 3, 3)$
7	$pmodel(18, 2, 3)$	$pmodel(18, 2, 3)$	$pmodel(18, 2, 3)$

Table 3.1: Sensitivity and ED-simialrity

according to the user's expectation.

Given a DNA sequence database, the settings of *ed-tree* determine the index size. In order to avoid too high cardinality of the cover generator to speed up the edit distance calculation, the number of segments in one probe should be no more than 3 and the edit distance range should be no more than 3 either.

On the one hand, the check-up table for edit distance should be fully loaded into memory. With edit distance range no more than 3, two bits can store one entry of the edit distance checkup table. For an example, with segment vector length $[6, 6, 6]$ and edit distance range 3, we should calculate the edit distances for all DNA sequence pairs with lengths of (5,6), (6,6) and (6,7). Hence, the check-up tables require $\frac{4^{11}+4^{12}+4^{13}}{4}$ bytes, i.e., 21MB in memory. On the other hand, the indices of level 1 and 2 should be fully loaded into memory also. In case of large DNA sequence databases, all 6-character DNA sequences would happen. Therefore, the pointers in indices of level 1 can be implied by the order of entries. Indices of level 2 include 4^{6+6} entries each of which includes a 4-byte pointer to its first index entry in level 3 and a 2-byte short figure to record the number of its entries in level 3. Hence, the index size of level 1 and 2 is $4^{6+6} \times (4 + 2)$ bytes, i.e., 96MB. The remainder of memory is used to record the temporary results during the search process.

According to the above analysis, given 1GB memory, the total length of the first two segments can be 13 which leads to an index size of $4^{13} \times (4+2)$ bytes, i.e., 384MB. Assume *pmodel*$(20, 2, 3)$ is adopted, the segment length vector should be either $[7, 6, 7]$ or $[6, 7, 7]$. In order to prune more in the level 1, $[7, 6, 7]$ is selected.

3.4 Performance Study

We implemented the *ed-tree* in C. All our experiments are done on a PC with a Pentium 4 1.6Ghz CPU, 256MB of SDRAM and a 7200rpm 20GB harddisk running Windows XP. Two databases are used for our experiments, esthuman.z and estother.z which were both downloaded from the NCBI website. The databases are composed from the alphabets {A,C,G,T,N} with N representing a wildcard i.e. N can be any one of the four alphabets. After removing the wildcard character, the estother.z database contains 2.07G bases while the esthuman.z database contains 1.55G bases. Because N is a wildcard and the edit distance allows “replace”, “insert” and “delete”, with regardless of N, *pmodel* still allows N to appear at most r times in the hit where r is the edit distance range adopted in *pmodel*.

3.4.1 Datasets

Figure 3.11 shows the sizes of various *ed-tree*s that are built on the two databases with the different parameters. We note that the first two levels of the *ed-tree* take up at most 96MB of storage and could easily be stored in the main memory. The third level of the *ed-tree* on the other hand will require storage of up 3GB.

s	H	Dataset	Level-1,2(MB)	Level-3(MB)
2	[6,6,6]	est_human	96	2720
2	[6,6,6]	est_other	96	3036
2	[6,5,7]	est_human	24	2713
2	[6,5,7]	est_other	24	3003
3	[6,6,6]	est_human	96	2075
3	[6,6,6]	est_other	96	2317
3	[6,5,7]	est_human	24	2068
3	[6,5,7]	est_other	24	2304

Figure 3.11: *ed-tree* Index Sizes, $w = 18$

3.4.2 Comparing The *ed-tree* with *Blastn*

Our focus here is to compare the efficiency of *ed-tree* against the latest version of Blastn(NCBI Blastn2) which is available from the NCBI website. The sensitivity of the probe models that are supported by the *ed-tree* is in fact comparable to Blastn as demonstrated in an earlier section. We conducted two sets of experiments to compare the efficiency of the *ed-tree* against Blastn by varying the database size and the query length. In the experiments, the length of probe sequence w is set to 18 and H is set to $[6, 6, 6]$.

The first set of experiments is designed to evaluate how the performance of Blastn and *ed-tree* varies with the database size. Three databases of varying sizes are used. The first two are the esthuman.z and estother.z databases while the third is a 'hybrid" dataset containing 2.3G bases, which is created by combining the whole esthuman.z and a part of estother.z. 1000 query subsequences, each with length of 250 bases, are randomly selected from the DNA of yeast(Saccharomyces cerevisiae) and the average time for answering the query is taken. As shown in Figure 3.12, the increase in query time for Blastn is much more significant than the *ed-tree* algorithms as Blastn suffers from its high I/O cost in large sequence databases. For $r = 2$, the *ed-tree* can be faster than Blastn by a factor of up to 6.

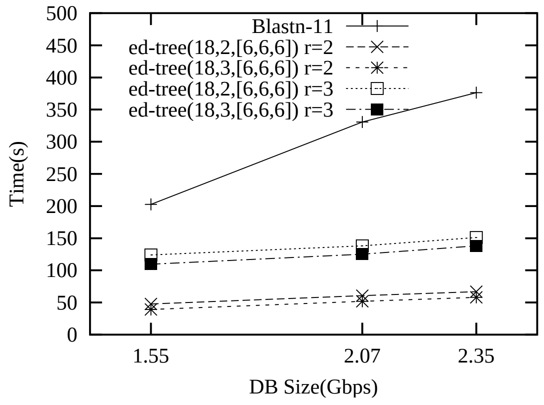


Figure 3.12: Speed vs DB Size (Query length=250)

For $r = 3$, the *ed-tree* still outperforms Blastn when the database size increased. We believe that the *ed-tree* will be much more capable in handling large DNA sequence databases than Blastn.

Another set of the experiments is carried out to investigate how the query length influences the efficiency of the *ed-tree* compared to Blastn. We randomly select 1000 subsequences with length varying from 30 to 250 bases from the yeast DNA. For each query length, we take the average query time and plot them against the query length. Figure 3.13 and 3.14 depict the relationship between the query time of the various algorithms versus the query length for the two databases. The query time of *ed-tree* increases linearly with the query length since the number of probes in the *ed-tree* grows linearly. As such, there are likely to be more hits on different parts of the databases which will incur more I/Os for the search. We note that in general, indexes will not be useful if most parts of the database have to be accessed and this is also applicable in the case of the *ed-tree*.

Blastn's performance is not significantly affected by the length of the query since its running time is dominated by the I/O time of its sequential scan which will not increase substantially for the longer queries. Notwithstanding, Blastn's performance is typically an order of magnitude slower than the *ed-tree*'s.

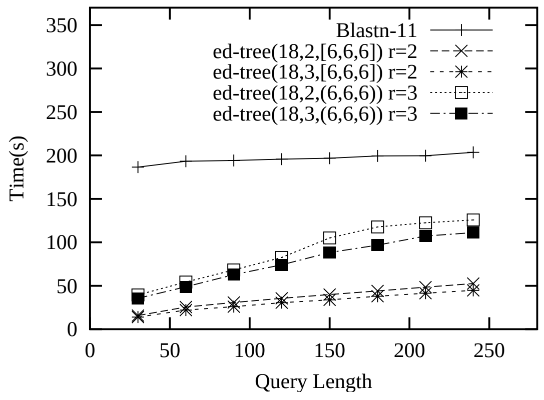


Figure 3.13: DB:est_human 1.55Gbps

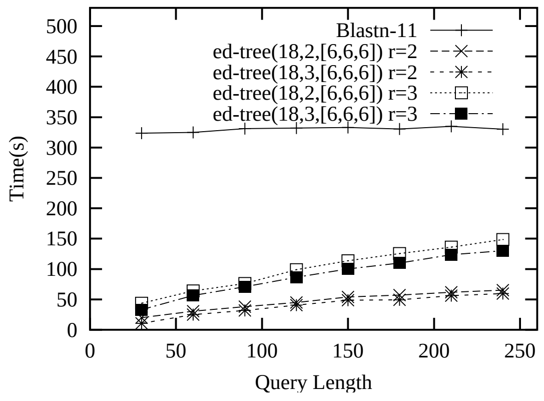


Figure 3.14: DB:est_other 2.07Gbps

3.4.3 Pruning Cost Analysis

We will next look at the effect of parameter settings on the *ed-tree* 's performance. Since there exist a large number of combinations for the parameter values of the *ed-tree*, we can only provide more insight for the more important ones.

From Figure 3.13 and 3.14 which are shown earlier, we make the following observations on the effect of r and s:

1. Query time of the *ed-tree* increases with increasing r

   Increasing r, the edit distance range means that a more relaxed query constraint is being specified. Naturally, this means that a larger result set will be returned. From the difference between the result of $r = 2$ and $r = 3$, we can tell that a difference of one edit operation can change the size of the result set substantially causing a significant increase in I/Os.

2. Query time of *ed-tree* decreases with increasing s

   Since increasing the skip interval s decreases the number of subsequences being indexed in the database, the number of I/Os operations for the search will also decrease. The effect of varying s is however less significant than r as it can be seen from Figure 3.13 and 3.14.

3.4.4 Effect of Parameters

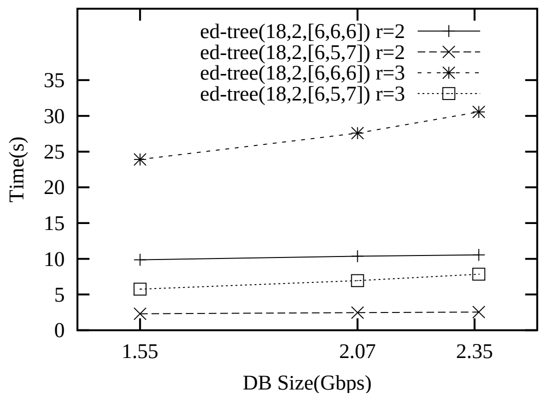


Figure 3.15: Level 1,2 Pruning time vs DB Size

We look at how the segment length vector H affects the performance of *ed-tree*, more specifically the pruning processes at the first and second level of the *ed-tree*. Figure 3.15 shows that the pruning time at the first two levels grows moderately with the database size for a query length of 250. One interesting observation here is that the *ed-tree*$(18, 2, [6, 6, 6])$ took more time for pruning than *ed-tree*$(18, 2, [6, 5, 7])$ regardless of the value of r. This is because there are significantly less nodes in the second level when H is set to $[6, 5, 7]$. For $H = [6, 6, 6]$, there are 4^{6+6}(=16, 777, 216) nodes in level 2, compared to 4^{6+5}(=4, 194, 304) nodes for $H = [6, 5, 7]$. At the third level however, r plays a more important role as shown in Figure 3.16 where the search time for $r = 3$ is always significantly higher than that for $r = 2$. In this case, the I/Os required for retrieving a larger result set dominate the cost.

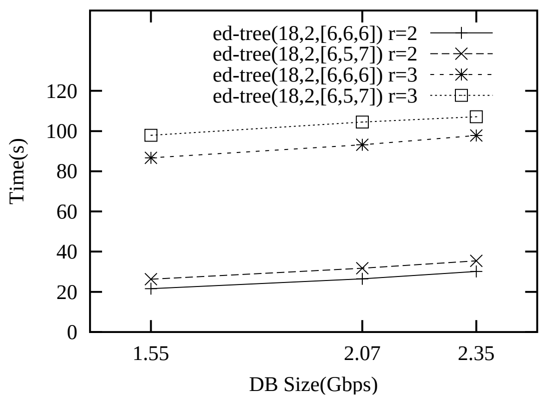


Figure 3.16: Level 3 Pruning time vs DB Size

Likewise, Figure 3.17 and 3.18 confirm that H has a more significant effect on the first two level of pruning while r has more effect on the third level pruning. Overall, since the I/Os at the third level dominate the total pruning time for the three levels, it makes sense to try to adjust H such that a small increase in pruning time at the first two levels causes a substantial decrease in the I/O time at the third level. In fact, this is the main principle we adopted in fine tuning H to achieve the best performance.

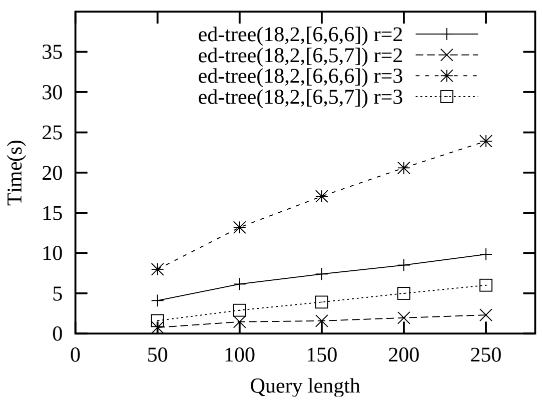


Figure 3.17: Level 1,2 Pruning time vs Query Length

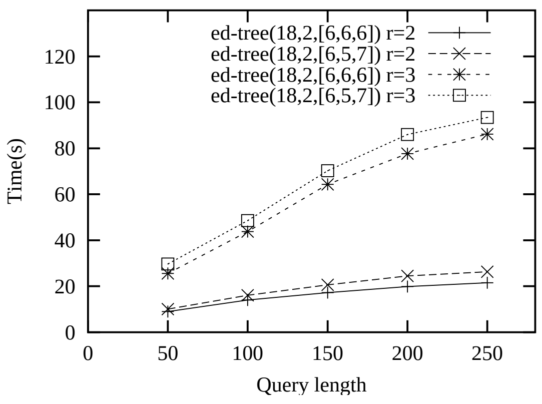


Figure 3.18: Level 3 Pruning time vs Query Length

3.5 Summary

The growing interest in genomic research has caused an explosive growth in the size of DNA databases making it increasingly challenging to perform searches on them. In this chapter, we proposed a model called pmodel for generally evaluating the effectiveness of the probing methodology. It helped us select suitable parameters to obtain the different searching requirements.

An index structure called the *ed-tree* was designed for supporting fast and effective homology searches on DNA databases. The *ed-tree* is developed to enable probe-based homology search algorithms like Blastn which generate short probe strings from the query sequence and then match them against the sequence database in order to identify potential regions of high similarity to the query sequence. Determining whether two sequences have edit distance no more than a given range is novelly transformed into computing the edit distances between their corresponding segments. Based on *ed-tree*, a fast algorithm is developed to do homology search which can guarantee the efficiency and sensitivity especially for detecting insertions and deletions. It supports more flexible probe model with longer probes and more relaxed matching. As a consequence, the *ed-tree* is not only more effective and efficient than the latest Blastn(NCBI Blast2) when supporting homology search but also takes up moderate storage compared to existing data structures like the suffix tree. The *ed-tree* can be also applied to protein sequences with different *pmodel* and index configurations. In FCDR system which will be discussed in Chapter 7, we use *ed-tree* for indexing protein sequences where the $pmodel(8, 1, 2)$ and $ed\text{-}tree(8, 1, [4, 4])$ are adopted.

According to our experimental results, to index a DNA database of 2 giga base pairs(Gbps), *ed-tree* only takes less than 3Gb of secondary storage which is easily handled by a desktop PC. The query time using *ed-tree* is up to 6 times lower than Blastn for large DNA sequence databases and this performance gap grows with the size of the DNA sequence database. Unlike previous sequence indexes, the size of the *ed-tree* is at most 3Gb for a sequence database of 2 billion base pairs. Considering that the mapped human genome contains around 3 billion base pairs, we believe that the *ed-tree* is well positioned for searching large DNA sequence database on a desktop computer.

CHAPTER 4

Substructure Clustering in Sequential 3D Object Datasets

4.1 Introduction

With an increasing number of new applications related to sequential 3D objects, such as 3D structural pattern retrieval on protein structures and pattern mining in the tracks of cell phones, clustering substructure on the sequential 3D object dataset becomes an important and meaningful approach that can be applied: (1) in structure-based drug design. Common substructures can help to understand the working of living organisms [28] and can be used to detect active binding sites to target organisms. (2) in protein remote homology detection. It is widely agreed that the similarities among distantly related proteins are often preserved at the level of their 3D structures, even when very little similarity remains at the sequence level [35]. In the absence of obvious sequence similarity, it is important for clustering substructures on the sequential linked C_α atoms to explore common substructures on remote homologous proteins. (3) in tracking moving objects in GSM and GPS systems. Similar tracks in a given time frame could be used to identify the relationships among moving objects.

There are many algorithms on subspace clustering for various datasets which include itemsets, sequences, trees, lattices and graphs [4, 46, 77, 88, 78, 95]. However, the existing techniques are no longer effective on sequential structures with 3D coordinates since they are mainly based on value similarity and pattern similarity instead of 3D structure similarity where translation and rotation should be considered. This has prompted us to study the subspace clustering on sequential 3D objects.

Traditionally, the minimized *root mean square distance* (*rmsd*) which considers all possible rotations and transformations is adopted to evaluate the dissimilarity between two 3D structures. Various measurements to detect clusters [4, 20, 22, 88, 95] have been defined according to applications and datasets. In order to facilitate fast comparison, we devise a simple but effective measurement, *feature difference summation* (*fds*), based on the summation of the difference on selected features on vertices.

In this chapter, we establish a model, *sCluster*, to describe the problem and to define the non-trivial clusters as *maximal sCluster* to avoid the clusters which can be contained by other clusters. Leveraging on the simplicity of *fds*, an efficient algorithm is devised for mining pairwise *maximal sCluster*s on two sequential 3D objects and a modified apriori algorithm is developed to expand the pairwise *maximal sCluster*s with respects to both the cardinality and the length. We also extend the approach to support query, i.e., to incrementally generate the *maximal sCluster*s only related to a given new object. Experiments have been conducted to study the impacts of settings to performance and query efficiency. Due to the absence of existing subspace clustering methods on sequential 3D objects, we create an *rmsd*-based clustering as the benchmark to evaluate the performance of our approach. Furthermore, randomly selected *sCluster*s in protein chains are plotted to show the effectiveness.

4.2 Definition And theory

4.2.1 Sequential 3D object

A sequential 3D object is a group of vertices where each vertex only links to its left-side and right-side adjacent vertices. It can be represented as $S[b : e]$, where S is a sequential 3D object, and b and e are the beginning vertex and the ending vertex respectively. Every vertex is a 3D coordinate. Fig 4.1 shows 3 sequential 3D objects: $P[1 : 8]$, $Q[1 : 9]$ and $R[1 : 10]$, which include 8, 9 and 10 vertices respectively.

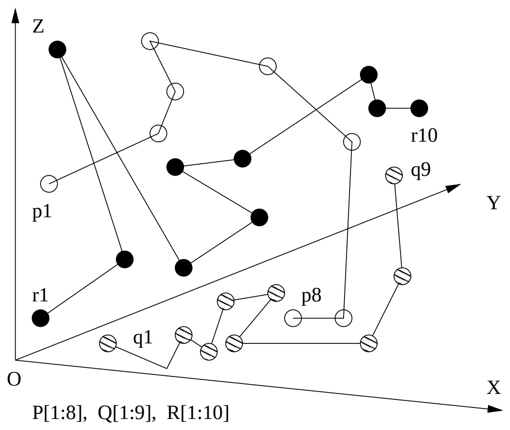


Figure 4.1: Example of sequential 3D objects

4.2.2 Similarity Evaluation

Traditionally, $rmsd$ is used to evaluate the similarity between two structures. Given two sequential 3D objects, $S[1:n]$ and $P[1:n]$, $rmsd$ is defined as:

$$rmsd(S[1:n], P[1:n]) = min\{\sqrt{\sum_{i=1}^{n} |T(S[i]) - P[i]|^2}\}$$

Here T is the isometric transformation with rotations and translations. To find the optimal matching from one structure to the other, optimal vertices alignments, rotations and translations should be identified before calculating the distance. For two sequential 3D structures, the alignment is determined by the order of vertices. In real-life datasets like protein chains, the typical number of vertices ranges from hundreds to thousands. Computing $rmsd$ is a time-consuming process. This promotes us to define a simple but effective distance measurement, called $feature\ difference\ summation$ (fds):

$$\begin{aligned} fds(S_1[1:n], S_2[1:n]) = & \quad \sum_{i=1}^{k}\sum_{j=1}^{n} \frac{|f_1[i][j] - f_2[i][j]|}{max(|f_1[i][j]|, |f_2[i][j]|)} \\ k: & \quad the\ number\ of\ features\ on\ each\ vertex \\ n: & \quad the\ number\ of\ vertices\ on\ one\ object \\ f_x[i][j]: & \quad the\ value\ of\ i^{th}\ feature\ on\ j^{th}\ vertex\ of\ S_x \end{aligned}$$

On every vertex, k features are extracted. The fds is defined as the summation of the difference on all features on all vertices. To evaluate the dissimilarity of two sequential 3D objects, on the vertex $S[i]$, we select three features:

1. $l[i]$: the edge length between $S[i-1]$ and $S[i]$

2. $a[i]$: the angle between the edge linking $S[i-1] - S[i]$ and the edge linking $S[i] - S[i+1]$

3. $t[i]$: the torsion angle between the plane fixed by $S[i-1]-S[i]-S[i+1]$ and the plane fixed by $S[i]-S[i+1]-S[i+2]$

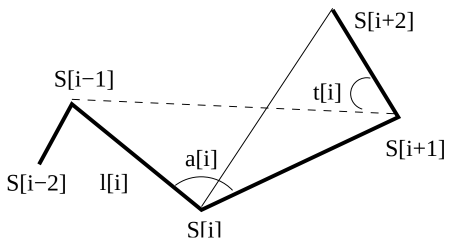


Figure 4.2: Features on $S[i]$: $l[i]$, $a[i]$ and $t[i]$

Fig 4.2 depicts the three features, $l[i]$, $a[i]$ and $t[i]$.

On each vertex, we calculate the difference of selected features and the feature differences over two vertices can be calculated independently. Furthermore, fds between two structures is the summation of the feature differences on all vertices. Therefore fds could be incrementally computed. In the study [83], we have proposed the measurement *ald*. Compared to *ald*, fds takes torsion angels between adjacent planes into account.

In order to study the effectiveness of fds, we compare fds to *ald* with the benchmark of *rmsd* on real-life dataset. The algorithm in [13, 58] was used to calculate *rmsd*. In protein 3D structures, the chain of C_α atoms basically describes the protein 3D structure and can be represented as a sequential 3D object although hydrogen and disulfide bonds also play important roles. We randomly download protein chains from SCOP database [63], and link the C_α atoms in each protein chain to be a sequential 3D object. From these protein chain 3D objects, we randomly selected two groups as datasets: D_1 and D_2. The D_1 includes 50 pairs of objects each of which contains 60 vertices where the objects are longer than 60 vertices have been truncated. The D_2 includes 50 pairs of objects each of which contains 40 vertices where the objects are longer than 40 vertices have been truncated.

Without loss of generality, we calculate *rmsd*, *ald* and *fds* and normalize the values by linearly mapping the values to the interval $[0, 1]$ and plot the normalized values of *rmsd*, *ald* and *fds* in Fig 4.3 and 4.4. We can observe *fds* approaches *rmsd* better than *ald* does in both D_1 and D_2. Table 4.2.2 illustrates the correlations between *rmsd* and *fds* in D_1 and D_2 are 0.63 and 0.32 which are significantly higher than the correlations between *rmsd* and *ald*. In the later section, some interesting clustering results will be presented to show the effectiveness. The *fds* is a summation of feature difference on vertices. In other applications, we suggest to extract different features and give features different weights to describe the structure in the best manner.

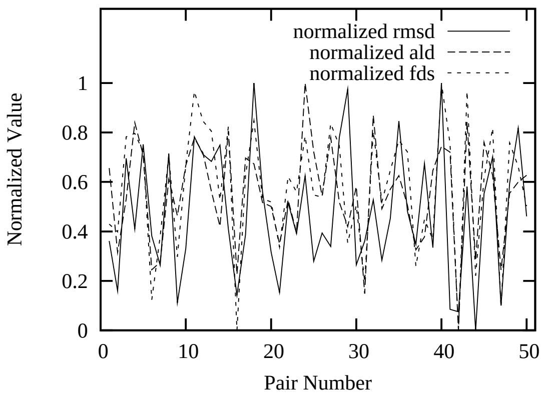


Figure 4.3: Comparison of fds, ald and $rmsd$ in D_1

Dataset	$Correlation(rmsd, ald)$	$Correlation(rmsd, fds)$
D_1	0.42	0.63
D_2	0.25	0.32

Table 4.1: Correlation comparison

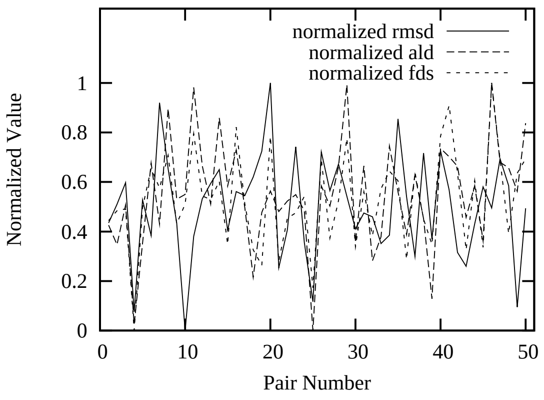


Figure 4.4: Comparison of fds, ald and $rmsd$ in D_2

4.2.3 *sCluster*

Given a sequential 3D dataset D and a maximum distance ε, *sCluster* is defined as:

1. *sCluster* includes a group of objects.
2. All objects include the same number of vertices.
3. Every object appears as a whole or a portion of a sequential 3D object in D.
4. For every two objects, $S[b_s : e_s]$ and $P[b_p : e_p]$, $fds(S[b_s : e_s], P[b_p : e_p]) \leq \varepsilon$ holds.

The cardinality of *sCluster* is defined as the number of objects which are included by the *sCluster*. The length of *sCluster* is defined as the number of vertices of any object which is included by the *sCluster*. One group of objects can be contained by another group of objects in terms of cardinality or length.

1. Given two groups of objects, C and C', if $C \subseteq C'$, we say C is contained by C' in terms of cardinality. As shown in Fig 4.5, $C = \{S_2[20 : 35], S_4[25 : 40]\}$ is contained by $C' = \{S_2[20 : 35], S_4[25 : 40], S_5[7 : 22]\}$ in terms of cardinality.

2. Given two groups of objects, $C = \{S_1[b_1 : e_1], ..., S_n[b_n : e_n]\}$ and $C'' = \{S_1[b''_1 : e''_1], ..., S_n[b''_n : e''_n]\}$, if $b_i - b''_i = b_j - b''_j$ and $e_i - e''_i = e_j - e''_j$ holds where $1 \leq i < j \leq n$, we say C'' is synchronized to C with the offset of $[b''_1 - b_1, e''_1 - e_1]$. Alternatively, we define an operator Π as $C'' = \Pi(C, [b''_1 - b_1, e''_1 - e_1])$.

   In addition, if $b''_1 - b_1 \leq 0$ and $e''_1 - e_1 \geq 0$, then we say C is contained by C'' in terms of length. For instance, $C'' = \{S_2[15 : 40], S_4[20 : 45]\}$ is synchronized to $C = \{S_2[20 : 35], S_4[25 : 40]\}$ with the offset $[-5, 5]$, i.e., $C'' = \pi(C, [-5 : 5])$. Furthermore, C is contained by C'' in terms of length.

Property 4.2.1 *Assume C is a sCluster and C' is a group of objects which is contained by C in terms of cardinality or length, then C' is a sCluster.*

Property 4.2.2 *Assume $C = \{S_1[b_1 : e_1], ..., S_n[b_n : e_n]\}$ is a sCluster with maximum distance ε and $C' = \{S_1[b'_1 : e'_1], ..., S_n[b'_n : e'_n]\} = \pi(C, [b, e])$. Let $Cr = \{S_1[br_1 : er_1], ..., S_n[br_n : er_n]\}$ where $[br_i : er_i] = [b_i : e_i] \cap [b'_i : e'_i]$ and $i \in [1 : n]$. Then Cr is a sCluster contained by C in terms of length and Cr is synchronized to C'.*

Proof: *Assume $b \geq 0$ and $e \leq 0$, then $br_i = b'_i$ and $er_i = e'_i$ where $i \in [1 : n]$ and C' is contained by C in terms of length. Thus, we have $Cr = C'$ which is a sCluster contained by C and is synchronized to C'. Similarly, we can prove the case when $b \leq 0$ and $e \geq 0$.*

Assume $b \geq 0$ and $e \geq 0$, then $br_i = b'_i$ and $er_i = e_i$ where $i \in [1 : n]$. Thus we have $br_i - b'_i = br_j - b'_j = 0$ and $er_i - e_i = er_j - e_j = 0$ where $1 \leq i < j \leq n$. Hence, $er_i - e'_i = er_i - (e_i + e) = -e = er_j - (e_j + e) = er_j - e'_j$. Therefore, $Cr = \pi(C', [0, -e]) = \pi(C, [b, 0])$ which is obviously a sCluster contained by C in terms of length and is synchronized to C'. Similarly, we can prove the case when $b \leq 0$ and $e \leq 0$.

In all 4 possible cases, the property holds.

Property 4.2.3 *With maximum distance ε, C, C' and C'' are sClusters which are obtained from object datasets A, A' and A'' respectively. Assume $A \cap A' = \emptyset$, $A \cap A'' = \emptyset$ and $A' \cap A'' = \emptyset$. If $C \cup C'$, $C \cup C''$ and $C' \cup C''$ are sClusters, then $C \cup C' \cup C''$ is also a sCluster.*

Proof: *For every two substructures from $C \cup C' \cup C''$, $S_i[b_i : e_i]$ and $S_j[b_j : e_j]$, we can find them from one of the sClusters: C, C', C'', $C \cup C'$, $C \cup C''$ or $C' \cup C''$. Then we have $fds(S_i[b_i : e_i], S_j[b_j : e_j]) \leq \varepsilon$. Hence, the property holds.*

Property 4.2.4 *Given a sCluster, C, assume the length is L, the minimum sCluster length is w and the cardinality is u. There are $\frac{(L-w+1)\times(L-w+2)}{2} \times (2^u - u - 1)$ sClusters that are contained by C in terms of cardinality or length.*

Proof: *On the one hand, without considering empty set and u sClusters where each sCluster includes only one substructure, there are $2^u - u - 1$ sClusters that are contained by C in terms of cardinality. On the other hand, for each sCluster, $SubC$, contained by C in terms of cardinality, there are $\frac{(L-w+1)\times(L-w+2)}{2}$ sCluster that are contained by $SubC$ in terms of length. Thus, $\frac{(L-w+1)\times(L-w+2)}{2} \times (2^u - u - 1)$ sClusters are contained by C.*

In order to avoid trivial results, we define *maximal sCluster* as the *sCluster* that cannot be contained by any other *sCluster*.

As shown in Fig 4.5, T is a dataset of 5 objects and $C = \{S_2[20 : 35], S_4[25 : 40]\}$ is a *maximal sCluster*. Since C is contained by C' in terms of cardinality and C is a *maximal sCluster*, C' cannot be a *sCluster* on T. Because C is contained by C'' in terms of length, we conclude C'' is not a *sCluster*.

Problem 4.2.1 *Given sequential 3D dataset T, we are going to find all maximal sClusters with minimum length w, minimum cardinality u and maximum distance*

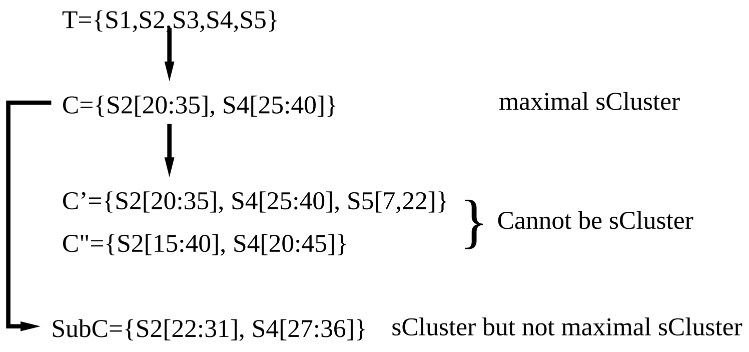

T={S1,S2,S3,S4,S5}
C={S2[20:35], S4[25:40]}
maximal sCluster
C'={S2[20:35], S4[25:40], S5[7,22]}
C"={S2[15:40], S4[20:45]}
Cannot be sCluster
SubC={S2[22:31], S4[27:36]}
sCluster but not maximal sCluster

Figure 4.5: Example of *maximal sCluster*

ε.

In this chapter, we focus on solving this problem efficiently and accurately. According to our knowledge, there is no method addressing this problem, while it often occurs especially in the recent bioinformatics and biopharmaceutical research.

Lemma 4.2.1 *Given $C = \{S_1[b_1 : e_1], ..., S_n[b_n : e_n]\}$ and $C' = \{S_1[b'_1 : e'_1], ..., S_n[b'_n : e'_n]\}$, where $C' = \Pi(C, [b, e])$, and both $C \cup \{P[b_p : e_p]\}$ and $C' \cup \{Q[b_q : e_q]\}$ are sClusters with maximum distance ε. Let $Cr = \{S_1[br_1 : er_1], ..., S_n[br_n : er_n]\}$ where $[br_i : er_i] = [b_i : e_i] \cap [b'_i : e'_i]$ and $i \in [1 : n]$. If there exists $\{P[br_p : er_p], Q[br_q : er_q]\}$ which is a sCluster synchronized to $\{P[b_p - b_1 + br_1 : e_p - e_1 + er_1], Q[b_q - b'_1 + br_1 : e_q - e'_1 + er_1]\}$. Let*

$$\begin{cases} [b'_p : e'_p] = [br_p : er_p] \cap [b_p - b_1 + br_1 : e_p - e_1 + er_1] \\ [b'_q : e'_q] = [br_q : er_q] \cap [b_q - b'_1 + br_1 : e_q - e'_1 + er_1] \end{cases}$$

Then

$$\Pi(C', [b'_q - b_q, e'_q - e_q]) \cup \{P[b'_p : e'_p]\} \cup \{Q[b'_q : e'_q]\}$$

is a sCluster with maximum distance ε.

Proof: First, according to the definition, we know $b_q - b'_q < 0$ and $e_q - e'_q > 0$. Then $\pi(C', [b'_q - b_q, e'_q - e_q])$ is a *sCluster* contained by C' in terms of length.

Second, since $\{P[br_p : er_p], Q[br_q : er_q]\}$ is synchronized to $\{P[b_p - b_1 + br_1 : e_p - e_1 + er_1], Q[b_q - b'_1 + br_1 : e_q - e'_1 + er_1]\}$, according to Property 4.2.2, $\{P[b'_p : e'_p], \{Q[b'_q : e'_q]\}$ is a *sCluster* contained by $\{P[br_p : br_p], Q[br_q : br_q]\}$ in terms of length and $b'_p - b'_q = (b_p - b_1 + br_1) - (b_q - b'_1 + br_1) = b_p - b_q + b$ and $e'_p - e'_q = (e_p - e_1 + er_1) - (e_q - e'_1 + er_1) = e_p - e_q + e$.

Third, $\Pi(C', [b'_q - b_q, e'_q - e_q]) \cup \{P[b'_p : e'_p]\} = \Pi(C, [b'_q - b_q + b, e'_q - e_q + e]) \cup \{P[b'_p : e'_p]\} = \bigcup_{i=1}^{n} \{S_i[b_i + b'_q - b_q + b : e_i + e'_q - e_q + e]\} \cup \{P[b'_p : e'_p]\}$. Since $(b_i + b'_q - b_q + b) - b'_p = b_i - b_q + b - (b_p - b_q + b) = b_i - b_p$ and $(e_i + e'_q - e_q + e) - e'_p = e_i - e_q + e - (e_p - e_q + e) = e_i - e_p$, $\Pi(C', [b'_q - b_q, e'_q - e_q]) \cup \{P[b'_p : e'_p]\}$ is synchronized to $C \cup \{P[b_p : e_p]\}$. Besides, with $b'_p \geq b_p$ and $e'_p \leq e_p$, we know that $\Pi(C', [b'_q - b_q, e'_q - e_q]) \cup \{P[b'_p : e'_p]\}$ is a *sCluster* contained by $C \cup \{P[b_p : e_p]\}$ in terms of length.

Fourth, as $Q[b'_q : e'_q] = Q[b_q + b'_q - b_q : e_q + e'_q - e_q]$, we know $\pi(C', [b'_q - b_q, e'_q - e_q]) \cup \{Q[b'_q : e'_q]\} = \pi(C' \cup \{Q[b_q : e_q]\}, [b'_q - b_q, e'_q - e_q])$. Since $b'_q \geq b_q - b'_1 + br_1 \geq b_q$ and $e'_q \leq e_q - e'_1 + er_1 \leq e_q$, $\pi(C', [b_q - b'_q, e_q - e'_q]) \cup \{Q[b'_q : e'_q]\}$ is a *sCluster* contained by $C' \cup \{Q[b_q : e_q]\}$.

Finally, according to Property 4.2.3, we know $\Pi(C', [b'_q - b_q, e'_q - e_q]) \cup \{P[b'_p : e'_p]\} \cup \{Q[b'_q : e'_q]\}$ is a *sCluster*. Lemma is proved.

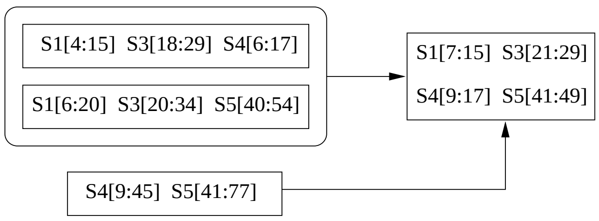


Figure 4.6: Sample of Lemma 4.2.1

In Fig 4.6, $\{S_1[4 : 15], S_3[18 : 29], S_4[6 : 17]\}$ and $\{S_1[6 : 20], S_3[20 : 34], S_5[40 : 54]\}$ imply that $\{S_1[6 : 15], S_3[20 : 29], S_4[8 : 17]\}$ and $\{S_1[6 : 15], S_3[20 : 29], S_5[40 : 49]\}$ are *sCluster*s with maximum distance ε. By combining them with the *sCluster* of $\{S_4[9:45], S_5[41:77]\}$, we have a new *sCluster*, $\{S_1[7:15], S_3[21:29], S_4[9:17], S_5[41:49]\}$.

4.3 Algorithms

Our algorithms include the pairwise *maximal sCluster* mining and a modified apriori mining process to generate *maximal sCluster* with the cardinality more than 2.

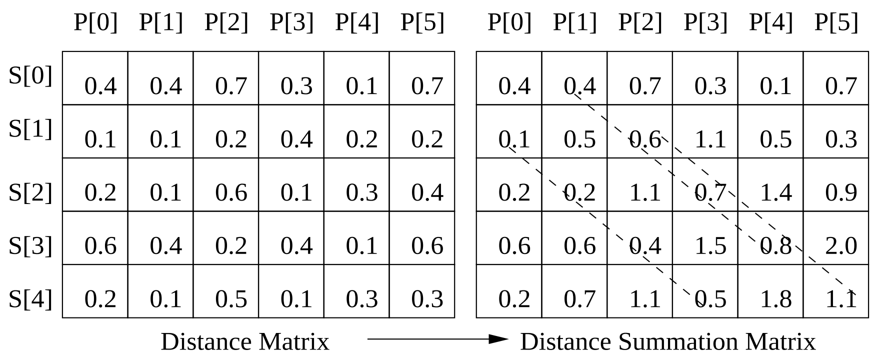
The first challenge is to cluster on different groups of dimensions for every sequential substructure. Similar substructures could appear in any object, and single structure may contain many vertices, for example one protein chain includes $C\alpha$ atoms ranging from hundreds to thousands.

The second challenge is to efficiently and effectively compute the distance between different substructures. Unlike the normal measurement such as Manhattan Distance and Euclidean Distance, the similarity is related to optimal rotation and translation, and is traditionally determined by *rmsd* which is computationally expensive. We have proposed fds to overcome the problem.

Finally, our method is deterministic without loss of qualified clusters. This significantly increases the computation cost due to combinations.

4.3.1 Mining Pairwise *Maximal sCluster*

Mining *maximal sClusters* on two objects is the basis for generating *sClusters*. Given S and P, without loss of generality, assume the length of S, m, is same with that of P. The algorithm to find the *sClusters* on $\{S, P\}$ is shown in Algorithm 4.3.1.



	P[0]	P[1]	P[2]	P[3]	P[4]	P[5]
S[0]	0.4	0.4	0.7	0.3	0.1	0.7
S[1]	0.1	0.1	0.2	0.4	0.2	0.2
S[2]	0.2	0.1	0.6	0.1	0.3	0.4
S[3]	0.6	0.4	0.2	0.4	0.1	0.6
S[4]	0.2	0.1	0.5	0.1	0.3	0.3

	P[0]	P[1]	P[2]	P[3]	P[4]	P[5]
S[0]	0.4	0.4	0.7	0.3	0.1	0.7
S[1]	0.1	0.5	0.6	1.1	0.5	0.3
S[2]	0.2	0.2	1.1	0.7	1.4	0.9
S[3]	0.6	0.6	0.4	1.5	0.8	2.0
S[4]	0.2	0.7	1.1	0.5	1.8	1.1

Figure 4.7: Example of pairwise *maximal sClusters*

Mining pairwise *maximal sCluster*
Input: two objects $S[1:m], P[1:m]$, maximum distance ε, minimum *sCluster* length w
Output: *maximal sCluster*

```
/* Compute distance matrix */
1. for i=0 to $m-1$
2.     for j=i to $m-1$
3.         $fds[i,j] = fds[j,i] = fds(S[i], P[j])$
4. $result \leftarrow \emptyset$
```

```
/* Search in up-triangle in distance summation matrix */
5. for j=0 to $m-w-2$
6.     for r=0 to $m-1-j$:     sum[r]=$\sum_{t=0}^{r} fds(t, j+t)$
7.     $\varepsilon' \leftarrow \varepsilon$
8.     for i=0 to $m-1-j-w$ /* Search in each diagonal */
9.             Binary search $k$ where $sum[k] = max\{sum[u]$ where $sum[u] \leq \varepsilon$ and $0 \leq u \leq m-1-j-w\}$
10.        $\varepsilon' \leftarrow \varepsilon' + fds[i, j+i]$
11.        If $k \geq w-1$ then $result \leftarrow result \cup \{S[i:i+k], P[j:j+k]\}$
```

```
12. Search in down-triangle similar with step 5-11
13. Return $result$
```

In Fig 4.7, $fds[i,j]$ in distance matrix represents the difference summation among all the corresponding features on $S[i]$ and $P[j]$. We convert it to distance summation matrix by summing up the distances by each diagonal. Assume maximum distance ε is 1.0. There are three *maximal sClusters*, $\{S[1:4], P[0:$

$3]\}, \{S[0:3], P[1:4]\}$, and $\{S[1:4], P[2:5]\}$.

The computation complexity of calculating the distance matrix and distance summation matrix is $O(m^2)$. In Step 8-11, the search of pairwise *maximal sClusters* on the diagonal which crosses i cells incurs i binary searches in 1, 2,..., i cells respectively. The complexity of this search is $O(\lg 1 + \lg 2 + ... + \lg i)$. Hence, the computational cost of searching pairwise *maximal sClusters* in up-triangle is $O(\sum_{i=1}^{m} \lg i!)$. Therefore, we summarize the total computational complexity of mining pairwise *maximal sClusters* as below:

$$O(m^2 + \textstyle\sum_{i=1}^{m} \lg i!) \text{ i.e. } O(m^2 \lg m) \text{ where } m \text{ is object length.}$$

Based on the pairwise *maximal sClusters*, we can gradually generate all the *sClusters* with larger cardinality using a modified apriori algorithm.

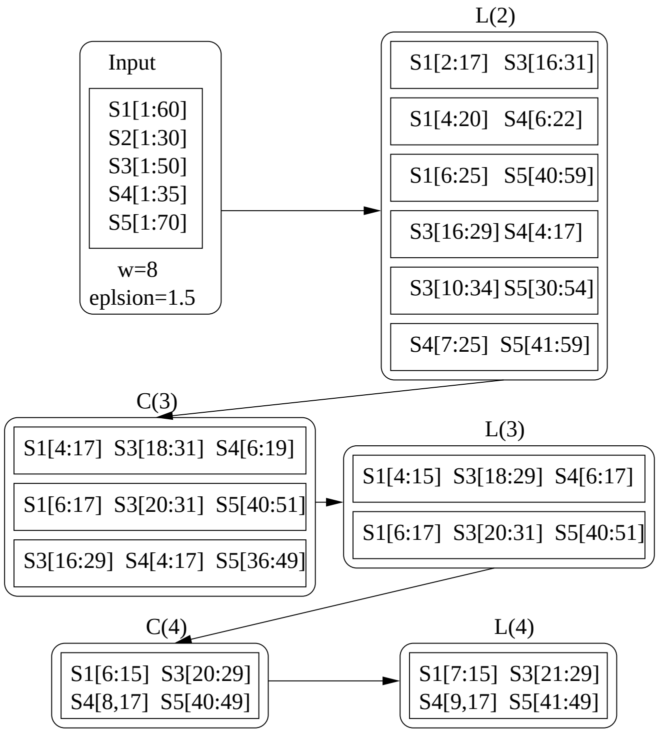

Input
S1[1:60]
S2[1:30]
S3[1:50]
S4[1:35]
S5[1:70]
w=8
eplsion=1.5
L(2)
S1[2:17] S3[16:31]
S1[4:20] S4[6:22]
S1[6:25] S5[40:59]
S3[16:29] S4[4:17]
S3[10:34] S5[30:54]
S4[7:25] S5[41:59]
C(3)
S1[4:17] S3[18:31] S4[6:19]
S1[6:17] S3[20:31] S5[40:51]
S3[16:29] S4[4:17] S5[36:49]
L(3)
S1[4:15] S3[18:29] S4[6:17]
S1[6:17] S3[20:31] S5[40:51]
C(4)
S1[6:15] S3[20:29] S4[8,17] S5[40:49]
L(4)
S1[7:15] S3[21:29] S4[9,17] S5[41:49]

Figure 4.8: Example of Algorithm 4.3.1

Mining *maximal sCluster*
Input: dataset $T = \{S_1, S_2, ..., S_n\}$, maximum distance ε, minimum length w
Output: *maximal sClusters*

```
/* Generate pairwise maximal sClusters */
1. $L(2) \leftarrow \bigcup_{1 \le i < j \le n} \{pairwise\ maximal\ sClusters\ on\ (S_i, S_j)\}$
---
/* Generate candidates $C(k+1)$ from $L(k)$ */
2. k=2
3. while $(L(k) \neq \emptyset)$
4.     $C(k+1) \leftarrow \emptyset$
5.     For any $E_1, E_2 \in L(k)$
6.       $E_1 = \{S_1[b_1 : e_1], ..., S_{k-1}[b_{k-1} : e_{k-1}], S_x[b_x : e_x]\}$
7.       $E_2 = \{S_2[b'_1 : e'_1], ..., S_{k-1}[b'_{k-1} : e'_{k-1}], S_y[b_y : e_y]\}$
8.       If $\{S_1[b_1 : e_1], ..., S_{k-1}[b_{k-1} : e_{k-1}]\}$ synchronizes to $\{S_2[b'_1 : e'_1], ..., S_{k-1}[b'_{k-1} : e'_{k-1}]\}$
9.       Then $[br_i, er_i] \leftarrow [b_i, e_i] \cap [b'_i, e'_i]$, $i \in [1 : k-1]$
10.           If $er_1 - br_1 \geq w$
11.            Then $newCandidate \leftarrow \{S_1[br_1 : er_1], ..., S_{k-1}[br_{k-1} : er_{k-1}]\}$
                     $\cup\{S_x[b_x + br_1 - b_1 : e_x + er_1 - e_1], S_y[b_y + br_1 - b'_1 : e_y + er_1 - e'_1]\}$
12.                 $C(k+1) \leftarrow C(k+1) \cup \{newCandidate\}$
---
/* Look up $L(2)$ to refine each candidate in $C(k+1)$ */
13.    $L(k+1) \leftarrow \emptyset$
14.      For $E = \{S_1[br'_1 : er'_1], ..., S_{k-1}[br'_{k-1} : er'_{k-1}], S_x[br'_x : er'_x], S_y[br'_y : er'_y]\} \in C(k+1)$
15.      If there exists $\{S_x[b''_x : e''_x], S_y[b''_y : e''_y]\} \in L(2)$ which synchronizes to $\{S_x[br'_x : er'_x], S_y[br'_y : er'_y]\}$
16.      Then $[br''_x : er''_x] \leftarrow [br'_x : er'_x] \cap [b''_x : e''_x]$
17.           $[br''_y : er''_y] \leftarrow [br'_y : er'_y] \cap [b''_y : e''_y]$
18.           If $er''_x - br''_x \geq w$
19.           Then $newsCluster \leftarrow \{S_x[br''_x : er''_x], S_y[br''_y : er''_y]\}$
                $\cup\{S_1[br'_1 + br''_x - br'_x : er'_1 + er''_x - er'_x], ..., S_{k-1}[br'_{k-1} + br''_x - br'_x : er'_{k-1} + er''_x - er'_x]\}$
20.                 $L(k+1) \leftarrow L(k+1) \cup \{newsCluster\}$
21.    $k \leftarrow k+1$
---
/* Mark redundant sClusters */
22. For each $B \in L(i)$, $i \in [2 : k]$
23.    If there exists $B' \in L(k+1)$ and $B \subset B'$
24.    Then mark $B$ as redundant sCluster
---
25. Return $\bigcup_{i=2}^{k} \{x : x \in L(i)$ and $x$ is not a redundant sCluster $\}$
```

A sample is shown in Fig 4.8 where the dataset includes 5 objects, maximum distance $\varepsilon = 1.5$ and minimum length $w = 8$. After computing pairwise *maximal sCluster*, the algorithm produces $L(2)$ which includes 6 *sCluster*s. We observe the *sCluster*s, $\{S_1[2:17], S_3[16:31]\}$ and $\{S_1[4:20], S_4[6:22]\}$, and $\{S_1[2:17]\} = \Pi(\{S_1[4:20]\}, [-2,-3])$. From step 6 to 12, we generate candidate $\{S_1[4:17], S_3[18:31], S_4[6:19]\}$ which is added into $C(3)$. During the refining process from step 13 to 21, we check the pairwise *maximal sCluster* from S_3 and S_4. We note $\{S_3[16:29], S_4[4:17]\}$ is synchronized to $\{S_3[18:31], S_4[6:19]\}$. Therefore, we refine the candidate $\{S_1[4:17], S_3[18:31], S_4[6:19]\}$ to the *sCluster* $\{S_1[4:15], S_3[18:29], S_4[6:17]\}$ which is added into $L(3)$. Similarly, we produce $C(4)$ and $L(4)$.

When generating candidates from $L(k)$ to $C(k+1)$, every two *sCluster*s have been checked if their first $k-1$ objects are synchronized. The computational complexity for generating $C(k+1)$ from $L(k)$ is

$$O(|L(k)|^2), \quad \text{where } |L(k)| \text{ is the number of } sCluster \text{ in } L(k)$$

When refining candidate from $C(k)$ to $L(k)$, the $L(2)$ is frequently accessed. We sort the *sCluster* of $L(2)$ in the order of the object IDs and establish a look-up table to speed up a single checking process to $O(\frac{|L(2)|}{n^2})$ where n is the number of objects. The computational complexity for refining $C(k)$ to $L(k)$ is:

$$O(\frac{|L(2)|}{n^2} \times |C(k)|), \text{ where } |C(k)| \text{ is number of candidates in } C(k) \text{ and } n \text{ is number of objects}$$

Property 4.3.1 *Algorithm 4.3.1 generates all maximal sClusters without false positive and without false negative.*

Proof: *On one hand, $L(2)$ includes all pairwise maximal sClusters. For any sCluster $B = \{S_1[b_1:e_1], S_2[b_2:e_2], S_3[b_3:e_3]\}$, there must exist sClusters $P = \{S_1[bp_1:ep_1], S_2[bp_2:ep_2]\}$, $Q = \{S_2[bq_2:eq_2], S_3[bq_3:eq_3]\}$ and $R =$*

$\{S_1[br_1 : er_1], S_3[br_3 : er_3]\}$ *where*

$$\begin{cases} [b_1 : e_1] \subseteq [bp_1 : ep_1] \text{ and } [b_1 : e_1] \subseteq [br_1 : er_1] \\ [b_2 : e_2] \subseteq [bp_2 : ep_2] \text{ and } [b_2 : e_2] \subseteq [bq_2 : eq_2] \\ [b_3 : e_3] \subseteq [bp_3 : ep_3] \text{ and } [b_3 : e_3] \subseteq [br_3 : er_3] \end{cases}$$

According to Lemma 4.2.1, Algorithm 4.3.1 will generate B and put it into $L(3)$. Similarly, we can deduce that all sClusters with k objects will be in $L(k)$. On the other hand, those sClusters which could be contained by any other sClusters are marked as redundant sClusters. Thus, we know all maximal sClusters are found.

4.3.2 Query Related *sCluster*s

With the growth of the number of applications related to sequential 3D data and the data size, it is often required to conduct query of the clusters which appear on a new object. The problem could be defined as:

Problem 4.3.1 *Assume the sClusters on dataset T with maximum distance ε are generated. Given an object $Q[1 : m]$, we are going to find the maximal sClusters related to Q, i.e., the maximal sClusters in $T \cup \{Q[1 : m]\}$ with maximum distance ε where every sCluster includes one object appearing in $Q[1 : m]$.*

Query related *maximal sClusters*
Input: Dataset $T = \{S_1, S_2, ..., S_n\}$, maximum distance ε, minimum *sCluster* length w, *sCluster* lists on T: $L(2)$, $L(3)$,..., $L(m)$, query object S_{n+1}
Output: *maximal sCluster* on $T + \{S_{n+1}\}$ related to S_{n+1}

```
1. $A(2) \leftarrow \bigcup_{1 \le i \le n} \textit{pairwise maximal sCluster on}(S_i, S_{n+1})$
2. k=2
3. while $(A(k) \neq \emptyset)$
4.     Generate candidates $C'(k+1)$ from $A(k) \cup L(k)$ similar with step 2-12 in Algorithm 4.3.1 where every candidate includes one object appearing in $S_{n+1}$.
5.     Look-up $A(2)$ to refine each candidate in $C'(k+1)$ to produce $A(k+1)$ similar with step 13-21 in Algorithm 4.3.1.
6. Mark redundant sClusters in $\bigcup_{i=1}^{k} A(i)$ similar with step 22-24 in Algorithm 4.3.1.
7. Return $\bigcup_{i=2}^{k} \{x : x \in A(i)$ and $x$ is a maximal sCluster $\}$
```

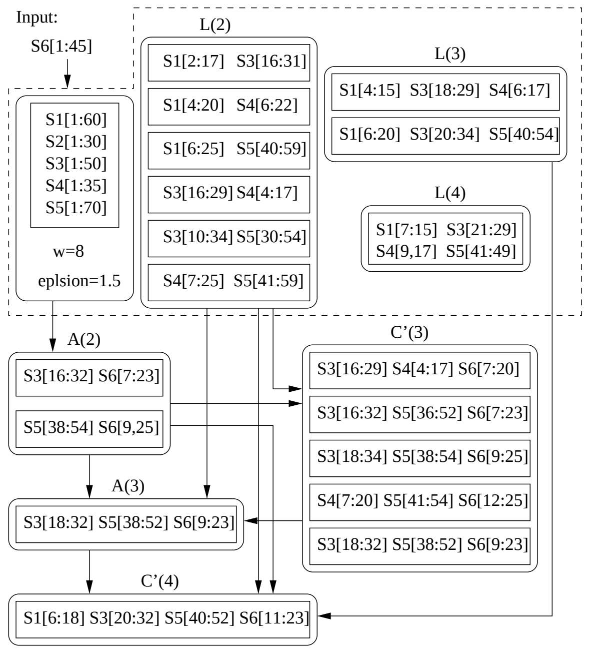


Figure 4.9: Example of Algorithm 4.3.2

Algorithm 4.3.2 is to find the *sClusters* related to the given object. Fig 4.9 is a sample to explain Algorithm 4.3.2. Here, $T = \{S_1, ..., S_5]$, and $L(2)$, $L(3)$ and $L(4)$

are the existing lists of *sCluster*s with cardinalities of 2, 3 and 4 respectively. Step 1 calculates pairwise *maximal sCluster*s between the query object, $S_6[1:45]$ and the objects in T. $A(2)$ is the list of qualified pairwise *maximal sCluster*s. It includes $\{S_3[16:32], S_6[7:23]\}$ and $\{S_5[38:54], S_6[9:25]\}$. We check *sCluster*s in $A(2)$ with *sCluster*s in $L(2)$ to see if they could be combined, and check if *sCluster*s in $A(2)$ can be combined each other. For example, $\{S_3[16:32], S_6[7:23]\}$ in $A(2)$ and $\{S_3[16:29], S_4[4:17]\}$ in $L(2)$ can generate a candidate $\{S_3[16:29], S_4[4:17], S_6[7:20]\}$ in $C'(3)$. Similarly, we produce the other four candidates in $C'(3)$. After that, we check candidates in $C'(3)$ to see if they are qualified. $\{S_3[16:29], S_4[4:17], S_6[7:20]\}$ is invalid due to the absence of *sCluster*s between S_4 and S_6. $\{S_3[16:32], S_5[36:52], S_6[7:23]\}$ becomes $\{S_3[18:32], S_5[38:52], S_6[9:23]\}$ after refining with $\{S_5[38:54], S_6[9:25]\}$ in $A(2)$. We notice that it is same as the last candidate in $C'(3)$. After that, we check *sCluster*s between $L(3)$ and $A(3)$ for $C'(4)$. Only one candidate $\{S_1[16:18], S_3[20:32], S_5[40:52], S_6[11:23]\}$ is produced by combining $\{S_3[18:32], S_5[38:52], S_6[9:23]\}$ in $A(3)$ and $\{S_1[60:20], S_3[20:34], S_5[40:54]\}$ in $L(3)$. However, it is invalid as there is no *sCluster*s between S_1 and S_6. Until now, all new *sCluster*s are generated in $A(2)$ and $A(3)$. Trivial *sCluster*s are marked in step 4. Finally, *maximal sCluster*s in $A(2) \cup A(3)$ are returned.

4.4 Experiments

We implement the algorithms in C++. All our experiments are done on a PC with a Pentium IV 2.6Ghz CPU, 1GB of SDRAM and an 80GB hard disk running Windows. Two groups of datasets are used. One is a group of protein chains from SCOP databases. The other is a group of synthetic chains where the vertices are

3D coordinates ranging from 0 to 1. First, we study the effects of parameters to the performance of mining *sCluster*s. Second, we present the response speed of query related *sCluster*s with a given object. Third, we compare *sCluster* method with *rmsd−based Clustering*. Finally, we plot randomly selected *sCluster* results from real datasets including HIV protein chains.

4.4.1 Effect of Parameters

The parameters of *sCluster* include object length, number of objects, minimum *sCluster* length w, and maximum distance ε.

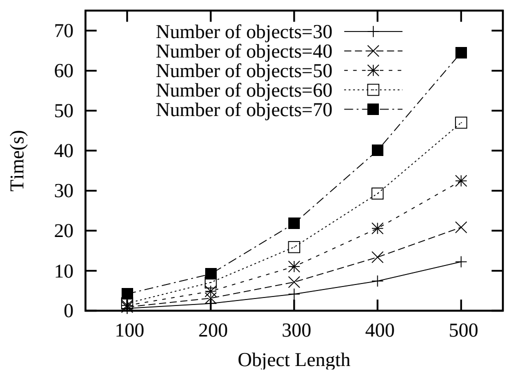


Figure 4.10: Object length VS. Clustering time

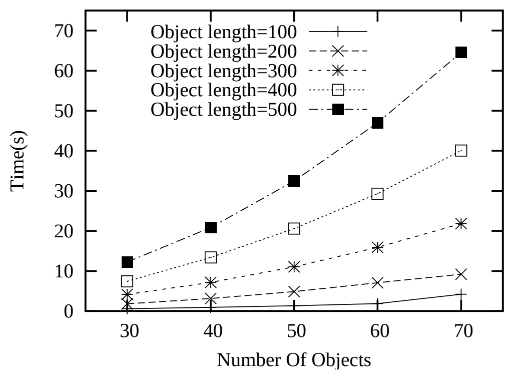


Figure 4.11: Number of objects VS. Clustering time

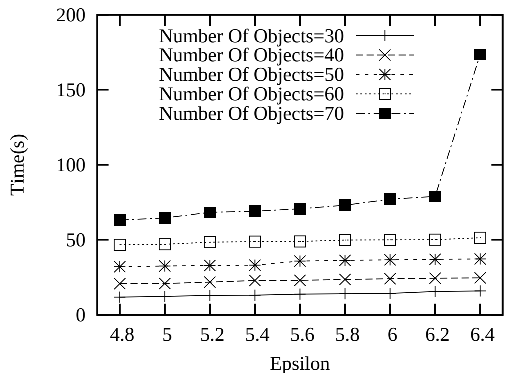


Figure 4.12: ε VS. Clustering time

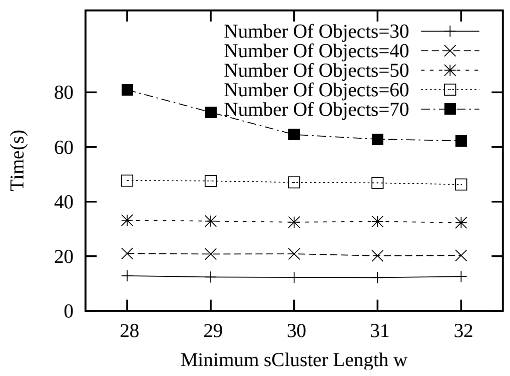


Figure 4.13: w VS. Clustering time

On the one hand, we study the clustering time of *sCluster* in terms of the object length on protein chain datasets. We set $w = 30$, $\varepsilon = 5$, and vary the object length from 100 to 500. In Fig 4.10, we can observe that, as the object length increases, the clustering time increases in a quadratic manner. This is because most of the clustering time is spent on mining pairwise *maximal sClusters* and it accords with the computational complexity analysis. On the other hand, we plot the clustering time with respect to the number of objects in Fig 4.11. We note the clustering speed degrades with the increase of dataset size since there are more distance matrix calculations and more candidates for a larger dataset.

We fix the object length at 500, $w = 30$ and plot the clustering time in Fig 4.12 when we vary ε from 4.8 to 6.4. We observe that *sCluster* shows superior scalability with ε when $4.8 \leq \varepsilon \leq 6.2$. This is because the pairwise *sCluster* mining dominates the candidate generation and refining process.

When $\varepsilon = 6.4$ and number of objects becomes 70, the clustering speed slumps down. This is because the number of *sCluster*s increases significantly due to the over relaxed ε. In this case, tens millions of *sCluster*s have been generated and they exhaust the memory. Because we expect people would be interested in at most thousands of patterns, the main memory should be large enough to accommodate the meaningful *sCluster*s unless the *sCluster* detection criteria is over relaxed.

Fig 4.13 presents the clustering time by varying the minimum *sCluster* length w from 28 to 32 in 5 protein chain datasets when $\varepsilon = 5.0$ and object length=500. In the datasets with protein chains less than 70, *sCluster* shows good scalability with respect to w. Upon investigation, we find that the execution time mainly depends on pairwise *sCluster* mining that is almost stable in the same dataset. In case of 70 objects with $w = 28$, our approach takes 80.9 seconds that is 11% larger than 72.1 seconds when $w = 29$ because shorter minimum *sCluster* length leads to more *sCluster*s and the costs for generating and refining candidates become significant.

4.4.2 Query *Maximal sCluster*s Related to New Object

We evaluate the query algorithm in protein chains with object length ranging from 100 to 500. Besides the protein chains used in the study on the effect of parameters, we randomly select more protein chains from SCOP database as query objects. Fig 4.14 and 4.15 describe the response time for a query in five datasets which include 30, 40, 50, 60, and 70 protein chains respectively. We study the query performance with respect to object length and number of objects. In all cases, our query approach shows scalable behavior as the query process is an incremental process where only the candidates related to the query object would be generated and refined.

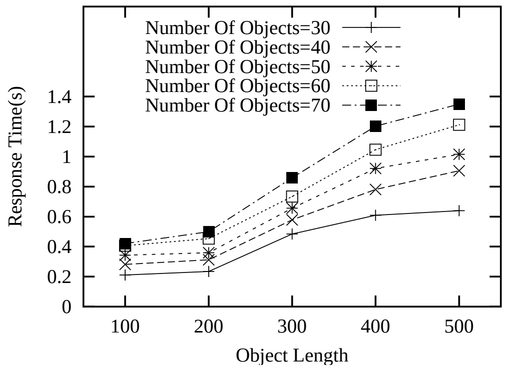


Figure 4.14: Object length VS. Query response time

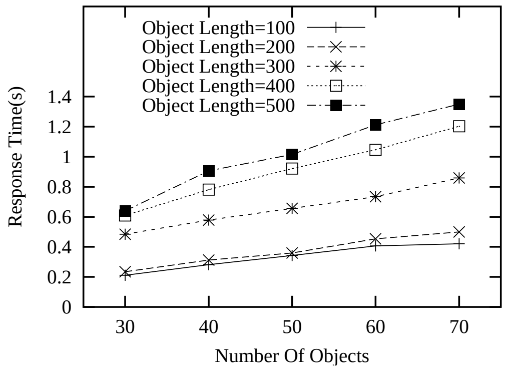


Figure 4.15: Number of objects VS. Query response time

4.4.3 Mining *sCluster*s in Synthetic Datasets

We generate synthetic datasets where the coordinate values of vertices are randomly ranging from 0 to 1. We fix $\varepsilon = 5$ and $w = 30$ while vary the object length and the number of objects. As shown in Fig 4.16 and 4.17, *sCluster* shows similar behaviors in terms of the number of objects and the object length with that in protein chain datasets.

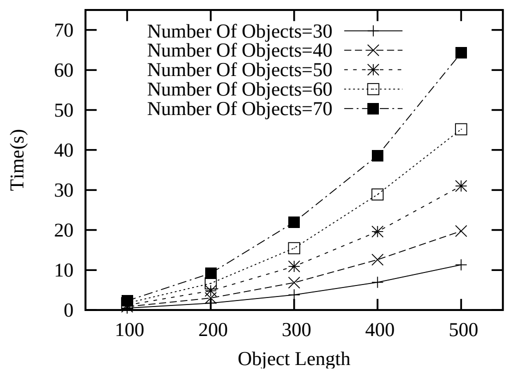


Figure 4.16: Object length VS. Clustering time on synthetic datasets

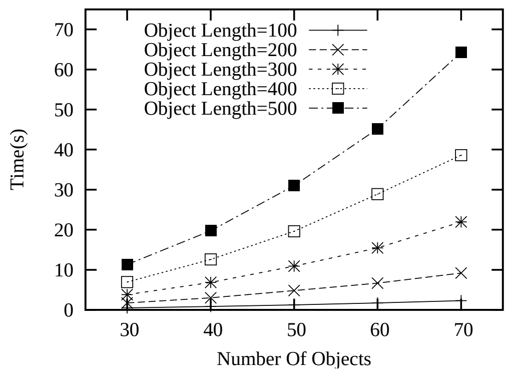


Figure 4.17: Number of objects VS. Clustering time on synthetic datasets

4.4.4 Comparison with *rmsd*-based Clustering

Since *sCluster* model is for mining 3D sequential substructure clusters with shift on dimensions which could not be well supported by the other clustering methods, we design an alternative algorithm based on *rmsd*, called $rmsd-based\ clustering$. In the pairwise *sCluster* mining, we incrementally calculate all the *rmsd*s on every diagonal of distance summary matrix. After that, we adopt the binary-search on determining the pairwise *maximal sCluster*s, i.e., the longest substructure pairs with maximum *rmsd* no more than ε are identified as the pairwise *maximal sCluster* on the diagonal. After computing pairwise *maximal sCluster*s, we use the same method in Algorithm 4.3.1 for generating and refining candidates. Due to the difference between *rmsd* and *fds*, we fix w to be 30 and set ε for $rmsd - based$ $clustering$ to produce *sClusters* which are included by the *sClusters* which are produced by *sCluster* with $\varepsilon = 5.0$.

First, in the experiment for studying scalability on object length, we use 30 protein chains and vary the object length from 100 to 500. Second, in the experiment for studying scalability on the number of objects, we fix the object length to be 200 and conduct experiments in different-sized datasets. Both Fig 4.18 and 4.19 show that *sCluster* outperforms the $rmsd - based$ $clustering$ by magnitudes. This is because $rmsd - based$ $clustering$ calculates multiple *rmsds* on each diagonal on each diagonal of distance summation matrix and the complexity of calculating *rmsd* is significantly higher than *fds*.

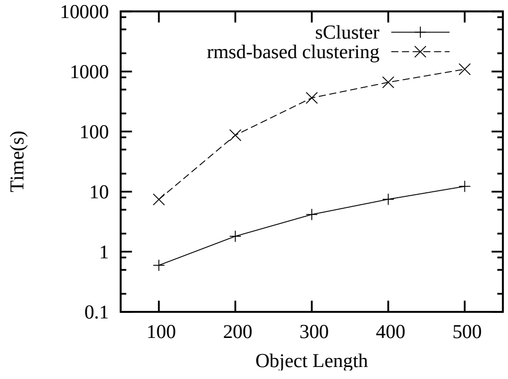


Figure 4.18: *sCluster* VS $rmsd - based$ $clustering$ on object length

4.4.5 Results of *sCluster*

In order to study the effectiveness of *sCluster*, we design 5 different cases with settings listed in Table 5.2 and plot the number of *maximal sClusters* with *sCluster* cardinalities in Fig 4.20. Case 1 is presented as a benchmark and the others are created by changing one of the settings. One observation appearing in all the cases is that the number of *sClusters* initially increases but decreases after a certain

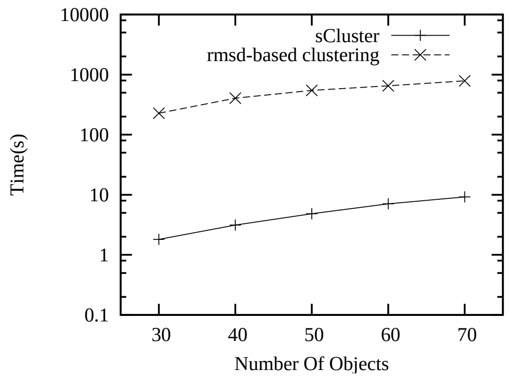


Figure 4.19: *sCluster* VS *rmsd − based clustering* on number of objects

sCluster cardinality. This is because the number of candidates increases during generating stage and the candidates could be filtered during refining stage. The trade-off between generating and refining candidates determines the number of valid *maximal sClusters*.

Parameters	Case 1	Case 2	Case 3	Case 4	Case 5
ε	5.0	5.0	5.0	6.2	5.0
w	30	30	30	30	28
Object length	500	100	500	500	500
Number of objects	70	70	30	70	70
Clustering time(s)	64.5	4.22	12.2	78.9	80.9

Table 4.2: Settings of cases

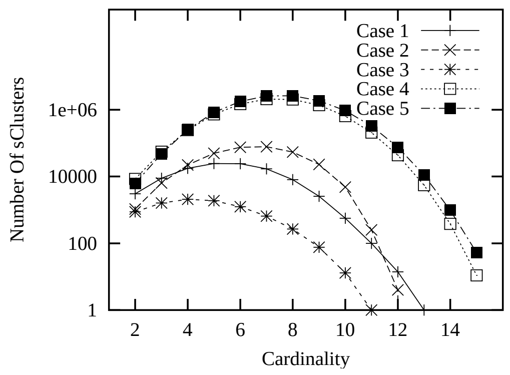


Figure 4.20: Cardinality VS. Number of *sClusters* in 5 cases

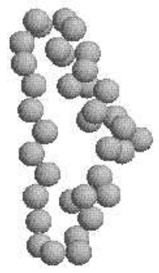


Figure 4.21: $d1mma\_2[150:182]$

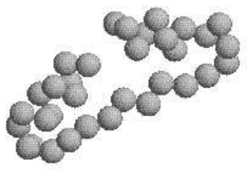


Figure 4.22: $d1d0xa2[151:183]$

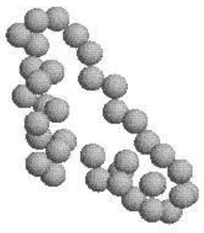


Figure 4.23: $d1d1aa2[157:189]$

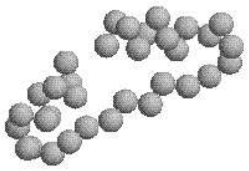


Figure 4.24: $d1d1ca2[159:191]$

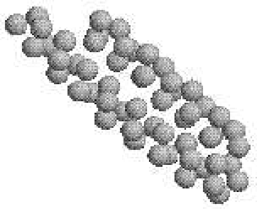


Figure 4.25: $d1b71a1[7:61]$

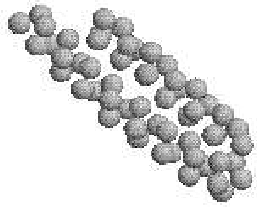


Figure 4.26: $d1bcfa\_[6:60]$

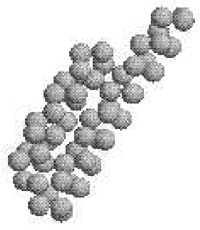


Figure 4.27: $d1euma\_\_[2:56]$

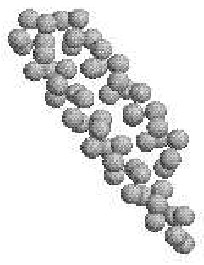


Figure 4.28: $d1jgca\_\_[5:59]$

From the results, we randomly select two *sCluster*s and depict them using *PDB2multiGIF* [47]. The first is from the motor protein domains in chicken pectoral muscles. It includes four objects: $d1mma\_2[150:182]$, $d1d0xa2[151:183]$, $d1d1aa2[157:189]$ and $d1d1ca2[159:191]$ as shown in Fig 4.21, 4.22, 4.23 and 4.24 respectively. The structure seems to be two α-helix structures connected by a structure which is consisted of a group of amino acids located in a line. The second is from the rubrerythrin n-terminal protein domains in ferritin. It includes four objects: $d1b71a1[7:61]$, $d1bcfa\_[6:60]$, $d1euma\_[2:56]$ and $d1jgca[5:59]$ as shown in Fig 5.5.1, 5.5.1, 5.5.1 and 5.5.1 respectively. The structure is a large α-helix with a sharp turn. The similarities among the objects in the two *sCluster*s are obvious. They imply that the results found by *sCluster* are effective and meaningful.

4.4.6 Application in HIV Protein 3D Structures

HIV, a virus that attacks human immune cells, has been firstly discovered on 1981 and spread worldwide to be a major killer now. The disease, acquired immunodeficiency syndrome, caused by HIV is well known by its acronym, AIDS. The structure of HIV protease revealed a crucial fact -like a butterfly, the enzyme is made up of two equal halves. HIV protease has only one such active site - in the center of the molecule where the two halves meet. Traditionally, scientists identify new drugs either by fiddling with existing drugs or by testing thousands of compounds in a laboratory. Using a structure-based strategy, pharmaceutical scientists have an initial advantage. If they could plug this single active site with a small molecule, they could shut down the whole enzyme - and theoretically stop the virus' spread in the body [65]. Structural biology has greatly enhanced researchers' understanding of HIV and has played a key role in the development of drugs to treat this deadly disease.

Therefore, it would be interesting and meaningful to apply *sCluster* to HIV data and investigate the findings. 168 HIV related protein chains listed in SCOP database are downloaded. On average, each protein chain is formed of 103 amino acids. Four cases with different settings as shown in Table 4.4.6 are evaluated where case 1 is considered as a benchmark. In case 2 and case 3, maximum distance ε and minimum *sCluster* length w are varied. Execution time in the first three cases are almost stable because most of the execution time are occupied by pairwise *sCluster* mining. While it is observed that number of *sCluster*s changes significantly with settings. In case 4, there are 100 randomly selected HIV related protein chains with average length of 107. For processing case 4, *sCluster* takes 9.72 seconds which is much faster compared to 16.18 seconds in processing case 1. This is because fewer chains incur fewer calculations on comparing every two chains.

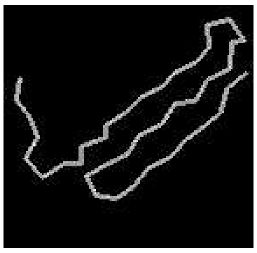


Figure 4.29: $d1c0ua1[431 : 470]$

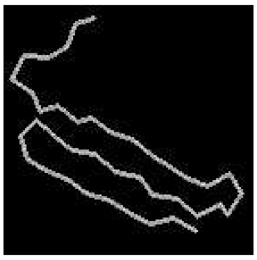


Figure 4.30: $d1c1ca1[431 : 470]$

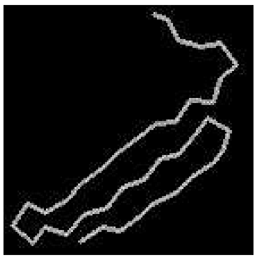


Figure 4.31: $d1jlga1[431 : 470]$

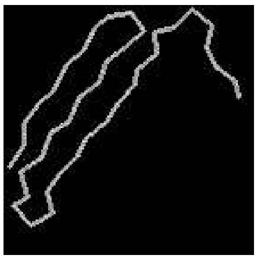


Figure 4.32: $d1rt1a1[431 : 470]$

We randomly plot two *sCluster*s using another popular visualization tool - Chime control. The first *sCluster* includes the substructures as depicted in Figure 4.29, 4.30, 4.31 and 4.32. This *sCluster* appears in the domains of reverse transcriptase of HIV. The second *sCluster* includes the substructures as depicted in Figure 4.33, 4.34, 4.35 and 4.36. It appears in human immunodeficiency virus type 2 protease. The similarities among the substructures in each *sCluster* can be easily observed. These substructure patterns frequently appear in HIV organisms. Therefore, as long as the chemical compound to bind one substructure in *sCluster*s can be identified, the binding would happen on many similar sites and the chemical compound consequently becomes a potential fatal killer to HIV.

Case	# of proteins	Average length	ε	w	Time(s)	# of *sCluster*s
1	168	103	4.0	40	16.18	824
2	168	103	5.0	40	16.74	12741
3	168	103	4.0	38	16.37	4005
4	100	107	4.0	40	9.72	578

Table 4.3: Experiments on HIV dataset. Note: we present number of *sCluster*s where each includes 4 similar substructures.

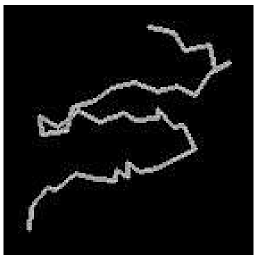


Figure 4.33: $d1hiia\_[1:40]$

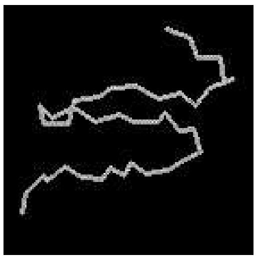


Figure 4.34: $d1idaa\_[1:40]$

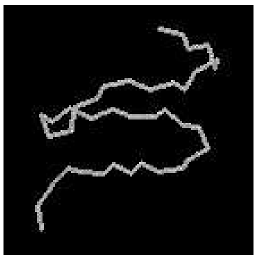


Figure 4.35: $d1idbb\_[1:40]$

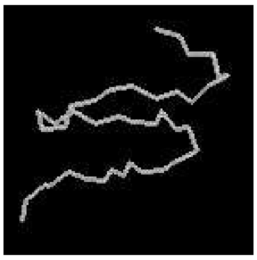


Figure 4.36: $d1idab\_[1:40]$

4.5 Summary

In this chapter, we study a new problem, clustering similar substructure on sequential 3D objects such as protein chains. We propose a *sCluster* model to support these applications. An simple but effective distance measurement, fds, is designed to evaluate the dissimilarity between sequential 3D objects. We devise a fast algorithm for discovering pairwise *maximal sCluster*s between two objects. A modified-apriori algorithm is presented to expand pairwise *maximal sCluster*s to obtain all *maximal sCluster*s without loss. The algorithm is further extended to produce *maximal sCluster*s which are related to a given query object.

Compared to the existing subspace clustering approaches, we differentiate *sCluster* by two points. The first is that *sCluster* focuses on mining clusters which could be located on different dimension group. The other is that *sCluster* is specialized for 3D sequential substructures where translation and rotation are involved during distance measuring.

The applications of *sCluster* model range widely from bioinformatics, biopharmaceutical research and moving-object relationship detection. Especially, with the explosion of protein 3D structures and structure pattern mining, *sCluster* would be a potential and ideal tool. In these applications, substructures may not be close while they are similar after isometric transformations and they are not necessarily in the same dimension groups. With distance measurement fds and *sCluster* algorithm, we can explore more interesting findings in a better and more efficient manner.

In the future, *sCluster* could be explored for mining generic 3D objects rather than limited in sequential 3D structures. Furthermore, we can extend *sCluster* by producing centroid for each cluster, leverage on the centroid to speed up query process, and present and visualize the centroid for users' better understanding.

CHAPTER 5

Mining 3D Sequential Patterns With Constraints

5.1 Introduction

As an enhancement of sCluster, we study mining sequential 3D patterns with constraints of the minimum support and minimum confidence in this chapter. Datasets often include objects from various classes, and it is possible for a pattern to appear in different classes. Mining patterns with the constraints of minimum support and minimum confidence is important and meaningful as it forms the basis of applications such as classification and prediction [55].

For proteins, it is well-known that their 3D structures influence the biological functions [53]. In the absence of obvious amino acid sequence similarity, the detection of 3D structural similarity is a powerful tool to study remote homologies and protein evolution. For spatial moving objects, similar sequential 3D patterns could be the moving tracks of objects in the same class.

One challenge in pattern mining is caused by the object length and the number of objects. The object length could range from tens of vertices to thousands of vertices, and the number of objects in datasets could vary as well. A practical approach should be scalable with respect to both the object length and the number of objects.

Another challenge arises from the significant difference existing among various real-life datasets. Features of patterns found in different datasets which are of the same size may be very different. It is helpful for a mechanism to automatically tune the settings to discover the meaningful patterns to adapt the mining approach to various applications.

In this study, we propose MSP, a new approach for mining maximal sequential 3D patterns with the constraints of minimum support and mining confidence based on a seed-and-extension strategy. According to our knowledge, this problem is not well studied but valuable in real-life applications. MSP includes three stages. First, patterns with fixed length appearing in two 3D objects are produced as the seeds. Second, the vertical extension, a novel depth-first search algorithm is adopted to locate the hits of seeds in all 3D objects with the constraints of minimum support and minimum confidence. Third, the horizontal extension is to extend every pattern to be the longest without loss of hits. Furthermore, a dual-level binary-search algorithm, *Detect Proper Settings* (DPS), is implemented to automatically identify the proper settings to produce the number of patterns specified by users.

As the initial study, sCluster [83] has been discussed in Chapter 4 for mining subspace clusters in sequential 3D structures. It is an extended apriori algorithm for clustering similar substructures without considering the constraints of minimum support and minimum confidence. In the later section, we would evaluate the performance by comparing MSP with sCluster by setting both minimum support and minimum confidence to be 0. Comparison experiments show that MSP is faster and more scalable than sCluster.

In our experiments on protein datasets, the randomly selected patterns show that MSP is effective to detect the frequent patterns. We also adopt the result patterns to binary classification on different protein families in SCOP database [63] and the PhysioNet/CinC Challenge 2002 dataset. The results show MSP achieves very high accuracy and MSP **finds out unknown and unexpected patterns** in remote homologous proteins. We believe MSP can be effectively adopted for classification and prediction in various applications and knowledge domains.

5.2 Definitions

Symbol	Definition
$S[b:e]$	Object from the b-th vertex to e-th vertex on sequential 3D Object S
ε	Error tolerance
$Len(P)$	Length of pattern P
w	Seed length
C	Dataset of class C
$\neg C$	Dataset of the classes other than C
$\lvert D\rvert$	Number of items in dataset D
$support(P)$	Support of pattern P
$confidence(P)$	Confidence of pattern P
min_sup	Minimum support
min_conf	Minimum confidence

Table 5.1: Symbols and definitions

Table 5.1 lists the main symbols and definitions used throughout the chapter.

Sequential 3D Objects: A 3D sequential object is a group of vertices where each vertex only links to its left-side and right-side adjacent vertices. It can be represented as $S[b : e]$, where S is a sequential 3D object, and b and e are the beginning vertex and the ending vertex respectively. The edges are connecting adjacent vertices.

Similarity Measurement: We adopt fds which is defined in Chapter 4 as the measurement to evaluate the similarity between two sequential 3D objects.

5.2.1 Pattern And Hit

In real-life applications, a sequential 3D pattern could appear in many objects with small changes. For example, protein structures dynamically change within a small range due to different chemical environments, temperature and so on. The tracks of two synchronized spatial moving objects could be slightly different. Therefore, the pattern is a group of similar objects defined as:

$$\begin{cases} If\ P\ is\ a\ pattern\ in\ data\ set\ D\ with\ error\ tolerance\ \varepsilon \\ then\ for\ any\ two\ objects\ p_1, p_2 \in P, fds(p_1, p_2) \leq \varepsilon\ holds. \end{cases}$$

$Hit(D, P, \varepsilon)$ is the superset of P which is defined as all the occurrences of pattern p in dataset D with error tolerance ε. Assume, s is a portion or the whole of a sequential 3D object in D, then

$$(\forall p \in P : fds(p, s) \leq \varepsilon) \rightarrow s \in Hit(D, P, \varepsilon)$$

The more the cardinality of $Hit(D, P, \varepsilon)$ ($|Hit(D, P, \varepsilon)|$), the more frequent pattern P is. Meanwhile, the longer pattern P is, the more significant pattern P will be. Therefore, we define the maximal pattern to avoid trivial patterns:

$$\begin{cases} If\ P\ is\ \text{a}\ maximal\ pattern\ in\ D\ with\ error\ tolerance\ \varepsilon\ then \\ (1)\ there\ does\ not\ exist\ pattern\ P'\ that\ P' \supset P \\ (2)\ there\ does\ not\ exist\ pattern\ P'\ that\ Len(P') > Len(P) \\ and\ every\ object\ in\ P\ appears\ in\ one\ of\ the\ object\ in\ P' \end{cases}$$

Assume dataset $D = \{S_1, S_2, S_3, S_4, S_5\}$, and $P = \{S_1[6:15], S_3[10:19], S_4[5:14]\}$ is a maximal pattern in D with ε, then we cannot find hits in S_2 or S_5. Furthermore, if we extend the pattern to left side or right side, then there should be at least one pair of structures in P that have distance larger than ε. In this chapter, we focus on mining maximal patterns with the constraints of minimum support and minimum confidence. The problem can be defined as:

Problem 5.2.1 *Mining maximal patterns*
Given data set $D = C \cup \neg C$, error tolerance ε, minimum support min_sup, minimum confidence min_conf, then we have

$$\begin{cases} support(P) = \frac{|Hit(C \cup \neg C, P, \varepsilon)|}{|C \cup \neg C|} \\ confidence(P) = \frac{|Hit(C, P, \varepsilon)|}{|Hit(C \cup \neg C, P, \varepsilon)|} \end{cases}$$

the problem is to find:

$$\{P : P\ is\ a\ maximal\ pattern\ in\ C \cup \neg C\ with\ \varepsilon$$
$$\wedge (support(P) \geq min\_sup) \wedge (confidence(P) \geq min\_conf)\}$$

5.3 Algorithm

MSP employs a seed-and-extension framework as shown in Figure 5.1. MSP algorithm is described in Algorithm 5.3 which includes generating seeds, vertical extension and horizontal extension.

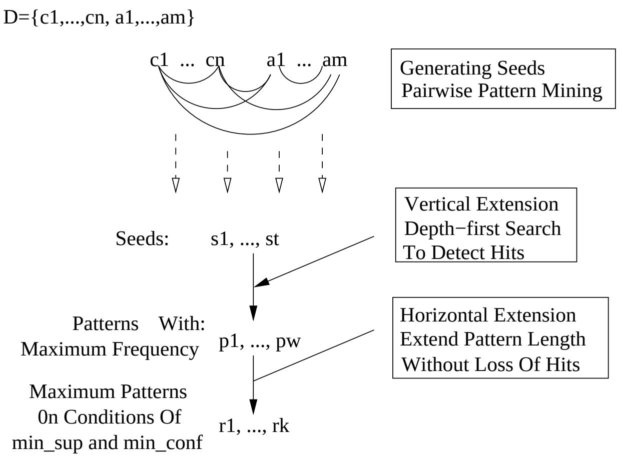


Figure 5.1: Framework of MSP

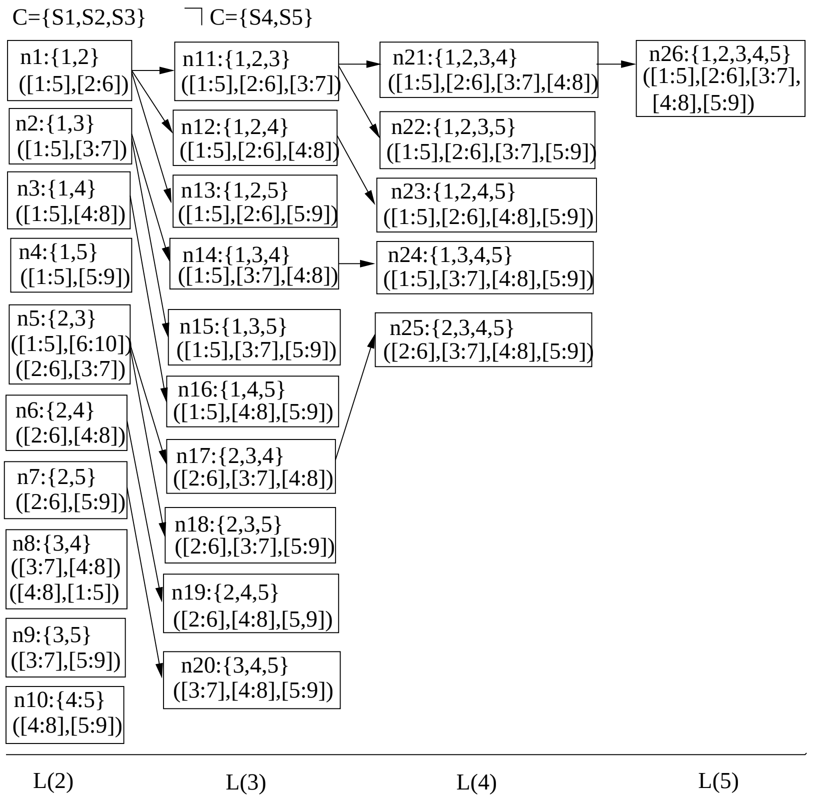

C={S1,S2,S3}
¬C={S4,S5}
n1:{1,2} ([1:5],[2:6])
n2:{1,3} ([1:5],[3:7])
n3:{1,4} ([1:5],[4:8])
n4:{1,5} ([1:5],[5:9])
n5:{2,3} ([1:5],[6:10]) ([2:6],[3:7])
n6:{2,4} ([2:6],[4:8])
n7:{2,5} ([2:6],[5:9])
n8:{3,4} ([3:7],[4:8]) ([4:8],[1:5])
n9:{3,5} ([3:7],[5:9])
n10:{4:5} ([4:8],[5:9])
n11:{1,2,3} ([1:5],[2:6],[3:7])
n12:{1,2,4} ([1:5],[2:6],[4:8])
n13:{1,2,5} ([1:5],[2:6],[5:9])
n14:{1,3,4} ([1:5],[3:7],[4:8])
n15:{1,3,5} ([1:5],[3:7],[5:9])
n16:{1,4,5} ([1:5],[4:8],[5:9])
n17:{2,3,4} ([2:6],[3:7],[4:8])
n18:{2,3,5} ([2:6],[3:7],[5:9])
n19:{2,4,5} ([2:6],[4:8],[5,9])
n20:{3,4,5} ([3:7],[4:8],[5:9])
n21:{1,2,3,4} ([1:5],[2:6],[3:7],[4:8])
n22:{1,2,3,5} ([1:5],[2:6],[3:7],[5:9])
n23:{1,2,4,5} ([1:5],[2:6],[4:8],[5:9])
n24:{1,3,4,5} ([1:5],[3:7],[4:8],[5:9])
n25:{2,3,4,5} ([2:6],[3:7],[4:8],[5:9])
n26:{1,2,3,4,5} ([1:5],[2:6],[3:7], [4:8],[5:9])
L(2)
L(3)
L(4)
L(5)

Figure 5.2: Example of vertical extension

Algorithm 5.3: MSP
input: 3D sequential structure classes, C and $\neg C$:
$C = \{S_1, S_2, ..., S_n\}$, $\neg C = \{S_{n+2}, ..., S_{n+m}\}$
error tolerance: ε, seed length: w,
minimum support: min_sup, minimum confidence: min_conf
output: patterns
data structure:

$$Node = \begin{cases} number\ of\ patterns & N \\ list\ of\ hits & H \end{cases}$$

Generating seeds

```
1.  $ST \leftarrow New\ Stack(\emptyset)$
2.  $L(2) \leftarrow \emptyset$, $LP \leftarrow \emptyset$
3.  $i \leftarrow n + m$
4.  while $(i > 1)$ $/*\, pairwise\ objects\ comparison *\,/$
5.      $j \leftarrow n + m$
6.      while $(j > i)$
7.          $H \leftarrow LSSSP(S_i, S_j, \varepsilon, w)$ /* longest pattern */
8.          $LP \leftarrow LP \cup \{H\}$
9.          $H \leftarrow$ Decompose $H$ to patterns with length of $w$
10.         $node \leftarrow New\ Node(|H|, H)$
11.         PUSH$(ST, node)$
12.         $L(2) \leftarrow L(2) \cup node.H$
13.         $j \leftarrow j - 1$
14.     $i \leftarrow i - 1$
```

Vertical extension

```
15. while ($ST \neq \emptyset$)
16.   $l \leftarrow$ POP($ST$) where $ut$ is the largest object ID in $l.H$
17.   if $ut = n + m$
18.     or $\frac{l.N+n+m-ut}{n+m} < min\_sup$
19.     or $\frac{|hits \in l.H \; from \; C|+1}{l.N+1} < min\_conf$
20.   then goto step 15
21.   $ST' \leftarrow New\ Stack(\emptyset)$
22.   for $ut < ut + i \leq n + m$
23.     $node \leftarrow New\ Node(0, \emptyset)$
24.     for $m \in l.H$: $m = \{S_{u1}[b_{u1} : e_{u1}], ..., S_{ut}[b_{ut} : e_{ut}]\}$
25.       if $\exists S_{ut+i}[b : e]$ where
26.             $\{S_{u1}[b_{u1} : e_{u1}], S_{ut+i}[b : e]\} \in L(2)$,
27.             ..., $\{S_{ut}[b_{ut} : e_{ut}], S_{ut+i}[b : e]\} \in L(2)$ hold
28.       then $node.H \leftarrow node.H \cup \{\{m \cup \{S_{ut+i}[b : e]\}\}\}$
29.             $node.N \leftarrow node.N + 1$
30.     if $node.N > 0$
31.     then PUSH($ST', node$)
32.           $L(t + 1) \leftarrow L(t + 1) \cup node.H$
33.   while ($ST' \neq \emptyset$)
34.     PUSH($ST$,POP($ST'$))
35.   go to step 15
```

Horizontal extension

```
36. For each pattern
37.   $P = \{S_1[b_1 : e_1], ..., S_t[b_t : e_t]\} \in L(2), ..., L(k)$
38.   $(l, r) \leftarrow (b_1, Len(S_1) - e_1)$
39.   for $i = 1$ to $t - 1$
40.     for $j = i + 1$ to $t$
41.       $(l', e') \leftarrow$ ExtPairPatt($LP, S_i[b_i : e_i], S_j[b_j : e_j]$)
42.       if ($l' < l$) then $l \leftarrow l'$
43.       if ($r' < r$) then $r \leftarrow r'$
44.       if $(l, r) = (0, 0)$ then go to step 36
45.   $L(t) \leftarrow L(t) - \{P\}$
46.             $\cup\{\{S_1[b_1 - l : e_1 + r], ..., S_n[b_n - l : e_t + r]\}\}$
47. return $\bigcup_{i=1}^{k} L(i)$ where $L(i) \neq \emptyset$
```

Figure 5.3: MSP Algorithm

5.3.1 Generating Seeds: Pairwise Pattern Mining

A seed is fixed-length pattern appearing in two objects. We compare every two objects to generate seeds using the *Longest Synchronized Similar SubObject Pair* (*LSSSP*) mining algorithm in Chapter 4 [83] as shown in step 7. The results of *LSSSP* are longest pairwise patterns stored in *LP*. In step 9, every pairwise pattern longer than minimum length would be divided into multiple seeds using a sliding window. For example, with seed length of 6, we will decompose the pattern

$$\{S_1[3:10], S_2[2:9]\}$$

into three fixed-length seeds,

$$\begin{cases} \{S_1[3:8], S_2[2:7]\} \\ \{S_1[4:9], S_2[3:8]\} \\ \{S_1[5:10], S_2[4:9]\} \end{cases}$$

The overall computational complexity of generating seeds is:

$$O((n+m)^2(l^2 + \textstyle\sum_{i=1}^{l} \lg i!)),\ where$$
$$l\ is\ the\ object\ length, (n+m)\ is\ the\ number\ of\ objects$$

5.3.2 Vertical Extension: Depth-first Search to Detect Hits

Vertical extension is to detect all the hits of seeds in the whole dataset. Cong et. al. [24] have proposed, Farmer, a depth-first search algorithm for microarray pattern mining. The method was verified to be efficient in handling long attribute lists because of the use of various pruning strategies. In datasets of spatial moving objects and protein chains, the object length can be as long as hundreds of vertices.

This promotes us to create a depth-first search approach on enumerate all the eligible hits.

Step 15 to 35 in Algorithm 5.3 describe the vertical extension. A stack ST is used to realize the depth-first searching order. The patterns including i hits are stored in both the result list $L(i)$ and the stack nodes.

In the sample shown in Figure 5.2, every box is a stack node which records all patterns in the objects listed in $\{\ \}$. The items in $\{\ \}$ are the object IDs which are hit by the patterns and $([\ ], ...[\ ])$ are the starting positions and the ending positions of the hits in objects. For example, node

$$n_5 : \{2, 3\}([1 : 5], [6 : 10]), ([2 : 6], [3 : 7])$$

includes two patterns,

$$\{S_2[1 : 5], S_3[6 : 10]\}\ and\ \{S_2[2 : 6], S_3[3 : 7]\}$$

Patterns with hits in the same objects are included in the same stack node. Node n_5 includes all the patterns appearing in S_2 and S_3. The enumeration is in ascending order. To vertically extend n_5, the algorithm will detect the hits in S_4 and S_5. To detect hits of $\{S_2[1 : 5], S_3[6 : 10]\}$ in S_4 , we check if there are any pairwise patterns in form of $\{S_2[1 : 5], S_4[b : e]\}$ and $\{S_3[6 : 10], S_4[b : e]\}$ in $L(2)$. In this sample, this extension fails. After that, MSP tries to extend $\{S_2[2 : 6], S_3[3 : 7]\}$ to S_4. We see $\{S_2[2 : 6], S_4[4 : 8]\}$ and $\{S_3[3 : 7], S_4[4 : 8]\}$ in $L(2)$. Consequently, we produce a new node $n_{17} : \{2, 3, 4\}([2 : 6], [3 : 7], [4 : 8])$. The algorithm further extends node n_{17} by detecting hits of the pattern, $\{S_2[2 : 6], S_3[3 : 7], S_4[4 : 8]\}$, in S_5. Similarly, we check $L(2)$ and find $\{S_2[2 : 6], S_5[5 : 9]\}$, $\{S_3[3 : 7], S_5[5 : 9]\}$ and $\{S_4[4 : 8], S_5[5 : 9]\}$ in $L(2)$. Then we extend n_{17} to n_{25}. Here we say n_{17} and n_{18}

are vertically extended by n_5. Since S_5 is the last object in $C \cup \neg C$, MSP does not continue vertical extension on node n_{25} but retrieves n_{18} from the stack. The order of generating stack nodes from n_5 is:

$$...n_5 \rightarrow n_{17} \rightarrow n_{25} \rightarrow n_{18}...$$

Because MSP follows depth-first search order, if a node is pruned, then all the nodes that could be vertically extended are pruned. It significantly reduces the search space. Furthermore, during vertical extension, *min_sup* and *min_conf* are used to prune ineligible nodes according to the below properties.

Property 5.3.1 *Assume $P_1 \in node$ and $P_2 \in node$, then*

$$\begin{cases} support(P_1) = support(P_2) \\ confidence(P_1) = confidence(P_2) \end{cases}$$

Proof: *Since patterns in one node are from the same objects, so the supports and confidences are the same.* □

Therefore, we define the support and the confidence of a node as the support and the confidence of any pattern which is included by the node respectively.

Property 5.3.2 *If node n_1 is vertically extended by n_2, then*

$$\begin{cases} support(n_1) > support(n_2) \\ confidence(n_1) \leq confidence(n_2) \end{cases}$$

Proof: *On one hand, because n_1 is vertically extended by n_2, there must be at least one more object included by n_1 compared to n_2. Thus, we have $support(n_1) > support(n_2)$. On the other hand, since vertical extension follows depth-first search* *order, the algorithm will extend to objects in C before objects in $\neg C$. If the algorithm extends to an object in C, then the confidence is always 100%. Otherwise, the hits in C will not increase while those in $\neg C$ are possible to increase. Hence, we have $confidence(n_1) \leq confidence(n_2)$.* $\square$

Leveraging on the properties, we implement two pruning strategies that speed up the search process. If the confidence or support of a node is ineligible, all patterns in the node will be pruned.

Pruning with ***min_sup*****:** Step 18 is to prune ineligible nodes using *min_sup*. It calculates the maximum support of the nodes that could be generated by the current node in vertical extension. If it is less than *min_sup*, the node will not be extended. Given a dataset $D = C \cup \neg C$ and a stack node n_i as:

$$\{u1, ..., ut\}$$
$$([b_{u1,1} : e_{u1,1}], ..., [b_{ut,1} : e_{ut,1}]), ..., ([b_{u1,1} : e_{u1,1}], ..., [b_{ut,k} : e_{ut,k}])$$

The maximum possible support of the nodes which are generated by vertical extension in n_i is

$$\frac{t + |C| + |\neg C| - ut}{|C| + |\neg C|}$$

For example, the maximum possible support of n_3 and the maximal possible support of n_5 are $\frac{2+1}{2+3} = 0.6$ and $\frac{2+2}{2+3} = 0.8$ respectively. If $min\_sup = 0.7$, then MSP will stop vertical extension on n_3 while continue on n_5.

Pruning with ***min_conf*****:** Step 19 is to prune ineligible nodes using *min_conf*. It calculates the confidence of current node. According to Property 5.3.2, the confidence of current node is the maximum confidence of the patterns that can be vertically extended by the current node. If it is less than *min_conf*, the node will not be extended. For example, the confidence of n_3 and the confidence of n_5 are

$\frac{1}{2} = 0.5$ and $\frac{2}{2} = 1$. If $min\_conf = 0.8$, then MSP will stop vertical extension on n_3 while continue on n_5.

5.3.3 Horizontal Extension: Extend Pattern Length without Loss of Hits

After detecting all hits, MSP obtains the patterns in fixed length. It is possible to extend the patterns in terms of length as well as to keep the distance between any two objects in one pattern be less than ε.

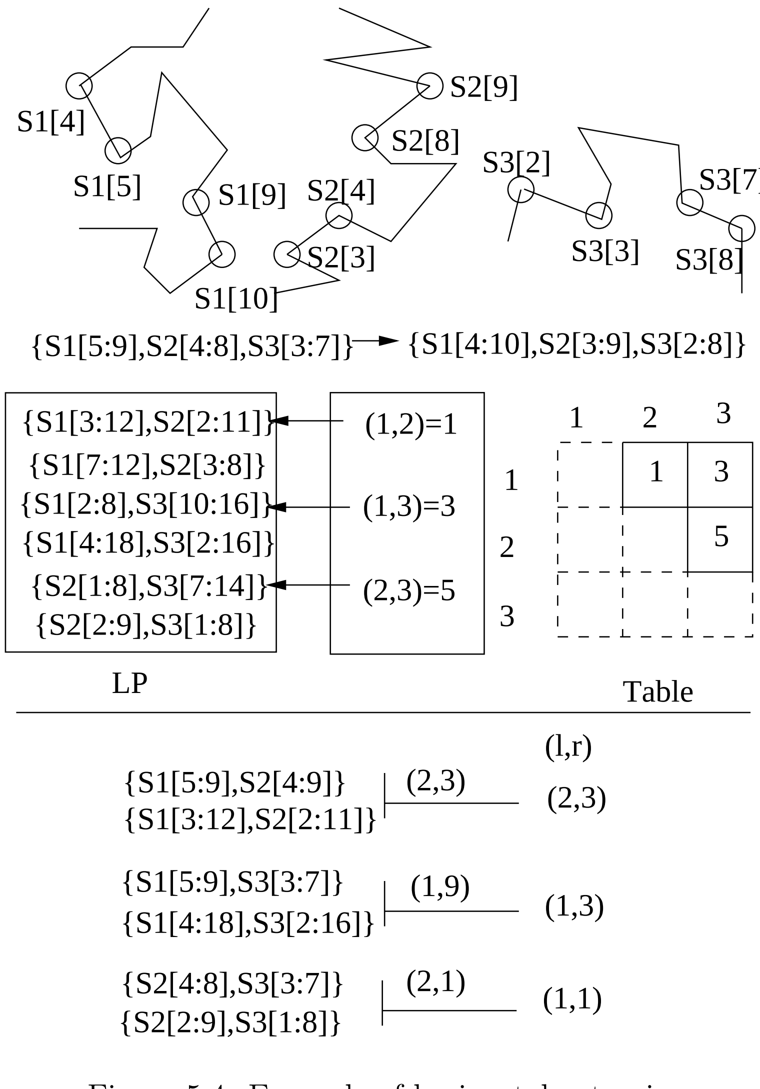


Figure 5.4: Example of horizontal extension

Problem 5.3.1 *Horizontal extension*

Given a similar structure set $\{S_1[b_1 : e_1], ..., S_n[b_n : e_n]\}$ *where*

$$\forall 1 \leq i < j \leq n : fds(S_i[b_i : e_i], S_j[b_j : e_j]) \leq \varepsilon$$

we are going to find the maximum value of the summation of left side extension l *and right side extension* r *where*

$$\forall 1 \leq i < j \leq n : fds(S_i[b_i - l : e_i + r], S_j[b_j - l : e_i + r]) \leq \varepsilon$$

This problem is solved by step 36 to 46 in Algorithm 5.3. For each pattern, we horizontally extend every two structures according to the longest pairwise patterns in LP with a subroutine

$$ExtPairPatt(LP, S_i[b_i : e_i], S_j[b_j : e_j])$$

$ExtPairPatt$ is to find the longest pairwise pattern which contains $\{S_i[b_i : e_i], S_j[b_j : e_j]\}$. Since the longest pairwise patterns in LP will be frequently checked, we build a table to store the entry for each two objects in LP. In sample shown in Figure 5.4, there are 3 objects and a pattern $\{S_1[5 : 9], S_2[4 : 8], S_3[3 : 7]\}$ is found by vertical extension. Here, minimum pattern length is 5. The longest pairwise patterns on S_1, S_3 are stored in LP from the third entry $\{S_1[2, 8], S_3[10 : 16]\}$. So the value of cell $(1, 3)$ in the table is 3.

We first check if there is patterns on S_1 and S_2 and the pattern could cover $\{S_1[5 : 9], S_2[4 : 8]\}$ with a synchronized left-side and right-side extension. $\{S_1[3 : 12], S_2[2 : 11]\}$ is a longest pairwise pattern covering $\{S_1[5 : 9], S_2[4 : 8]\}$ with offset $(2, 3)$. Similarly, we find the offset of $\{S_1[4 : 18], S_3[2 : 16]\}$ on $\{S_1[5 : 9], S_3[3 : 7]\}$

is $(1, 9)$ that intersects with $(2, 3)$ to be $(1, 3)$. After that, we find the longest pairwise pattern $\{S_2[2 : 9], S_3[1 : 8]\}$ contains $\{S_2[4 : 8], S_3[3 : 7]\}$ with the offset of $(2, 1)$. The maximum offset becomes $(1, 1)$ by intersecting $(2, 1)$ with $(1, 3)$.

On one hand, the extension offset on two objects of a pattern iteratively intersects with the current offset until the offset related to every two objects in the pattern is intersected. It guarantees that the pattern cannot be extended longer otherwise the fds value between at least two objects in the pattern is larger than ε. On the other hand, all maximal patterns are found without loss because the stage of generating seeds produces at least one seed for any one of the maximal patterns. Although it is possible for a pattern to be a subset of another pattern, the pattern can be easily filtered with a simple post-process.

5.3.4 Detection of Proper Settings

MSP involves various parameters which influence the number of generated patterns and processing time. In real-life applications, it is inconvenient for users to decide settings especially when they are not familiar with the algorithm. A loose pattern detection setting would produce too many patterns that may take unacceptable time and exhaust resources, while a tight setting would generate too few patterns for future study. Furthermore, datasets especially from different areas can be very different. It is difficult to pre-define a setting that suits all datasets. This motivates us to devise a strategy to detect proper settings automatically. We define the problem as follows:

Problem 5.3.2 *Detect proper settings*

Given a data set $C \cup \neg C$, min_sup and min_conf, we are going to find a proper setting on (w, ε) to generate approximate n patterns each of which is with t hits, where t is the minimum integer no less than $min\_sup \cdot |C \cup \neg C|$.

In order to avoid too many patterns, an approximate number of patterns should be defined. Since the number of patterns grows with the increasing of error tolerance ε and the decreasing of seed length w, we create a heuristic approach to automatically detect proper settings based on a dual-level binary-search schema as shown in DPS Algorithm 5.3.4. The first level and the second level are with respect to w and ε respectively. Initially, we set the range of w and the range of ε as:

$$[w_{min}, w_{max}] : w_{min} = 1,\ w_{max} = \textit{object length}$$
$$[\varepsilon_{min}, \varepsilon_{max}] : \varepsilon_{min} = 0,\ \varepsilon_{max} = 3 * \textit{object length}$$

Optimistically, we assume the objects are exactly same with each other. It implies that $w_{max} = \textit{object length}$ and $\varepsilon_{min} = 0$. Pessimistically, we assume the objects are very different. It implies either the pattern length is as small as 1 or the error tolerance is as large as $3 * \textit{object length}$ because the maximum difference based on fds on one vertex is the number of the features, i.e., 3. The range can be defined more precisely with domain knowledge in real-life applications. As a start, (w, ε) is set as $(w_{max}, \varepsilon_{min})$. The μ_w and μ_ε are the incremental steps on w and ε respectively. We fix w while tune ε to check if there is ε to produce patterns as expected. If number of patterns differs from n significantly, we increase or decrease ε in a manner of binary search within $[\varepsilon_{min}, \varepsilon_{max}]$. If there is no proper ε with the fixed w, we tune w in a manner of binary search within $[w_{min}, w_{max}]$. During vertical extension of MSP, if we find that the existing patterns are much more than n, then we stop it to speed up the detection. By this means, a proper setting to generate approximate n patterns will be detected. The times for detection of proper settings to invoke MSP is:

$$O(lg\frac{w_{max} - w_{min}}{\mu_w} \cdot lg\frac{\varepsilon_{max} - \varepsilon_{min}}{\mu_\varepsilon})$$

```
Algorithm 5.3.4: Detect Proper Settings (DPS)
**Input:** 3D sequential structure classes, $C$ and $\neg C$:
$C = \{S_1, S_2, ..., S_n\}$,        $\neg C = \{S_{n+2}, ..., S_{n+m}\}$
minimum support: $min\_sup$,    minimum confidence: $min\_conf$
number of hits: $t$,      range of number of patterns: $[n_{min}, n_{max}]$
**Output:** $(w, \varepsilon)$ where:
  $|MSP(C, \neg C, w, \varepsilon, min\_sup, min\_conf)| \in [n_{min} : n_{max}]$
**Method:**
1.  Initialize $[\varepsilon_{min} : \varepsilon_{max}]$ and $\mu_\varepsilon$
2.  Initialize $[w_{min} : w_{max}]$ and $\mu_w$
3.  $[\varepsilon_l : \varepsilon_r] \leftarrow [\varepsilon_{min} : \varepsilon_{max}]$; $[w_l : w_r] \leftarrow [w_{min} : w_{max}]$
4.  $\varepsilon \leftarrow \varepsilon_l$; $w \leftarrow w_r$
5.  $Result \leftarrow MSP(C, \neg C, w, \varepsilon, min\_sup, min\_conf)$;
/* A counter for pattern number is inserted to MSP. Once the counter
becomes larger than a certain number, MSP will stop. */
6.  if $|Result| \in [n_{min} : n_{max}]$ then return $(w, \varepsilon)$
7.  if $|Result| > n_{max}$     /* too many patterns */
8.  then if $\varepsilon_u > \varepsilon_l$ then { $\varepsilon_u \leftarrow \varepsilon - \mu_\varepsilon$; $\varepsilon \leftarrow \frac{(\varepsilon+\varepsilon_l+\mu_\varepsilon)}{2}$ }
9.      else if $w_r < w_l$ then return failure;
10.          else {$w_l \leftarrow w + \mu_w$; $w \leftarrow \frac{(w_u+w+\mu_w)}{2}$;
11.              $[\varepsilon_l : \varepsilon_r] \leftarrow [\varepsilon_{min} : \varepsilon_{max}]$; $\varepsilon \leftarrow \varepsilon_l$}
12. else if $\varepsilon_r > \varepsilon_l$          /* too few patterns */
13.     then { $\varepsilon_l \leftarrow \varepsilon + \mu_\varepsilon$; $\varepsilon \leftarrow \frac{(\varepsilon+\varepsilon_l-\mu_\varepsilon)}{2}$ }
14.     else if $w_r < w_l$ then return failure;
15.          else {$w_r \leftarrow w - \mu_w$; $w \leftarrow \frac{(w_l+w-\mu_w)}{2}$ }
16.              $[\varepsilon_l : \varepsilon_r] \leftarrow [\varepsilon_{min} : \varepsilon_{max}]$; $\varepsilon \leftarrow \varepsilon_l$}
17. goto step 5.                                        □
```

Figure 5.5: DPS Algorithm

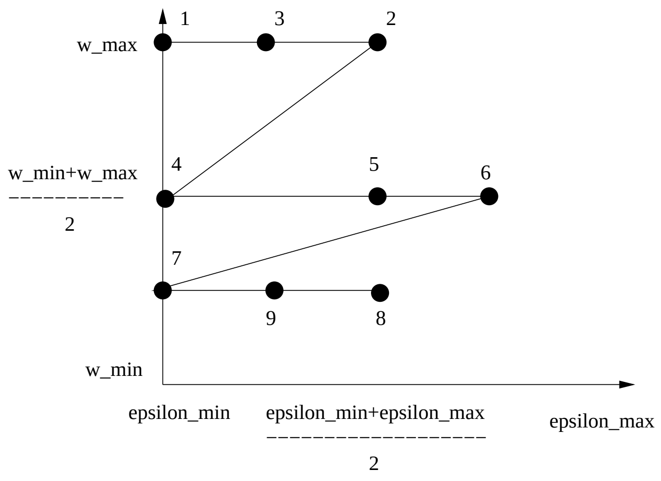


Figure 5.6: Example of DPS Algorithm

Figure 5.6 illustrates an example of DPS algorithm. It starts from point 1 $(\varepsilon_{min}, w_{max})$. Too few patterns are generated, so the algorithm tests $(\frac{\varepsilon_{max}+\varepsilon_{min}}{2}, w_{max})$. However, it generates too many patterns. Then DPS algorithm tests $(\varepsilon_{min} + \frac{\varepsilon_{max}-\varepsilon_{min}}{4}, w_{max})$. While DPS algorithm still generates too many patterns and the step μ_ε is larger than $\frac{\varepsilon_{max}-\varepsilon_{min}}{4}$, DPS Algorithm adjusts w to $\frac{w_{max}+w_{min}}{2}$. Finally, the setting of $(w_{min} + \frac{w_{max}-w_{min}}{4}, \frac{\varepsilon_{max}+\varepsilon_{min}}{2})$ generates patterns as expected.

5.4 Experiments

We design a group of experiments to evaluate MSP on a PC with a Pentium 4 2.6Ghz CPU, 1GB of SDRAM and a 7200rpm 40GB hard disk running Windows XP. Two groups of datasets are used. One is a synthetic dataset with 3D coordinates which are floatpoint numbers ranging from 0 to 1. The other includes protein chains from various families from SCOP database. We study the effect of parameters to the performance followed by the comparison experiments between MSP and sCluster. Finally, we will apply the results to protein family classification and PhysioNet/CinC Challenge 2002 dataset to test the effectiveness.

5.4.1 Parameters

The parameters of MSP includes length of seeds, error tolerance, number of objects, object length, *min_sup* and *min_conf*. In this section, we study the impacts on the performance of parameters on protein chains.

Effect of number of objects: We study the scalability in terms of number of objects on protein chains where seed length w is 15. Both *min_sup* and *min_conf* are set to 0. First, we tune error tolerance ε ranging from 1.2 to 1.7, but fix object length to be 500 vertices. The results are presented in Figure 5.7. Second, we tune the protein chain length ranging from 100 vertices to 500 vertices but fix ε to be 1.7. The results are presented in Figure 5.8.

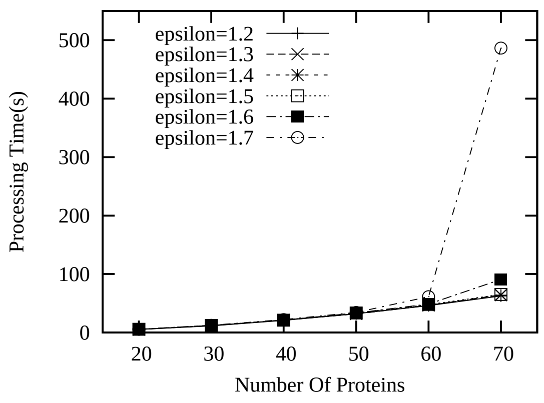


Figure 5.7: Number of objects and ε VS. Processing time

Figure 5.7 and 5.8 show that the processing time increases with the number of objects modestly except in case of $\varepsilon = 1.7$, number of objects=70 and object length=500 where too many patterns are generated during vertical extension. This case can be avoided by DPS Algorithm 5.3.4. In the other cases, because the most time-consuming part is generating seeds, MSP is scalable with the increasing of number of objects.

Effect of object length: We test MSP on protein chains with seed length=15, *min_sup* $= 0$ and *min_conf* $= 0$ by tuning object length from 100 to 500. In case

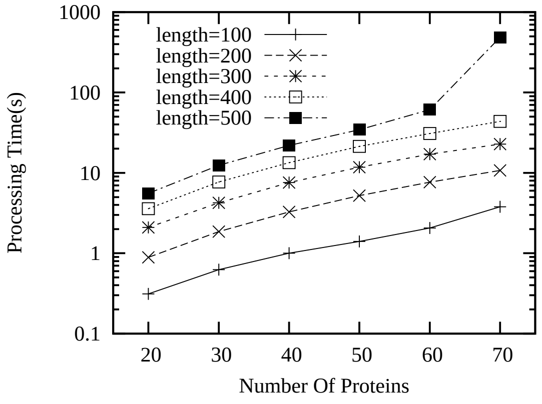


Figure 5.8: Number of objects and object length VS. Processing time

of chains longer than target length, we truncate it. Figure 5.9 depicts the results of the experiments in 60 protein chains with ε ranging from 1.2 to 1.7. MSP shows a good scalability in terms of object length.

Figure 5.10 shows the results of the experiments with $w = 15$, $\varepsilon = 1.7$, $min\_sup = 0$ and $min\_conf = 0$ in 6 datasets including 20, 30, 40, 50, 60 and 70 protein chains respectively. Except the case of 70 protein chains with object length of 500, we observe that the processing time increases modestly with object length. In case of 70 protein chain with object length of 500, the setting of (w, ε) leads to too many patterns.

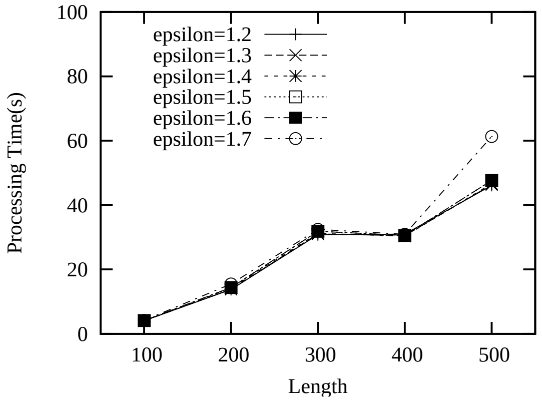


Figure 5.9: Object length and ε VS. Processing time

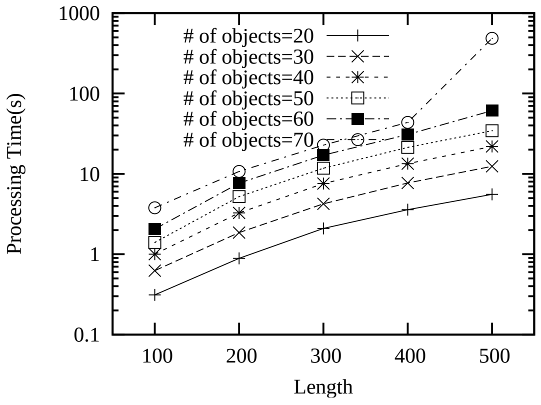


Figure 5.10: Object length and number of objects VS. Processing time

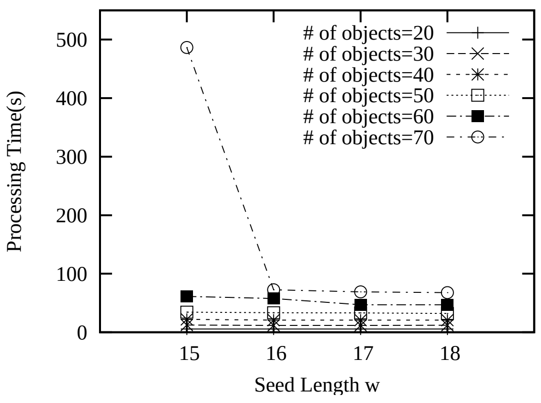


Figure 5.11: Seed length and number of objects VS. Processing time

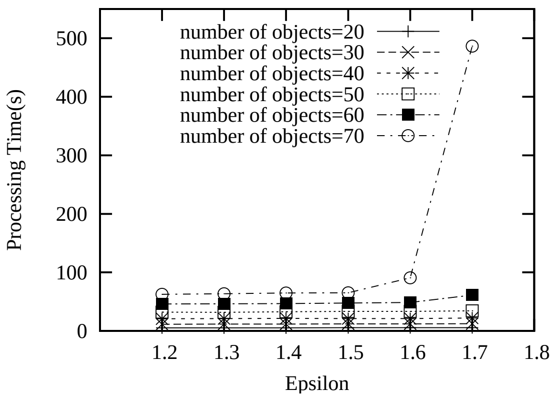


Figure 5.12: ε and number of objects VS. Processing time

Effect of seed length: Seed length is studied in 6 datasets which include 20, 30, 40, 50, 60 and 70 protein chains respectively while we fix $\varepsilon = 1.7$ and object length=500. Figure 5.11 displays that the processing time does not vary significantly except the setting of $w = 15$ in 70 protein chain dataset because most of time is taken in seeds generation which mainly depends on the dataset size. The reason that seed length of 15 in 70 protein chains leads to significant slump-down of processing speed is that the short seed length forms the loose condition to detect patterns. It can also be avoided by DPS Algorithm 5.3.4.

Effect of ε: Figure 5.12 shows the effect of ε in various datasets with object length of 500, $w = 15$, $min\_sup = 0$ and $min\_conf = 0$. We notice the processing time becomes sensitive for $\varepsilon = 1.7$ and number of object=70 due to too many patterns. This case would be prevented by DPS Algorithm. With the other settings, MSP slumps down with the relaxing of ε modestly.

Effect of min_sup and min_conf: We tune min_sup from 0 to 0.5 while fix $min\_conf = 0$, $w = 15$ and object length=500 in datasets of 20, 30, 40, 50, 60 and 70 objects respectively. Figure 5.13 shows the speed for $min\_sup = 0.2$ is 3 times faster that for $min\_sup = 0.3$ in 70-object dataset since the pruning with min_sup significantly speeds up the vertical extension. Figure 5.14 shows the results by fixing $min\_sup = 0$ while tuning min_conf between 0.1 to 0.5. In 70-object dataset, the pruning with min_conf also speeds up the processing. We observe both pruning strategies enhance the mining performance in large datasets, and the pruning with min_sup is more effective than the pruning with min_conf.

Number of patterns: Numbers of patterns generated in 4 datasets are plotted in Figure 5.15. Table 5.2 describes the number of objects and the object length in each dataset. Here we set $w = 15$, $\varepsilon = 1.7$, $min\_sup = 0$ and $min\_conf = 0$. We observe the number of patterns increases almost by 1 magnitude when the number of objects grows from 60 to 70 because the number of combination of the objects grows significantly and the number of generated patterns is related to the features of

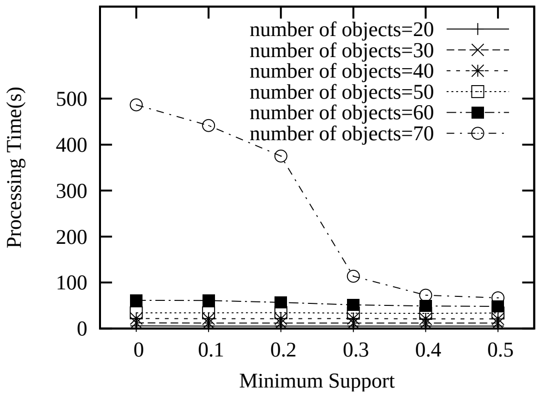


Figure 5.13: *min_sup* and number of objects VS. Processing time

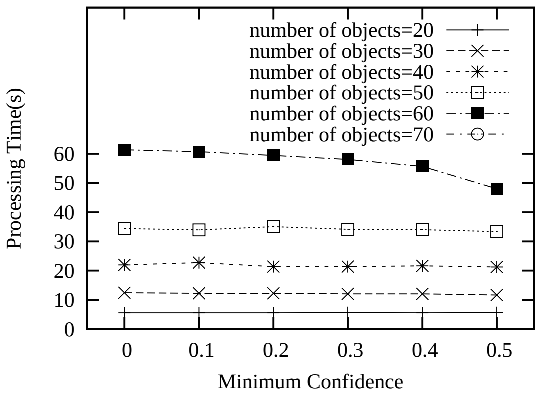


Figure 5.14: *min_conf* and number of objects VS. Processing time

datasets. Although MSP is scalable in terms of both number of objects and object length, it is important and necessary to detect proper settings automatically since the patterns generated in different datasets are significantly different.

Case	1	2	3	4
number of objects	70	60	30	60
object length	500	500	500	100

Table 5.2: Settings of cases

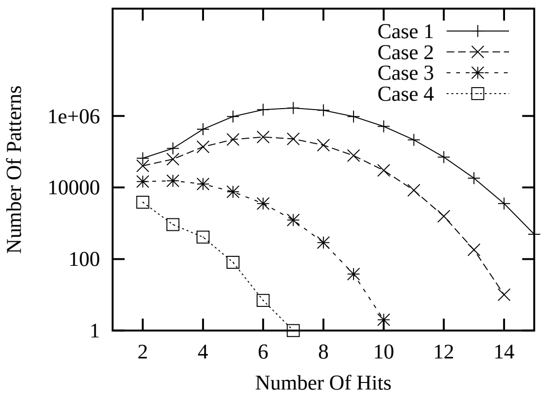


Figure 5.15: Number of patterns VS. Number of hits

5.4.2 Comparing MSP with sCluster

To study the scalability of MSP in terms of number of objects, length of objects and error tolerance, we design three groups of comparison experiments with sCluster on real protein chains and synthetic data. Since sCluster does not support the constraints of minimum support and minimum confidence, we set $min\_sup = 0$ and $min\_conf = 0$ in all comparisons. In this case, the patterns found by MSP are the same with those found by sCluster because both MSP and sCluster are designed to find out all maximal patterns without loss.

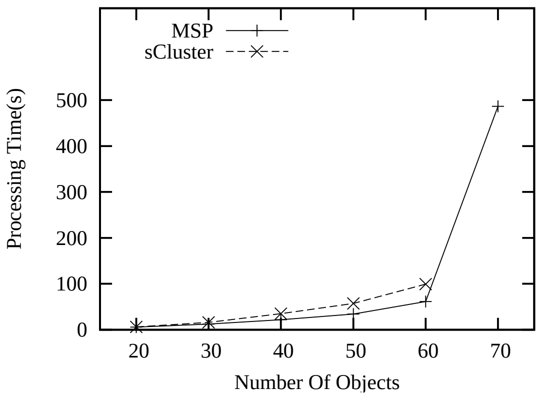


Figure 5.16: MSP VS. sCluster on number of objects

First, we compare the performance in protein chains. To study the scalability in terms of number of objects, we set $\varepsilon = 1.7$, $w = 15$, and object length=500. Figure 5.16 shows MSP is slightly faster than sCluster when the objects are not more than 50 because the processing time of both approaches is mainly taken in generating seeds. For 60 objects, we observe MSP is almost 2 times faster than sCluster. Furthermore, MSP succeeds in producing results for 70 objects while sCluster fails due to too many patterns that have to be retained in memory to produce larger patterns.

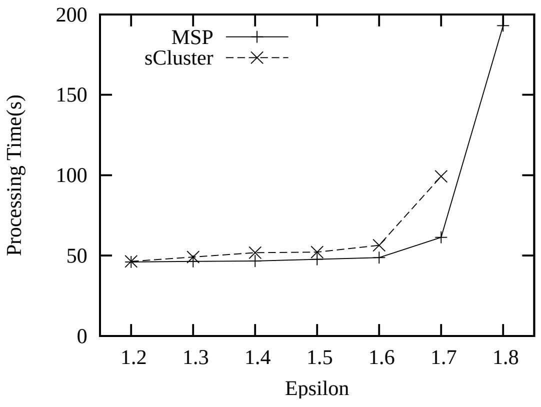


Figure 5.17: MSP VS. sCluster on ε

Figure 5.17 describes the scalability comparison results in term of ε in 60-object dataset. The two approaches perform similar when ε ranges from 1.2 to 1.6. When $\varepsilon \geq 1.6$, MSP starts to show better performance than sCluster due to different extension strategies. For $\varepsilon = 1.8$, MSP generates all results while sCluster does not complete since too many medium results have to be resident in memory. Figure 5.18 depicts the processing time in terms of the object length with $\varepsilon = 1.7$ and it shows that MSP is about 2 times faster than sCluster.

Second, we generate synthetic datasets with coordinate values randomly ranging from 0 to 1 and study the scalability with respect to number of objects and object length. The comparison results shown in Figure 5.19 and 5.20 are similar with the results in protein chain datasets.

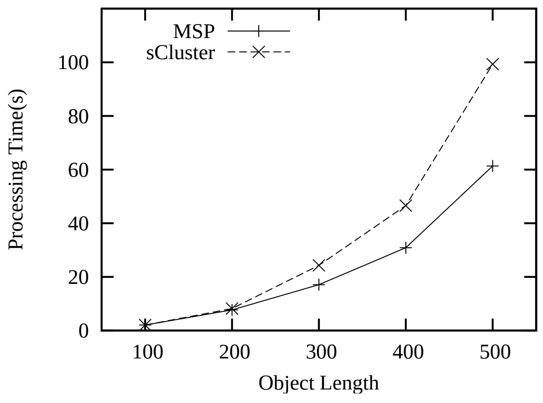


Figure 5.18: MSP VS. sCluster on object length

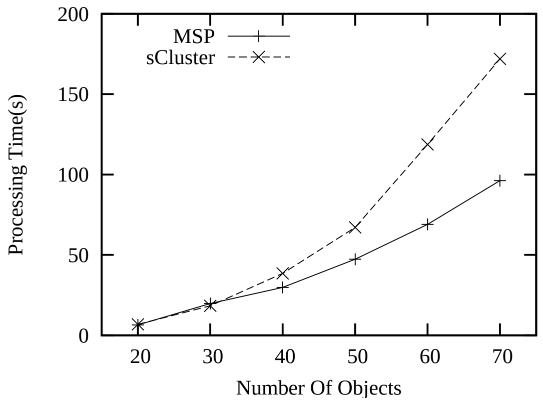


Figure 5.19: MSP VS. sCluster on number of objects in synthetic data

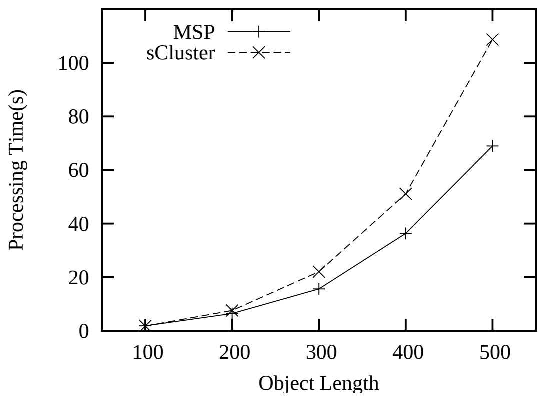


Figure 5.20: MSP VS. sCluster on object length in synthetic data

In all, MSP is faster and more scalable compared to sCluster in terms of number of object, object length and error tolerance. For relatively large dataset, MSP can be 2 times faster than sCluster, and MSP succeeds in producing results in some cases while sCluster fails. Besides, MSP supports the constraints of minimum support and minimum confidence while sCluster does not.

5.5 The Applications of MSP

MSP can be used in many areas. In this section, we adopt MSP in binary classification for SCOP protein families and the PhysioNet/CinC Challenge 2002 dataset.

5.5.1 MSP for Binary Classification in Protein Structures

SCOP is a protein database classified in five hierarchal levels: class, fold, superfamily, family and individuals. We evaluate the binary classification accuracy on three datasets. Each dataset includes protein chains from two families. The first dataset and the second dataset are used in [38]. The first dataset (D1) includes 18 protein chains from two families, nuclear receptor ligand-binding domain proteins (NB) and the prokaryotic serine protease family (PSP), which are from α-class and β-class respectively. The second dataset (D2) includes 44 protein chains from eukaryotic serine proteases (ESP) and prokaryotic serine proteases that belong to the same superfamily. The third dataset (D3) includes 50 protein chains from rubrerythrin n-terminal domains (RN) and 50 protein chains from calmodulin-like troponin c domains (CTC).

First, protein chains are represented as sequential 3D objects by connecting the adjacent C_α atoms on amino acids. Second, the dataset is partitioned to training set and test set and we conduct pattern mining on training data using MSP. Third, the obtained maximal patterns are used to detect hits in test data. The pattern collection with higher hit rate in test protein chain identifies the candidate family.

Table 5.3 presents the target number of patterns, min_sup and min_conf that are the inputs to automatically detect proper settings of (w, ε). DT and AD(0.1) are the methods presented in [38].

Experiments show that in all test cases the patterns correctly predict protein families. It implies that the patterns found by MSP are meaningful and useful in protein classification.

From the obtained maximal patterns in rubrerythrin n-terminal domains, we randomly select a pattern including 4 objects, $\{d1b71a1[7:61]$, $d1bcfa\_[6:60]$, $d1euma\_[2:56]$ and $d1jgca[5:59]\}$, and plot them in Figure 5.21 using $PDB2multiGIF$ [47]. The pattern is a $helix - turn - helix$ which is consisted of two $\alpha - helix$s connected by a turn and its objects are similar with each other.

		$(min\_sup$, $min\_conf)$	(w, ε)
D1	NB	(30%, 100%)	(20,2.73)
	PSP	(30%, 100%)	(20,0.99)
D2	PSP	(30%, 100%)	(20,0.99)
	ESP	(30%, 100%)	(20,0.80)
D3	RN	(15%, 85%)	(34,1.03)
	CTC	(15%, 85%)	(25,3.81)

Table 5.3: Datasets and settings

	MSP	DT	AD(0.1)
D1	100%	100%	100%
D2	100%	95%	95%
D3	100%	-	-

Table 5.4: Accuracy comparison among MSP, DT and AD(0.1) in binary classification.

We also conduct mining on the protein chain dataset randomly selected from the multi-domain class (for those with domains of different fold and for which **no homologies are known at present**) [63], and present a pattern found from

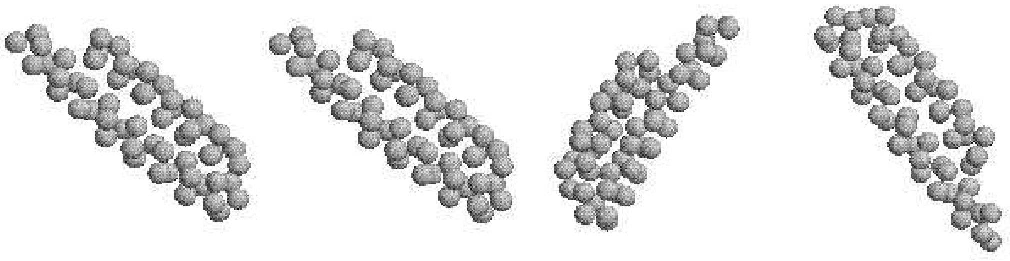


Figure 5.21: Sample pattern - 1: $\{d1b71a1[7:61], d1bcfa\_[6:60], d1euma\_[2:56], d1jgca[5:59]\}$

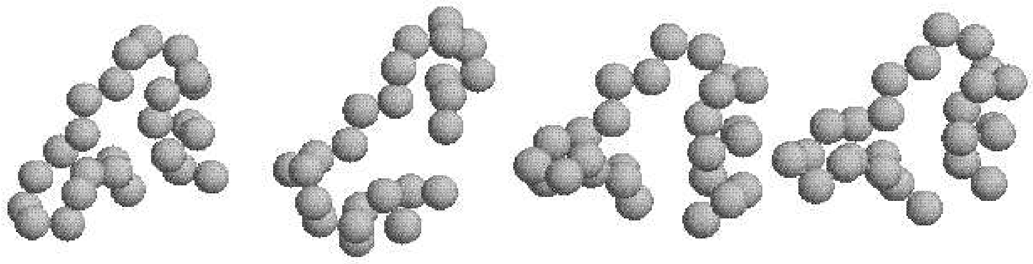


Figure 5.22: Sample pattern - 2: $\{d1bmr\_[2:26], d1cn2\_[3:27], d1i6ga\_[2:26], d1nrb[1:25]\}$

this dataset in Figure 5.22. This pattern includes four objects, $\{d1bmr\_[2:26]$, $d1cn2\_[3:27]$, $d1i6ga\_[2:26]$ and $d1nrb[1:25]\}$. We observe the pattern is a group of amino acid atoms in a strand sitting at the middle of a small helix and a turn. The similarity among the four objects is obvious. Therefore, we believe MSP can be an effective approach for mining **unknown and unexpected patterns** especially for the protein chains whose homologies are unknown.

5.5.2 MSP for PhysioNet/CinC Challenge 2002 Dataset

The PhysioNet/CinC Challenge 2002 dataset
http://www.physionet.org/challenge/2002/dataset.tar.gz
consists of 50 time series of inter-beat intervals. Each series contains between 20 and 24 hours of data (between 70,000 and 130,000 intervals). Approximately half

of the series contain data derived from ambulatory ECG recordings of subjects between the ages of 20 and 50 who have no known cardiac abnormality. The other half contain synthetic data, which have been generated using the models submitted by other researchers. Each model has been used to generate two time series. We remove the 4 unknown records and conduct the classification over the real and synthetic records.

Although the records are one-dimension data, we derive two angles on every average inter-beat interval per 60 seconds as the features to calculate the summation of the feature difference. Assume I_{i-1}, I_i and I_{i-1} are three consecutive intervals, then one angle on I_i is the arc cosine of $\frac{I_{i-1}-I_i}{max(I_{i-1},I_i)}$ and the other angle on I_i is the arc cosine of $\frac{I_i-I_{i+1}}{max(I_i,I_{i+1})}$.

	precision	recall
Real	85.19%	92.31%
Synthetic	80%	84.21%
Average	82.59%	88.26%

Table 5.5: Classification result for PhysioNet/CinC Challenge 2002 Dataset

We conduct our classification based on two simple hypotheses. One is that the number of pattern occurrences on each real record should be within a reasonable range because it is recorded from the people without cardiac abnormality. The other is that MSP would produce either only a few or a lot of patterns on synthetic records. On one hand, because every model has been used to generate only two time series, if one model is significantly different from the other models, the records would be hit only by a few patterns. On the other hand, if multiple models share similar mathematical models, the records would be hit by many patterns.

Therefore, we generate patterns over the dataset, check the occurrences of the patterns in each record and sort the records according to the occurrences of patterns. The records which are hit by only a few patterns or hit by a lot of patterns are classified to be synthetic. We can achieve high precision and recall in the classification for both synthetic and real data. As shown in Table 5.5, the average precision and recall are 82.59% and 88.26% respectively. We believe MSP is a promising tool to be deployed to various applications.

5.6 Summary

In all, we have introduced a new generic pattern mining approach MSP that employs a seed-and-extension framework. Different from the previous work, MSP is to find maximal patterns in sequential 3D datasets with the constraints of minimum support and minimum confidence. Every pattern is a group of similar sequential 3D objects. Pairwise patterns with short and fixed length are generated as the seeds. Vertical extension is conducted to find all hits of the seed patterns with the constraints and horizontal extension is to expand patterns to be the longest without loss of hits. MSP is likely to be deployed in many scientific datasets such as protein chains and spatial moving objects. In order to enhance the adaptability, DPS Algorithm is designed to automatically detect proper settings for the given dataset. Comparison study shows MSP is efficient and scalable in terms of both number of objects and object length. Applying MSP in protein family classification and PhysioNet/CinC Challenge 2002 dataset shows the maximal patterns produced by it are effective in real-life applications.

CHAPTER 6

Remotely Homology Detection Based on Protein 3D Structures

6.1 Introduction

Protein structure can elucidate its function in both general and specific terms as well as its evolutionary history [15, 53]. While remote homologies are evolutionarily divergent, and there is no significant amino acid sequence overlap to known proteins. It is now widely agreed that the similarities among distantly related proteins are often preserved at the level of their 3D structure, even when very little similarity remains at the sequence level [35]. In the absence of obvious sequence similarity, the association rules on 3D structures are meaningful to find the evolutionary relationship for remotely homologous proteins.

In our approach, every protein chain is represented as a 3D sequential structure. The C_α atoms are vertices and the links to connect adjacent C_α atoms are the edges.

The similarity between protein chains is evaluated by the summation of the feature difference on all the vertices. We employ sCluster for mining 3D structure patterns and build the binary classification rule group ($BCRG$) where every rule takes form of (m, C), m is the frequent substructure pattern and C is a class label. The possibility of the rule group to correctly differentiate two classes is referred to as the confidence.

Classifications for remotely homologous proteins pose a challenge for existing sequence-based methods because the 3D structure information should be considered. The current association rule mining algorithms [6, 24, 33, 98] are based on feature lists. Such approaches cannot be directly applied to protein structure datasets. This is because the frequent 3D substructure patterns should be extracted before generating rule groups and the occurrences of patterns cannot be simply determined by exact matching.

On the other hand, upon our investigation there are too few frequent substructure patterns that appear in 10 protein families while the number of protein families is typically in the order of tens to hundreds. Therefore, it seems reasonable to create highly accurate binary classifiers and to incorporate them to conduct multi-class classification effectively.

There are some studies on combining multiple binary classifications for multi-class classification. The naive $One - Vs - All$ approaches demonstrate good accuracy but may consume expensive computing resources [21]. The other approaches are recursively dividing the classes into two groups of classes [76]. Our proposed $DBCT$ works a different manner. Given N classes, $DBCT$ to iteratively conducts $N - 1$ $One - Vs - One$ contests to select the final candidate. The structural difference between two protein families is evaluated by a measurement, called family structural difference (fsd) and the binary classifications are iteratively carried out on the two families with the largest fsd value.

Our purpose is to conduct accurate classification for proteins especially for remote homologous proteins. Due to the low sequence identify to the known proteins, traditional approaches based on amino acid sequences are not effective enough. This prompts us to retrieve the protein 3D structure patterns to form an effective classification method. In this chapter, we present an algorithm for mining association rule groups on protein 3D structures. The rule groups are novelly incorporated to classify multiple classes using $DBCT$. The experiments illustrate the accuracy and the efficiency.

Unlike the existing approaches, our method employs the frequent protein structural patterns in 3D-coordinate as the main features to identify family and a new approach, i.e. $DBCT$, to incorporate binary classification rule groups for multi-class classification.

6.2 Preliminary

In this section, we introduce some basic definitions and algorithms that are useful for further discussion.

6.2.1 Definitions

In this study, datasets are derived according to SCOP database [63]. Every protein chain is described as 3D sequential structure where the vertices are C_α atoms and the edges are the links to connect neighboring C_α atoms. We represent the substructure of a protein chain as $S[b : e]$ where S stands for the protein chain, b and e stand for the beginning vertex and the ending vertex respectively. The structural similarity between two protein chains is determined by the features on

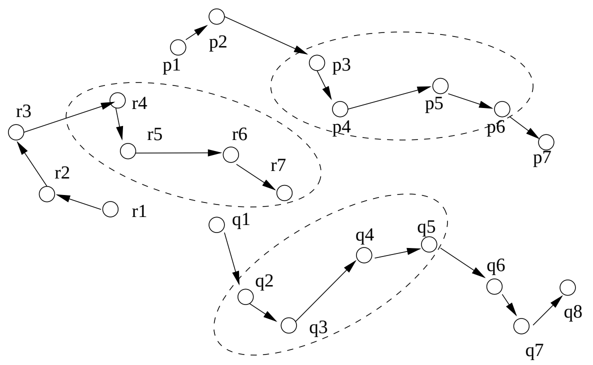


Figure 6.1: Sample motif: $\{(R[4:7], P[3:6], Q[2:5])\}$

the vertices including the distance between adjacent vertices, the angle between adjacent edges and the torsion angle between adjacent planes. It is defined as fds in Chapter 4.

The fds is the summation of feature difference on all vertices and it would be applied by sCluster for mining 3D structural patterns. Motifs are 3D substructure patterns which frequently appear in protein chains. Whether a structure pattern is eligible to be a motif is determined by the number of occurrences and the length of patterns. Moreover, because the protein structures vary in different environments and different technologies cause different measure errors, it is necessary to allow variations in the detection of motifs. Hence, we define motifs as the substructure patterns found from the given dataset C as below:

$motifs(C, w, u, \varepsilon)$ where

$C:$ *protein* $3D$ *structure dataset*

$u:$ *every motif includes at least u occurrences in C*

$w:$ *every occurrence of a motif includes at least w vertices*

$\varepsilon:$ *maximum fds distance between two occurrences*

Figure 6.1 shows an example with three chains in the dataset $C = \{R[1:7], Q[1:8], P[1:8]\}$ where minimum length $w = 4$, number of occurrences $u = 3$, and maximum fds distance $\varepsilon = 0.3$. There is one eligible motif, i.e.,

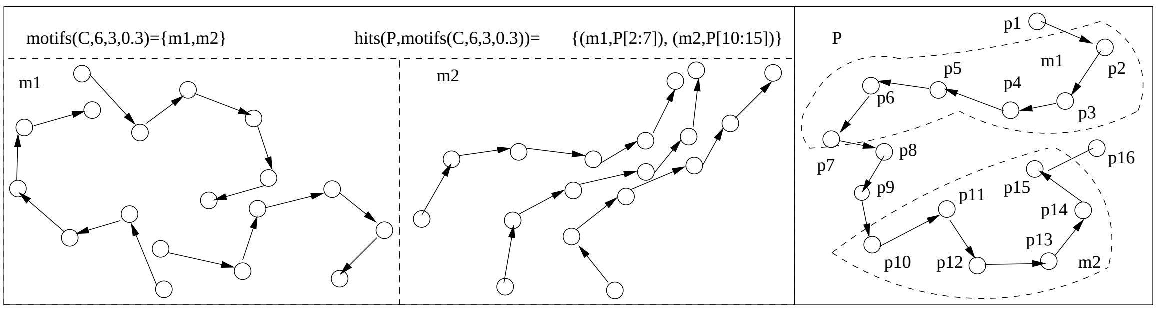

motifs(C,6,3,0.3)={m1,m2}
hits(P,motifs(C,6,3,0.3))= {(m1,P[2:7]), (m2,P[10:15])}
m1
m2
P
p1
p2
p3
p4
p5
p6
p7
p8
p9
p10
p11
p12
p13
p14
p15
p16
m1
m2

Figure 6.2: Example of hits: $\{(m_1, P[2:7]), (m_2, P[10:15])\}$

$motifs(C, 4, 3, 0.3) = \{(R[4:7], P[3:6], Q[2:5])\}$.

Motifs are employed to detect homologies among unknown and existing proteins. Given a new protein chain P and the motifs mined from existing protein dataset C, we define the hits as all occurrences of motifs in P, i.e., $hits(P, motifs(C, w, u, \varepsilon))$. Hits are a group of tuples taking forms of (t, A) where t is a substructure of P and A is a motif. The fds between t and a structure in A is no more than ε.

Figure 6.2 shows an example of hit. Here, $w = 6$, $u = 3$ and $\varepsilon = 0.3$. In protein P, there are two substructure $P[2:7]$ and $P[10:15]$ similar with motif m_1 and motif m_2 respectively. Therefore, two hits $(m_1, P[2:7])$ and $(m_2, P[10:15])$ are found.

6.2.2 Mining Motifs with Gaps

In order to classify proteins, we conduct mining on each protein family for frequent sequential 3D substructures as motifs using sCluster [83]. Since the original sCluster does not consider gaps, motifs found by sCluster are normally short and appearing frequently. However, it is important to consider insertions, deletions and replacements which are caused by mutations. In order to detect motifs with gaps, we conduct a heuristic extension process over the pairwise motifs obtained by sCluster.

In this study, we deploy the gapped alignment which is widely studied in sequences to sequential 3D structures. Gap penalty function similarly with the protein sequence extension [8] is applied to extend the pairwise motifs. Because a single mutational event may insert or delete a large number of residues, it has been argued that long gaps should not cost much more than short ones [8] and we adopt an affine gap penalty function as:

$$Gap\ Penalty = G + Ln$$

Here G is gap opening penalty, L is gap extension penalty and n is length of gap. We set $G = 11$ and $L = 1$. For every match, we compensate by a deduction of 4 on the gap penalty similarly with BlastP [8].

Since structures vary slightly due to the measurement errors and the internal variations in the protein, we propose a criteria to determine the match with an error tolerance δ. If a vertex on one structure $S_1[x : x]$ matches the corresponding vertex on the other structure $S_2[y : y]$, the summation of feature differences between $S_1[x : x]$ and $S_2[y : y]$ is no more than δ, i.e., $fds(S_1[x : x], S_2[y : y]) \leq \delta$. We define the extension problem as:

Problem 6.2.1 *Assume we have pairwise motif $\{S[a : b], P[c : d]\}$, we are going to extend it along left-hand, i.e., to find the integer pair (i, j) to maximize $i + j$ where the gap penalty between the alignment on $S[a - i : a - 1]$ and $P[c - j : c - 1]$ is less than a given maximum gap penalty. We call (i, j) the left-hand maximum extension offset. The extension should be also conducted along right-hand.*

Algorithm 6.2.2 describes the left-hand extension for Problem 6.2.1. The edit operation - insert, delete or replace, is determined for mismatches as shown in step 9 to 13. The gap opening cost would be imposed if it is the first mismatch after the previous matches. This algorithm is to consider local optimal alignments rather than global optimal alignments. In order to avoid too many motifs, we employ a

```
Algorithm 6.2.2: Left-hand extension on pairwise motifs
Input: pairwise motif: $\{S[a:b], P[c:d]\}$
maximum gap penalty: $MGP$,      error tolerance $\delta$
Output: left-hand extension offset $(i, j)$
Method:
Initialization
1.  $Penalty \leftarrow 0$, $IsExtension \leftarrow false$
2.  $i \leftarrow 1$, $j \leftarrow 1$;
Test boundaries and maximum gap penalty
3.  If $a - i < 1$ or $c - j < 1$ Then return $(i - 1, j - 1)$
4.  If $Penalty > MGP$ Then return $(i - 1, j - 1)$
Extend in case of matches
5.  If $fds(S[a - i : a - i], P[c - j : c - j]) \leq \delta$
6.  Then $i \leftarrow i - 1$, $j \leftarrow j - 1$
7.      $Penalty \leftarrow Penalty - 4$, $IsExtension \leftarrow false$
8.      Goto step 3
Extend in case of mismatches
9.  If $fds(S[a - i : a - i], P[c - j - 1 : c - j - 1]) \leq \delta$
10. Then $i \leftarrow i - 1$, $j \leftarrow j - 2$            /* Insert */
11. Else If $fds(S[a - i - 1 : a - i - 1], P, [c - j : c - j]) \leq \delta$
12.     Then $i \leftarrow i - 2$, $j \leftarrow j - 1$        /* Delete */
13.     Else $i \leftarrow i - 1$, $j \leftarrow j - 1$        /* Replace */
14. $Penalty \leftarrow Penalty + 1$                 /* gap cost */
15. If $\neg IsExtension$
16. Then $Penalty \leftarrow Penalty + 11$        /* gap opening cost */
17.     $IsExtension \leftarrow True$
18. Goto step 3
```

Figure 6.3: Left-hand extension on pairwise motifs

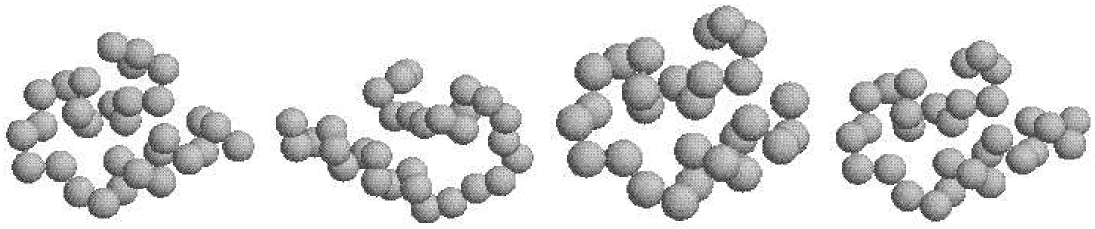


Figure 6.4: Sample motif: $\{d1dm2a[141:170], d1ckpa[144:173], d1b38a[157:186], d1aq11\_[144:173]\}$

tight δ ranging from 0.05 to 0.1 and a small MPG value ranging from 12 to 15. On the other hand, the right-hand extension would be conducted similarly with the left-hand extension.

After the extension, the pairwise motifs would be used to generate motifs with more occurrences by the modified-apriori algorithm in sCluster. We conduct mining on the protein kinases catalytic subunit protein chains and present a randomly selected motif in Figure 6.4. This motif includes 4 occurrences, $d1dm2a[141:170]$, $d1ckpa[144:173]$, $d1b38a[157:186]$ and $d1aq11\_[144:173]$. The similarity among the four occurrences is obvious. This sample shows that this method is effective to discover protein 3D structure patterns.

6.2.3 Mining Motifs as Specified

The motifs found from different classes are different and they usually vary with the settings. In order to efficiently generate enough motifs with given occurrences, we adopt DPS algorithm which was described in Chapter 5.3.4 to automatically detect suitable settings for the given protein family.

Problem 6.2.2 *Give protein chain dataset C, motif occurrences u, range of number of motifs $[n_{min} : n_{max}]$, we are going to find the proper settings including maximum distance ε and minimum motif length w to produce n motifs each of which appear at least u times in C, where $n \in [n_{min} : n_{max}]$.*

$(n, u) =$	(400,4)		(600,5)	
family	(ε, w)	$\|motifs\|$	(ε, w)	$\|motifs\|$
RN	(1.75,24)	480	(3.92,32)	693
HEC	(2.81,26)	413	(3.92,29)	578
EP	(2.81,30)	454	(3.92,33)	587
GST	(2.43,30)	435	(3.84,34)	667

Table 6.1: Datasets and settings

DPS Algorithm facilitates users by hiding the details of 3D structural pattern mining. Giving number of motifs (n) and the motif occurrence (u), it will self-tune (ε, w) to produce enough qualified motifs. For example, we are going to generate motifs on 4 protein families, rubrerythrin n-terminal domains (RN), human enterovirus B coat proteins (HEC), Eukaryotic proteases (EP) and Glutathione S-transferase (GST) with two groups of settings, $(n, u) = (400, 4)$ and $(n, u) = (600, 5)$. RN is a α protein family. Both HEC and EP are β protein families, and GST is a α/β protein family. There are 30 protein chains in each family, and the amino acid sequence length of protein chains ranges from 70 to 500. The protein chains in same family have sequence identities less than 30%. We set the ranges for ε and w as below:

$$[\varepsilon_{min} : \varepsilon_{max}] = [1.5 : 4],\ [w_{min} : w_{max}] = [15 : 35]$$

Table 6.1 shows the suitable settings under two different requirements $(n, u) = (400, 4)$ and $(n, u) = (600, 5)$. We set $[n_{min} : n_{max}] = [n - 100 : n + 100]$. For family RN, the setting identified by DPS to generate motifs for $(n, u) = (400, 4)$ is $(\varepsilon, w) = (1.75, 24)$ where 480 motifs are generated.

6.3 Binary Classification Rule Group

It is more possible for a motif to appear in the same family than in the other family. For motifs which appear in more than one families, they are not unique to differentiate families. Therefore, we adopt motifs appearing in only one family to build the association rule group. Given dataset $D = \{C_1, C_2\}$ where C_1 and C_2 are two protein families, with the number of motifs n and the motif occurrence u, we define the binary classification rule group ($BCRG$) as a group of 2-dimension tuples $(motif, family)$:

$$\begin{aligned}
& BCRG \\
= \; & \{(m_1, C_1) : m_1 \in motifs(C_1, w_1, u_1, \varepsilon_1) \\
& \wedge hits(m_1, motifs(C_2, w_2, u_2, \varepsilon_2)) = \emptyset\} \\
\cup \; & (m_2, C_2) : m_2 \in \{motifs(C_2, w_2, u_2, \varepsilon_2)\} \\
& \wedge hits(m_2, motifs(C_2, w_2, u_2, \varepsilon_2)) = \emptyset\} \\
& where\ w_i\ and\ u_i\ are\ generated\ by\ DPS(C_i, n, u)
\end{aligned}$$

For tuple (m_i, C_i) in the binary classification rule group, motif m_i appears in family C_i but does not appear in family C_j where $1 \leq i \neq j \leq 2$. The motifs in $BCRG$ are unique for each family and will be used by binary classifiers. To evaluate the distinctiveness of $BCRG$, we define the confidence of $motifs$ as:

$$\begin{aligned}
& conf(motifs(C_i, w_i, u_i, \varepsilon_i) : \{C_1, C_2\}) \\
= \; & 1 - \frac{|hits(m_i, motifs(C_j, w_j, u_j, \varepsilon_j))|}{|motifs(C_i, w_i, u_i, \varepsilon_i)|} \\
& Here, m_i \in motifs(C_i, w_i, u_i, \varepsilon_i) \\
& 1 \leq i \neq j \leq 2
\end{aligned}$$

Confidence of motifs from C_1 indicates the probability that the motifs only appear in C_1. Motifs with higher confidence could be more possible to detect correct family. However, too high confidence may lead to the over fit of training set so that the

```
Algorithm 6.3: Create BCRG
Input: Training set $D$: $\{C_1, C_2\}$
number of motifs: $n$,      motif occurrence: $u$
Output:$BCRG$
Method:
1. $(w_1, \varepsilon_1) \leftarrow DPS(C_1, n, u)$
2. $(w_2, \varepsilon_2) \leftarrow DPS(C_2, n, u)$
3. $R_1 \leftarrow motifs(C_1, w_1, u, \varepsilon_1)$
4. $R_2 \leftarrow motifs(C_2, w_2, u, \varepsilon_2)$
5. $Result_1 \leftarrow \emptyset$; $Result_2 \leftarrow \emptyset$
6. For each $i$ from 1 to 2
7.     For each motif $m \in R_i$, $1 \leq i \neq j \leq 2$
8.       If $hit(m, R_j, L_j, \varepsilon_j) = \emptyset$
9.       Then $Result_i \leftarrow Result_i \cup \{(m, C_i)\}$
10. Return $Result_1$ and $Result_2$
```

Figure 6.5: Create $BCRG$s

motifs could not hit the unknown protein structures even if they are from the same family with the motif.

Algorithm 6.3 describes the process to produce $BCRG$. For $m \in motif(C_i, w_i, u_i, \varepsilon_i)$, if $hit(m, motif(C_i, w_i, u_i, \varepsilon_i) = \emptyset$, then add it into $BCRG$. Two binary classification rule groups, $Result_1$ and $Result_2$, are generated to identify C_1 and C_2 respectively.

6.4 Binary Classification Tree

There are some research works for multi-class classification based on binary classifications. An approach for handling multi-class classification is to generate all possible 2-class classifiers between K classes from training examples. This approach is known as $One - Vs - All$ method which generates $K(K-1)/2$ binary classifiers given K classes of training examples. In case of many protein families,

$One - Vs - All$ strategy is computationally expensive. To avoid this situation, we create a new binary classification tree with height of $K-1$ to support K-class classification.

6.4.1 Family Structural Difference

In SCOP database, proteins are grouped into classes on the basis of their secondary structures. Every class is further classified into multiple families. Motifs illustrate the 3D structural features of protein family. Given two protein families C_1 and C_2, number of motifs n and motif occurrence u, a measurement is defined to evaluate $family\ structural\ difference$ as

$$\begin{aligned} & fsd(C_1, C_2, n, u) \\ = \ & conf(motifs(C_1, w_1, u, \varepsilon_1) : \{C_1, C_2\}) \\ \times \ & conf(motifs(C_2, w_2, u, \varepsilon_2) : \{C_1, C_2\}) \\ & (w_i, \varepsilon_i)\ is\ generated\ by\ DPS(C_i, n, u) \end{aligned}$$

The fsd is the product of the motif confidences on two protein families. The higher fsd value, the more different the two families are. In the later section, experimental results show that fsd is reasonable and effective to tell structural difference between two families.

6.4.2 Deterministic Binary Classification Tree

Obviously, it's easier to differentiate two significantly different families than two similar families. Binary classification tree is an order to iteratively partition group into two subgroups. A height of $n-1$ binary classification tree, where n is the number of classes, leads to a multi-class classification method based on $n-1$ binary classifications ordered by the tree. Leveraging on fsd value, a deterministic binary classification tree is constructed where the difference between the two classes in every node is the largest within the subtree rooted by the node.

Definition 6.4.1 *Deterministic binary classification tree* $(DBCT)$

Node=

$$\begin{cases} pair & (C_l, C_r) \;\; C_l, C_r \in C \\ L & pointer\ to\ left\ child\ node \\ R & pointer\ to\ right\ child\ node \end{cases}$$

DBCT(C)=

$$\begin{cases} null & if\ C = \emptyset \\ Node((C_1, C_1), null, null) & if\ |C| = 1 \\ Node(\quad (C_l, C_r) : & if\ |C| \geq 2 \\ |fsd(C_l, C_r, n, u)| & \\ = max(|fsd(C_i, C_j, n, u)| : C_i, C_j \in C) & \\ pointer\ to\ DBCT(C - \{C_l\}), & \\ pointer\ to\ DBCT(C - \{C_r\}) \quad) & \end{cases}$$

Here n is the number of motifs, u is the motif occurrence.

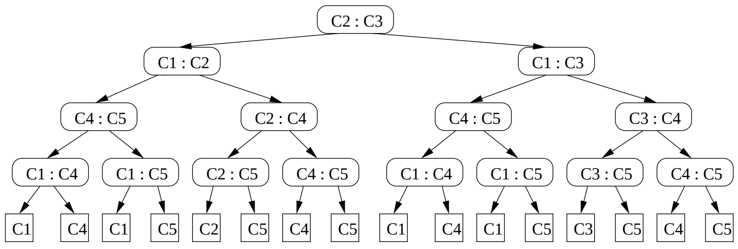


Figure 6.6: $DBCT(\{C_1, C_2, C_3, C_4, C_5\})$

conf	C_1	C_2	C_3	C_4	C_5
C_1	-	0.9	0.8	0.3	0.2
C_2	0.6	-	0.6	0.4	0.9
C_3	0.5	1	-	0.5	0.5
C_4	0.7	0.8	0.7	-	0.6
C_5	0.9	0.3	0.4	0.4	-

Table 6.2: Families and confidences

(Fam_1, Fam_2)	fsd	(Fam_1, Fam_2)	fsd
(C_1, C_2)	0.54	(C_1, C_3)	0.4
(C_1, C_4)	0.21	(C_1, C_5)	0.18
(C_2, C_3)	0.6	$(C_2, C_4$	0.32
(C_2, C_5)	0.27	(C_3, C_4)	0.35
(C_3, C_5)	0.2	(C_4, C_5)	0.24

Table 6.3: Families and fsd

Table 6.2 and 6.3 describe $motif$ confidences and fsd value between any two families on sample dataset $\{C_1, ..., C_5\}$ respectively. Cell (i-th row, j-column) in Table 6.2 is $conf(m \rightarrow C_i : \{C_i, C_j\})$. Cell($i$-th row, j-column) in Table 6.3 is $fsd(C_i, C_j, n, u)$. For example, Table 6.2 shows that $conf(m \rightarrow C_1 : \{C_1, C_2\}) = 0.9$ and $conf(m \rightarrow C_2 : \{C_1, C_2\}) = 0.6$. We see $fsd(C_1, C_2) = 0.9 \times 0.6 = 0.54$ in Table 6.3.

In this sample, the largest fsd value is between C_2 and C_3, so the binary classifier on C_2 and C_3 is possibly the most accurate. Hence, (C_2, C_3) should be at the root of binary classification tree. Given a test protein, if it is determined by $BCRG$ on C_2 and C_3 to be more possible to belong to family C_2, it should be the input of the $BCRG$ between two families which are the most different excluding C_3, and this $BCRG$ is located at the left node of node (C_1, C_3).

The two classes in the root of $DBCT$ is always the two classes with the largest fsd value. The left-hand subtree under node $((C_l, C_r), l, r)$ recursively depicts the $DBCT$ which is created on families without C_r. Each level of non-leaf node means

```
Algorithm 6.4.2: Create $DBCT$
**Input:** $D = \{C_1, ..., C_t\}$
minimum number of motifs: $n$,      motif occurrence: $u$
**Output:** $DBCT(D, n, u)$
**Method:**
1. if $t = 0$ then return null
2. if $t = 1$ then return node$((C_1, C_1), null, null)$
3. $r \leftarrow new\ node$
4. $r.pair \leftarrow (C_l, C_r)$
where $fsd(C_l, C_r)$=$max(fsd(C_i, C_j) : C_i, C_j \in C)$
5. $r.left \leftarrow DBCT(D - \{C_l\})$
6. $r.right \leftarrow DBCT(D - \{C_r\})$
7. return $r$
```

Figure 6.7: Create $DBCT$

one binary classification. So in total there would be $n-1$ $BCRG$s used for n-family in one protein classification. Every protein classification follows through one path from the root to leaf node which specifies the order of binary classifiers. Algorithm 6.4.2 describes the construction process of the deterministic binary classification tree.

6.5 Experiments

We implement the multi-family classification proteins in C. All experiments are done on a PC with a Pentium 4 2.6Ghz CPU, 1GB of SDRAM and a 7200rpm 40GB hard disk running Windows XP. A group of experiments are designed to study the effect of settings and multi-family classification accuracy as well as the performance.

Family	Abbreviation
Rubrerythrin n-terminal domains	RN
Human enterovirus B coat	HEC
Eukaryotic proteases	EP
Glutathione S-transferase	GST
Glycosyl hydrolase domains	GH
Tyrosine-dependent oxidoreductases	TDO
C1 set domains	CS
Amylase, catalytic domains	AC

Table 6.4: Families and abbreviations

6.5.1 Dataset

The dataset for experiments is downloaded from

ftp://vax2.fccc.edu/dunbrack/pub/culledpdb/cullpdb_pc30_res3.0_R1.0_d040717_chains3672.gz.

The sequence identity between any two protein chains is less than 30%. We check the corresponding family for each protein in SCOP database. Proteins in the same family are remotely homologous. 320 protein chains from 8 families each of which includes 40 protein chains are randomly selected. The 8 families are listed in Table 6.4. In case that the protein chains are less than 40, we randomly select other chains in the same family from SCOP database to make up.

6.5.2 Accuracy of Binary Classifier

Since binary classifiers are the foundation of $DBCT$, we conduct experiments to investigate the accuracy of binary classifier on real data. Two datasets that we test are same with that are used in [38]. The first dataset (C_1) includes two protein families that belong to two different SCOP classes. The first family is the nuclear receptor ligand-binding domain proteins (NB) from all alpha class and the second one is the prokaryotic serine protease family (PSP) from all β-class. The second dataset (C_2) includes the families of eukaryotic serine proteases (ESP) and

Dataset	LG DT	LG AD(0.1)	LG CD	$DBCT$	(n, u)
C_1	100%	100%	95%	100%	(600,4)
C_2	95%	95%	98%	100%	(600,4)

Table 6.5: Accuracy of binary classifier - 1

Families	Accuracy	Families	Accuracy
(RN, HEC)	100%	(RN, EP)	100%
(RN, GST)	100%	(HEC, EP)	100%
(HEC, GST)	100%	(EP, GST)	100%

Table 6.6: Accuracy of binary classifiers - 2

the prokaryotic serine proteases (PSP). These two families belong to the same superfamily. All proteins included in datasets C_1 and C_2 were selected from the culled PDB list (http://www.fccc.edu/ research/labs/dunbrack/pisces/culledpdb.html) with no more than 60% pair-wise sequence similarity in order to remove highly homologous proteins. The comparison results are presented in Table 6.5. Here $LG - DT$, $LG - AD(0.1)$ and $LG - CD$ represent three methods respectively proposed in [38]. The setting is $(n, u) = (600, 4)$. We see that our method correctly predict the families for all test cases.

We also evaluate the accuracy on a portion of the downloaded dataset which includes 160 protein chains from the 4 families, RN, HEC, EP and GST. The dataset is divided into training set and test set which include 120 and 40 protein chains repectively. Here the setting is $(n, u) = (600, 4)$. Table 6.6 lists the accuracies in all the 6 binary classifications. For every test protein chain, the binary classifiers predict its family correctly. The high accuracy of binary classification rule groups forms a solid foundation for multi-family classification.

			setting 1		
family	RN	HEC	EP	GST	Average
RN	-	100%	100%	100%	100%
HEC	97.34%	-	64.09%	77.39%	79.61%
EP	82.64%	17.88%	-	63.73%	54.75%
GST	84.04%	82.79%	72.57%	-	79.80%

			setting 2		
family	RN	HEC	EP	GST	Average
RN	-	100%	100%	100%	100%
HEC	63.71%	-	16.46%	62.87%	47.68%
EP	38.92%	2.96%	-	25.62%	22.50%
GST	85.46%	82.62%	75.18%	-	81.09%

Table 6.7: Motif confidences, setting 1: $(n, u) = (600, 4)$, setting 2: $(n, u) = (200, 5)$

6.5.3 Confidence

Given a protein family, the number of motifs is determined by three parameters, maximum distance ε, motif occurrence u and minimum motif length w. We target to find more motifs with higher occurrence where larger maximum distance and shorter minimum motif length are required. On the other hand, with the relaxing of motif detection conditions, many motifs appear in multiple families and they deteriorate confidence. We design a group of experiments on a 4-family dataset to study the relationship among n, u and confidence.

In Table 6.7, cell(i-th column, j-th row) is the confidence of motifs from the i-th family versus the j-th family, i.e.:

$$conf(motifs(family_i, w_i, u, \varepsilon_i) : \{family_i, family_j\})$$

where (w_i, ε_i) is determined by $DPS(family_i, n, u)$. In this experiments, $(n, u) = (600, 4)$. All motifs from RN only appear in RN while motifs from EP frequently occur in other families. The reason is that RN is an α-protein family whereas HEC and EP are β-protein family and GST is $\alpha + \beta$-protein family. Upon our investigation, most of the motifs in RN are consisted of α-helix structures linking with other structures. For example, Figure 6.4 depicts a motif which is consisted of two helixes connected by a group of amino acids located in a line.

Since both family HEC and family EP are beta proteins which include many β-strands, the 3D structural similarity is high even though the sequence identify between any two protein chains is trivial. So it can be observed that the motif confidence for family EP to differentiate family HEC is 17.88%. On the other hand, the confidence for motifs to differentiate family EP from RN is 97.34% and the confidence for motifs to differentiate family HEC is 82.64%. This is because family RN is α protein family which is significantly different from family HEC and EP which are β-protein families. We also see the average confidences of motifs with setting 1 are higher than that with setting 2 because higher motif occurrence incurs tighter motif detection conditions.

6.5.4 Precision And Recall

The downloaded dataset is divided into a training set with 240 protein chains and a testing set with 80 protein chains. There are 30 and 10 protein chains for each family in training set and in testing set respectively. We build $DBCT$ over the training set with $(n, u) = (600, 4)$ and predict the families for protein chains in testing set. As a benchmark of $DBCT$, $One - Vs - All$ methodology is also implemented to evaluate both effectiveness and efficiency of $DBCT$.

Table 6.8 describes the precision and recall of $DBCT$ and $One - Vs - All$ on every family in the dataset. We observe that the average precision and recall of $DBCT$ are 89.36% and 78.75% respectively. In most of the families, the precision is 100%. The precision in family TDO is 43.5% because it's a α and β protein family where α-helix and β-strand are largely interspersed and would be regarded as independent motifs. Recalls in the families are also good. For GST and AC,

Family	Precision $DBCT$	Precision $One - Vs - All$	Recall $DBCT$	Recall $One - Vs - All$
RN	100%	100%	100%	100%
HEC	100%	100%	80%	80%
EP	100%	100%	80%	80%
GST	100%	100%	50%	60%
GH	71.4%	76.9%	100%	100%
TDO	43.5%	52.6%	80%	80%
CS	100%	100%	90%	90%
AC	100%	100%	50%	60%
Average	89.36%	91.19%	78.75%	81.25%
Time(s)	0.83	7.7		

Table 6.8: $DBCT$ VS. One-Vs-All on precision and recall

recall is 50% because both GST and AC are α and β protein families where the test protein chains are easily hit by the motifs in forms of α-helix and β-strand.

We also notice that the precision and recall of $DBCT$ are very close to $One - Vs - All$ method. However, the average response speed for $DBCT$ is 0.83 second per prediction which is 9 times faster than $One - Vs - All$. The baseline accuracy [89, 97] for the classification using amino acid sequence information alone was recently created as 69.6% for proteins with sequence identity less than 35%. Therefore, it could be concluded that $DBCT$ is an effective and efficient mechanism to incorporate binary classification rule groups to conduct multi-class classifications.

6.6 Summary

In summary, we have created a new approach for protein classification purely based on 3D structures. It is a good complement to existing classification methods for remotely homologous proteins whose sequence identities to known proteins are low while the functionalities are similar.

First, sCluster is adopted and enhanced for mining frequent 3D structural patterns with gaps from protein families. Second, motifs are used to build binary classification rule groups ($BCRG$). Experimental results illustrate that the accuracy reaches 100% in all test cases. In order to support multi-class classification, the deterministic binary classification tree ($DBCT$) is proposed to incorporate $n-1$ $BCRG$s. In the comparison experiments with $One - Vs - All$ method, $DBCT$ significantly enhances the efficiency while its precision and recall are close to the $One - Vs - All$ method. We believe our method can be a new direction for detecting the isomorphism and remote homologies in various real-life biological and pharmaceutical applications.

CHAPTER 7

FCDR: Finding Coding DNA Regions for Similar 3D Protein Structures

7.1 Introduction

It is commonly accepted that proteins with similar functions can share similar 3D structures but different amino acid sequences. Proteins are the results of translation from DNA sequences where DNA sequences are transcribed into messenger RNA (mRNA) and further translated into proteins. Every three nucleotides (codon) determine which amino acid will be added next in the growing protein chain. This translation is conducted by ribosomes. They start to assemble on the first AUG (start codon) in mRNA and read the rest of codons sequentially. When a stop codon is encountered (UAA, UAG, or UGA), mRNA and the ribosome will be dissociated.

In this chapter, we describe our FCDR System for mining DNA sequences which

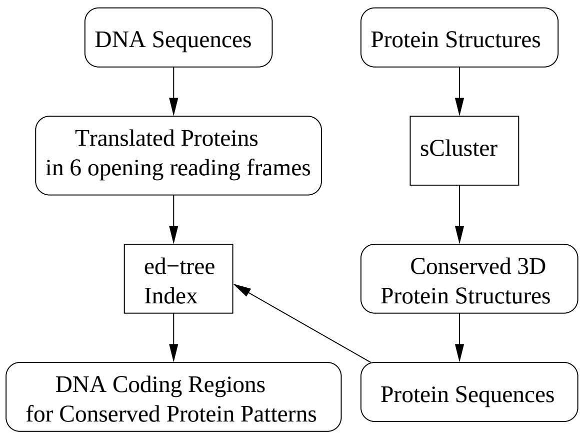

DNA Sequences
Protein Structures
Translated Proteins in 6 opening reading frames
sCluster
ed−tree Index
Conserved 3D Protein Structures
DNA Coding Regions for Conserved Protein Patterns
Protein Sequences

Figure 7.1: Architecture of FCDR System

code conserved 3D structures in a given protein dataset. There are many tools for translating DNA to protein sequences. However, according to our knowledge, no research has been done on translating a protein 3D structure to DNA sequence while it is helpful to detect the DNA for the cause of disease and to identify biochemical compounds for disease resistance.

7.2 Problem Description

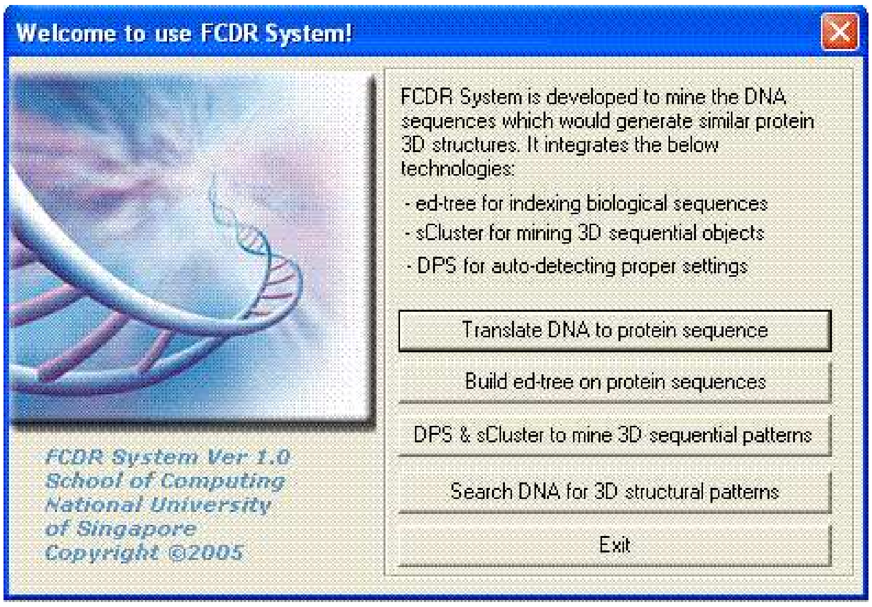
Given DNA sequences and protein 3D structures of a biological organism, we are going to mine the DNA subsequences which code the frequently appearing 3D structure patterns in this organism.

7.3 System Architecture

We create FCDR system for mining DNA sequences which code similar 3D protein structures. The main interface is shown in Figure 7.2 which includes:

1. Translate DNA to protein sequence.

2. Build $ed - tree$ on protein sequences.


Welcome to use FCDR System!

FCDR System is developed to mine the DNA sequences which would generate similar protein 3D structures. It integrates the below technologies:

- ed-tree for indexing biological sequences
- sCluster for mining 3D sequential objects
- DPS for auto-detecting proper settings

Translate DNA to protein sequence
Build ed-tree on protein sequences
DPS & sCluster to mine 3D sequential patterns
Search DNA for 3D structural patterns
Exit

FCDR System Ver 1.0
School of Computing
National University
of Singapore
Copyright ©2005

Figure 7.2: Main interface of FCDR System

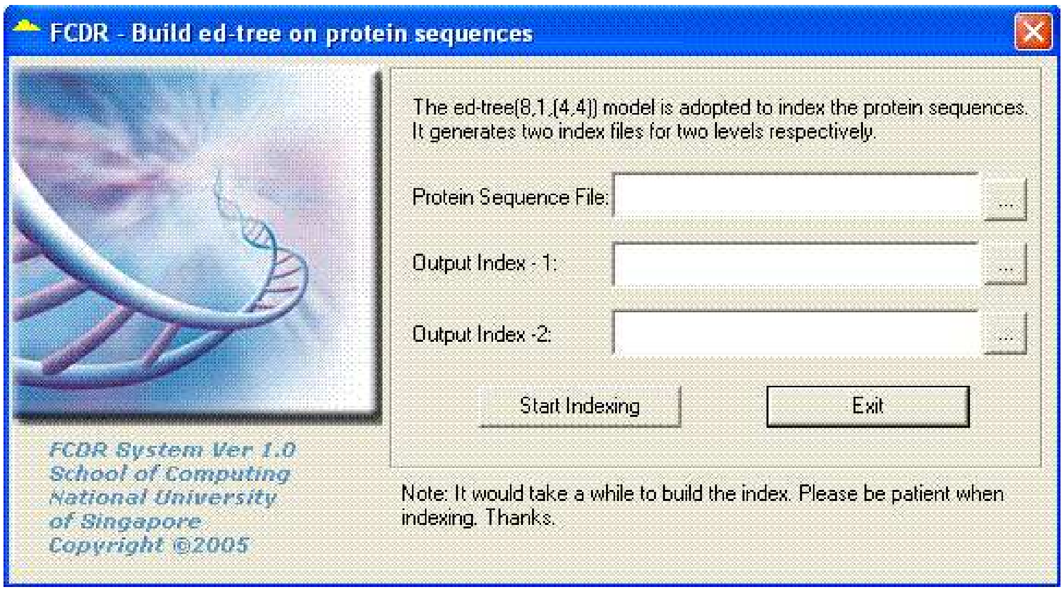
3. DPS & sCluster to mine 3D sequential patterns.

4. Search DNA for 3D protein patterns.

Component "Translate DNA to protein sequence" and component "Build $ed-tree$ on protein sequences" are the preprocessing on DNA sequences of target organism. Component "DPS & sCluster to mine 3D sequential patterns" is the preprocessing on protein 3D structures of target organism. Component "Search DNA for 3D protein patterns" integrates Chime [61] visualization control with searching method on protein $ed-tree$. Users can view the frequent 3D structure patterns, amino acid sequences and conduct query for the coding DNA sequences.

7.3.1 Translate DNA to Protein Sequence

For DNA sequences, we firstly translate them into protein amino acid sequences in 6 open reading frames using the facilities provided by bioinformatics.org. Since every three nucleotides codes one amino acid, the translation can start at 3 position. On


FCDR - Build ed-tree on protein sequences

The ed-tree(8,1,(4,4)) model is adopted to index the protein sequences. It generates two index files for two levels respectively.

Protein Sequence File:

Output Index - 1:

Output Index -2:

Note: It would take a while to build the index. Please be patient when indexing. Thanks.

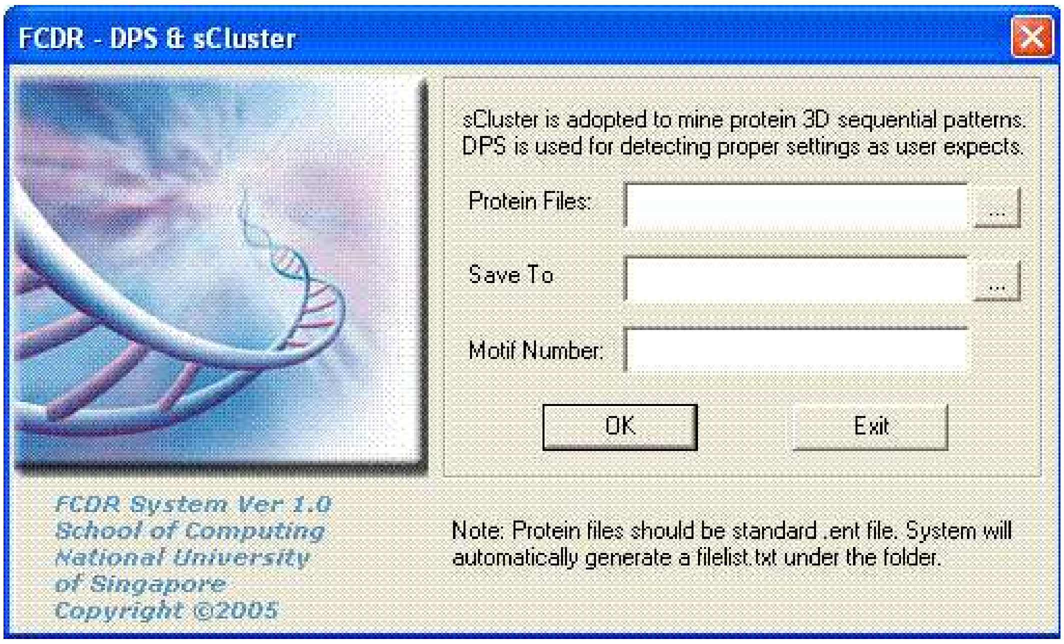
FCDR System Ver 1.0
School of Computing
National University
of Singapore
Copyright ©2005

Figure 7.3: Interface of building $ed - tree$ on proteins

the other hand, it can be translated either directly or reversely. Therefore there are 6 open reading frames for translating DNA sequences.

7.3.2 Build $ed - tree$ on Protein Sequences

Secondly, we build an $ed - tree(8, 1, (4, 4))$ index for the translated protein sequences. Since there are 20 distinct amino acids, we select 8 as seed length. Sliding window with size of 1 character is adopted to generate subsequences which would be indexed in $ed - tree$. Each subsequence is partitioned into two parts, each of which is consisted of 4 characters. Compared to the $ed-tree$ for DNA, the $ed-tree$ for proteins is to index shorter sequences with 2 levels only. Figure 7.3 shows the interface for users to build index on protein sequence file.


FCDR - DPS & sCluster

sCluster is adopted to mine protein 3D sequential patterns. DPS is used for detecting proper settings as user expects.

Protein Files:

Save To

Motif Number:

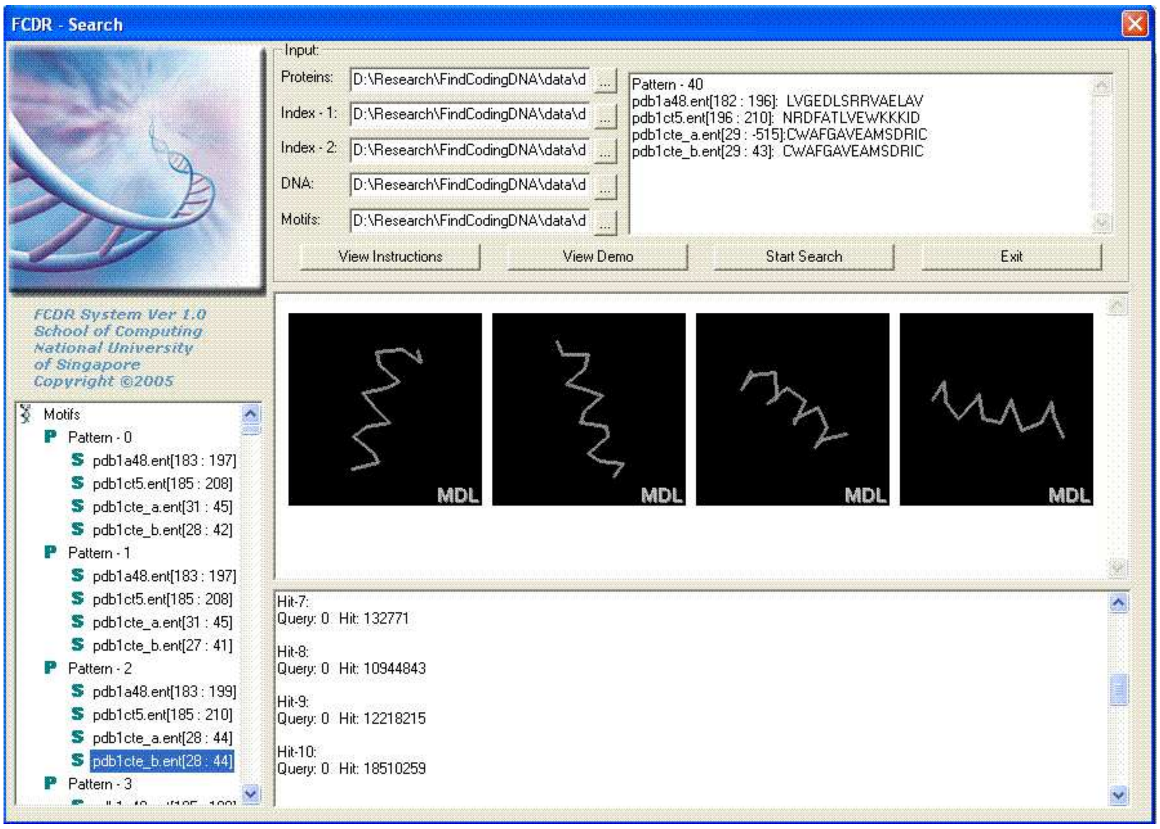
FCDR System Ver 1.0
School of Computing
National University
of Singapore
Copyright ©2005

Note: Protein files should be standard .ent file. System will automatically generate a filelist.txt under the folder.

Figure 7.4: Interface of mining protein 3D patterns

7.3.3 DPS & sCluster to Mine Similar 3D Protein Structures

We use sCluster to mine conserved 3D protein structures. As displayed in Figure 7.4, users can specify the number of expected 3D structure patterns. DPS will automatically detect proper settings for sCluster to mine the substructure patterns as users' expectation. To enable a better understanding, we apply Chime control [61] to visualize results in an interactive manner. Users can adjust viewpoints of the structure pattern using mouse and the corresponding amino acid sequences are extracted and listed.



Proteins: D:\Research\FindCodingDNA\data\d
Index - 1: D:\Research\FindCodingDNA\data\d
Index - 2: D:\Research\FindCodingDNA\data\d
DNA: D:\Research\FindCodingDNA\data\d
Motifs: D:\Research\FindCodingDNA\data\d

```
Pattern - 40
pdb1a48.ent[182 : 196]:  LVGEDLSRRVAELAV
pdb1ct5.ent[196 : 210]:  NRDFATLVEWKKKID
pdb1cte_a.ent[29 : -515]:CWAFGAVEAMSDRIC
pdb1cte_b.ent[29 : 43]:  CWAFGAVEAMSDRIC
```

FCDR System Ver 1.0
School of Computing
National University
of Singapore
Copyright ©2005

```
Motifs
  Pattern - 0
    pdb1a48.ent[183 : 197]
    pdb1ct5.ent[185 : 208]
    pdb1cte_a.ent[31 : 45]
    pdb1cte_b.ent[28 : 42]
  Pattern - 1
    pdb1a48.ent[183 : 197]
    pdb1ct5.ent[185 : 208]
    pdb1cte_a.ent[31 : 45]
    pdb1cte_b.ent[27 : 41]
  Pattern - 2
    pdb1a48.ent[183 : 199]
    pdb1ct5.ent[185 : 210]
    pdb1cte_a.ent[28 : 44]
    pdb1cte_b.ent[28 : 44]
  Pattern - 3
```

```
Hit-7:
Query: 0  Hit: 132771

Hit-8:
Query: 0  Hit: 10944843

Hit-9:
Query: 0  Hit: 12218215

Hit-10:
Query: 0  Hit: 18510259
```

Figure 7.5: Interface of searching DNA sequences for protein 3D structures

7.3.4 Search Coding DNA Regions for 3D Protein Structures

After preprocessing on DNA sequences and protein 3D structures, users can view 3D structure patterns and conduct search for the coding DNA sequences as shown in Figure 7.5. A web browser control is located at the middle of this interface and it initially presents the instructions. When users select motif file, patterns will be listed into the tree viewer and the selected pattern will be visualized by Chime control. At the same time, amino acid sequences on the selected pattern are presented also. Users can easily identify conserved 3D structures which are coded by different DNA sequences.

The search is to find out sequences similar with those appearing on the conserved 3D patterns from amino acid sequences translated from DNA dataset. The positions, annotations and detailed DNA sequences would be illustrated.

FCDR System is an intuitive and efficient tool for researchers to identify DNA coding regions which generate similar 3D structures. It is helpful for studying DNA homologies.

7.4 Experiments

We implement the kernel algorithms in C++ and build GUI for FCDR System in Visual C++. FCDR System works on Windows XP or Windows 2000/2003. All experiments are done on a PC with a Pentium IV 2.6Ghz CPU, 1GB of RAM and a 7200rpm 80GB hard disk running Windows XP. A group of experiments are designed to study user experience as well as performance.

7.4.1 Datasets

We conduct a study on yeast saccharomyces cerevisiae which is commonly known as the baker's budding yeast. The saccharomyces genome database has been downloaded from www.ncbi.nlm.nih.gov. The dataset includes about 12 million base pairs of nucleotides. 32 protein 3D chains in yeast saccharomyces are selected from SCOP database. On average, each protein chain is consisted of 269 amino acids.

7.4.2 Preprocessing on DNA Sequence Dataset

Firstly, DNA sequences are translated to amino acid sequences in 6 open reading frames. The results are saved in FASTA format, for example:

```
>gi|Saccharomyces cerevisiae chromosome I|reading frame 1|direct
FIINFLYIYYIIILIYIIKIIFIIKIFILLSGFRLPWPGPGIIN***IIINN
```

>gi|Saccharomyces cerevisiae mitochondrion chromosome I|reading frame 1|reverse

YGYYIIIKHISV**YNYSNIYIYITFFY*YIYMDSFLKRGSVPLPLGRGPSL

Lines with prefix of "$> gi$" are the annotations that illustrate DNA source, position of starting translation and translation direction. These annotations would be referred to in result presentation. After translation, yeast saccharomyces cerevisiae protein sequences include about 24 million amino acids.

Secondly, we build an $ed - tree(8, 1, (4, 4))$ index on the protein sequences and produce two index files.

7.4.3 Preprocessing on Protein 3D Structure Dataset

With 32 yeast saccharomyces cerevisiae structures, we use DPS and sCluster to mine 300 patterns and 500 patterns separately. Table 7.1 depicts the auto-detected settings (w, ε) and the number of obtained patterns. Here, w is the minimum sCluster length and ε is the maximum distance.

Expected # of patterns	# of obtained patterns	(w, ε)	Processing time (s)
500	459	(15,3.81)	17.6
300	354	(18,2.79)	12.2

Table 7.1: Settings and results of DPS and sCluster on saccharomyces cerevisiae yeasts

To generate 500 patterns, DPS automatically determines the setting as $(w, \varepsilon) = (15, 3.81)$. Hence, sCluster algorithm produces 459 patterns, each of which includes 4 substructures.

7.4.4 Visualization And Query

After the sCluster process, all patterns and their substructures are listed as shown in Figure 7.5. We randomly plot one pattern as shown in Figure 7.6.

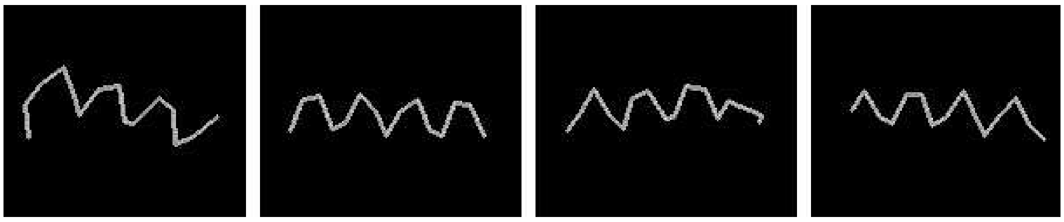


Figure 7.6: Sample pattern in FCDR System

The related information is listed as below:

$pdb1a48[183:197]$	$VGEDLSRRVAELAVK$
$pdb1ct5[197:211]$	$RDFATLVEWKKKIDA$
$pdb1cte\_a[26:40]$	$CGSCWAFGAVEAMSD$
$pdb1cte\_b[30:44]$	$WAFGAVEAMSDRICI$

Because there are 20 distinct amino acids, the enumeration space will be too large if the seed is long. We adopt $pmodel(8,1,2)$ for searching homologous protein sequences. The subsequence with edit distance no more than 2 to any one of the subsequences in the pattern would be found out. Sequences in the above pattern hit 185 subsequences in the translated dataset. The query takes 3.5 seconds. One selected sample output is depicted as below:

```
Hit-3: Query - 0      Seed - 15955340
> gi|6322960|ref|NC_001144.1|Saccharomyces cerevisiae chromosome XII
complete chromosome sequence > frame|open reading frame 1|reverse
Query:    VGEDLSRRVAELAVK
Seed:      GVGELSRR
DNA:     TGGTTTAAAAGTCATACTTCTCTA
```

In this hit, subsequence $VGEDLSRR$ from $pdb1a48[183:197]$ has edit distance 2 to $GVGELSRR$, i.e.

$$ed(\text{“}VGEDLSRR\text{”}, \text{“}GVGELSRR\text{”}) \leq 2$$

“GVGELSRR” is translated from DNA subsequence “TGGTTTAAAAGTCAT-ACTTCTCTA” which is located at 15955340 in Saccharomyces Cerevisiae chromosome XII with the reverse direction on open reading frame 1.

7.5 Summary

FCDR System is a method for biological researchers to mine frequent protein 3D structures and to query their coding DNA regions. It integrates the module for translating DNA sequences to protein sequences, the module for indexing protein sequences, the module for visualizing protein 3D structures and the module for searching DNA regions which code selected 3D structures. In our experiment, we also found out many DNA coding regions for protein structure patterns in the saccharomyces cerevisiae yeast.

CHAPTER 8

Conclusions

8.1 Thesis Findings

In this thesis, we studied some important issues about query and mining in biological databases, mainly about DNA sequences and protein structures. We investigated the existing work and identified the problems which were not well solved or fresh but important. We proposed new approaches to each problem. Our research results are meaningful and valuable compared to the previous research. Also our results presented some interesting research directions and the potentials to be applied to real-life applications.

Our first target was to create a fast similarity search method in large DNA sequence database on desktop PC. The motivation was because DNA sequence databases become larger and larger, and biologists often hope to create a query system on their own desktop PC with limited memory and CPU. While the previous works mainly were either based on sequential scan or suffix structures which suffer from the memory consuming for datasets as large as the whole human genome.

In Chapter 3, we proposed the *ed-tree* for indexing large DNA sequence databases with almost fixed-sized index. A new probe model has been designed to detect valuable local alignment candidates. Compared to the popular method Blastn, our probe model is more sensitive and able to detect longer candidates. Theorem 3.3.1 was explored to exactly calculate edit distance on probes in a more efficient manner compared to the classical dynamic programming. This enabled to detect insertion and deletion when generating candidates. Compared to the exactly-matching seed model or the seed detection models with only replacements, our probe model was substantially sensitive because it allowed gaps which were meaningful in biological applications such as discovering mutations and evolutionary relationships.

A new index, *ed-tree*, was devised to organize probes and the positions of their occurrences. Experiments showed that index size was relatively fixed and moderate. Search algorithm has been implemented based on *ed-tree*. For large-sized DNA sequence databases such as 1.5-2Gbps, *ed-tree* system can be 3-6 times faster than BlastN on desktop PC without loss of result effectiveness.

To extend the homology mining to protein structures, we discussed the problem of finding structure patterns in sequential 3D datasets. Mining sequential patterns with respect to 3D coordinates has not been studied well but appear in various important applications such as protein chains, moving objects and so on. This motivated us to conduct a study on this topic.

In Chapter 4, we defined $feature\ difference\ summation$ (fds) for evaluating the dissimilarity between two sequential 3D objects. Since fds is the difference summation of the selected features on all the vertices in the sequential 3D object, it could be simpler and more efficiently compared to the traditional measurement *rmsd*. Experiments showed it was effective for detecting frequent patterns. We defined sCluster model to formulate the subspace clustering problem. To avoid redundancy, *maximal* sCluster was defined with respect to both the length and the occurrences of patterns. As a foundation of mining all *maximal* sClusters, we have given out an algorithm to find the longest sClusters on two objects. Compared to the naive algorithm for this problem which has a computational complexity of $O(L^3)$ (wherein L is the object length), our algorithm reduced the complexity to $O(L^2 \lg L)$. From the pairwise structural patterns, we applied the *apriori* approach with extensions to produce all *maximal* sClusters lever-by-level without loss. In order to study the performance and effectiveness, we built a naive algorithm for mining *maximal* sClusters. Experiments showed that sCluster could be faster than naive algorithm by magnitudes. The randomly selected results illustrated the accuracy and effectiveness of our approach. With sCluster, biologists and pharmaceutists can detect protein structure patterns without regard to amino acid sequence identity, and can get a shortcut to find bundling proteins for some disease organisms. It is applicable and meaningful in real-life applications.

In order to find the **M**aximal **S**equential 3D **P**atterns with the constraints of minimum support and minimum confidence, MSP was proposed as an improvement of sCluster. Each pattern is a group of similar sequential 3D objects appearing in a given dataset. MSP involves three stages: generating seeds with pairwise pattern mining, vertical extension to detect all the hits with the constraints using a depth-first search and horizontal extension to extend the pattern length without loss of hits. Furthermore, we proposed a method to automatically **D**etect **P**roper **S**ettings, DPS, in order to adapt MSP to various datasets. DPS is a dual-level binary search algorithm with respects to seed length and error tolerance. DPS calls the MSP for mining patterns and it stops when the patterns are significantly more than expected. Binary search was adopted to find good settings within a pre-defined ranges of seed length and error tolerance. The experiments on protein chains and synthetic data showed MSP significantly outperforms sCluster. We applied MSP to protein family classification, and the obtained patterns correctly classified the protein families on all the tested binary-class datasets. We also applied MSP to PhysioNet/CinC Challenge 2002 dataset and achieved a good accuracy in the classification event.

For the purpose of complementing the traditional protein classification approaches which mainly leveraged on amino acid sequences, we were promoted to build a classifier for remotely homologous proteins purely based on 3D structure patterns. In Chapter 6, we investigated the characteristics of proteins, applied sCluster and MSP for protein 3D pattern mining and built the binary classification rule groups. Furthermore, $Deterministic\ Binary\ Classification\ Tree$ was designed to incorporate binary classifiers to enable multi-class classification. Experiments on various protein families showed that the system discovered valuable motifs and both the precision and recall of our approach were high. Meanwhile, protein prediction time has been significantly reduced compared to the $One-VS-All$ method.

To deploy our research results into real-life applications, we have incorporated $ed-tree$ on protein sequences and $sCluster$s on protein 3D structures into a FCDR System. It interactively visualizes frequent 3D protein structures and enables researchers to find the DNA regions which code similar protein structures.

8.2 Future Works

To conduct indexing and query on large DNA sequence databases on personal PC, *ed-tree* can be deployed to be a complete usable application. On the other hand, current *ed-tree* is mainly to index the database, and the length of query sequence is ranging from tens to hundreds of nucleic acids. In case of long query sequence and large sequence database for example, yeast genome and human genome, *ed-tree* can be used to index both query and database. A query algorithm deserves to be developed to compare two *ed*-trees.

sCluster and MSP are generic subspace cluster approaches to sequential 3D objects. It provides a framework on 3D structural pattern mining. One direction is to apply this algorithm into more real-life applications such as web applications, moving objects and so on. In each application, different features should be selected for calculating fds according to the characteristics of datasets. The other direction is investigate 3D structural patterns in all kinds of 3D objects rather than only sequential 3D objects. This topic is more meaningful and challenging.

To bring sCluster and MSP to real-life applications, it is possible and valuable to integrate them with a web-based interface for researchers to conduct query and mining on protein chains. A few interesting applications can be supported. One is to mine 3D structure patterns appearing from different protein families. It can discover the homology between families, superfamilies and so on. The other is to detect remotely homologous proteins where the sequence similarity is ambiguous but the structural similarity and functional similarity are high. The obtained patterns can also be presented in a rotating 3D manner by integrating the results with various visualization methods such as $PDB2multiGIF$ [47] and Chime [61].

BIBLIOGRAPHY

[1] C.C. Aggarwal, J.L. Wolf, P.S. Yu, C. Procopiuc, and J.S. Park. Fast algorithms for projected clustering. In *SIGMOD*, pages 61–72, 1999.

[2] C.C. Aggarwal and P.S. Yu. Finding generalized projected clusters in high dimensional spaces. In *SIGMOD*, pages 70–81, 2000.

[3] S.A. Aghili, D. Agrawal, and A.E. Abbadi. Pads: Protein structure alignment using directional shape signatures. In *DASFFA*, pages 17–29, 2005.

[4] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *SIGMOD*, pages 94–105, 1998.

[5] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A.I. Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI/MIT Press, 1996.

[6] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *VLDB*, pages 487–499, 1994.

[7] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman. Basic local alignment search tool. *Molecular Biology*, 215:403–410, 1990.

[8] S.F. Altschul, T.L. Madden, A.A. Schafer1, J. Zhang, Z. Zhang, W. Miller, and D.J. Lipman. Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic Acids Research*, 25:3389–3402, 1997.

[9] A. Andreeva, D. Howorth, S.E. Brenner, T.J.P. Hubbard, C. Chothia, and A.G. Murzin. Scop database in 2004: refinements integrate structure and sequence family data. *Nucleic Acids Research*, 32:D226–D229, 2004.

[10] R. Baeza-Yates and G. Navarro. Fast approximate string matching. *Algorithmica*, 23:127–158, 1999.

[11] D.A. Benson, M.S. Boguski, D.J. Lipman, J. Ostell, and B.F. Ouellette. Genbank. *Nucleic Acids Research*, 26:1–7, 1998.

[12] D.A. Benson, I. Karsch-mizrachi, D.J. Lipman, J. Ostell, B.A. Rapp, and D.L. Wheeler. Genbank. *Nucleic Acids Research*, 28:15–18, 2000.

[13] P.J. Besl and N.D. Mckay. A method for registration of 3-d shapes. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 14:239–256, 1998.

[14] C. Branden and J. Tooze. *Introduction to Protein Structure.* Garland Publishing inc, New York and London, 2001.

[15] S.E. Brenner, C. CHOTHIA, J.P. Hubbard, and A.G. Murzin. Understanding protein structure: Using scop for fold interpretation. *Methods in Enzymology*, 266:635–643, 1996.

[16] J. Buhler. Efficient large-scale sequence comparison by locality-sensitive hashing. *Bioinformatics*, 17:419–428, 2001.

[17] S. Burkhardt, A. Crauser, P. Ferragina, H.P. Lenhof, and M. Vingron. q-gram based database searching using a suffix array (quasar). In *RECOMB*, Lyon, April 1999.

[18] S. Busuttil, J. Abela, and G. Pace. Support vector machines with profile-based kernels for remote protein homology detection. In *GIW*, 2004.

[19] J. Chang and D. Jin. A new cell-based clustering method for large, high-dimensional data in data mining applications. In *SAC*, pages 503–507, 2002.

[20] C. Cheng, W.C. Fu, and Y. Zhang. Enclus: Entropy-based subspace clustering for mining numerical data. In *KDD*, 1999.

[21] S. Cheong and S.H. Oh. Neural information processing. *Journal of Atmospheric and Oceanic Technology*, 2, 2004.

[22] L.P. Chew, D.P. Huttenlocher, K. Kedem, and J.M. Kleinberg. Fast detection of common geometric substructure in proteins. In *RECOMB*, pages 104–114.

[23] G. Cong. *Discover, Recycle And Reuse Frequent Patterns In Association Rule Mining*. PhD thesis, School of Computing, National University of Singapore, 2004.

[24] G. Cong, A.K.H. Tung, X. Xu, F. Pan, and J. Yang. Farmer: Finding interesting rule groups in microarray datasets. In *SIGMOD*, pages 143–154, 2004.

[25] L. Conte, S.E. Brenner, T.J.P. Hubbard, C. Chothia, and A. Murzin. Scop database in 2002: refinements accommodate structural genomics. *Nucleic Acids Research*, 30(1):264–267, 2002.

[26] B. Cui. *Indexing for Efficient Main Memory Processing.* PhD thesis, School of Computing, National University of Singapore, 2003.

[27] L. Dehaspe, H. Toivonen, and R.D. King. Finding frequent substructures in chemical compounds. In *KDD*, pages 30–36, 1998.

[28] I. Eidhammer, I. Jonassen, and W.R.Taylor. Structure comparison and structure patterns. *Journal of Computational Biology*, 7:685–716, 2000.

[29] C. Faloutsos, K.S. McCurley, and A. Tomkins. Fast discovery of connection subgraphs. In *KDD*, pages 118–127, 2004.

[30] E. Giladi, M. Walker, J. Wang, and W. Volkmuth. Sst: An algorithm for searching sequence databases in time proportional to the logarithm of the database size. In *RECOMB*, 2000.

[31] R.S.C. Goble, P. Baker, and A. Brass. A classification of tasks in bioinformatics. *Bioinformatics*, 17:180–188, 2001.

[32] J. Han and M. Kamber. *Data Mining: Concepts and Techniques.* Morgan Kaufmann, 2000.

[33] J. Han and J. Pei. Mining frequent patterns by pattern-growth:methodology and implications. In *KDD Explorations 2(2)*, pages 14–20, 2000.

[34] J.A. Hartigan. *Clustering Algorithms.* John Wiley & Sons, New York, 1975.

[35] L. Holm and C. Sander. Mapping the protein universe. *Science*, 273:595–603, 1996.

[36] Y. Hou, W. Hsu, M. Lee, and C. Bystroff. Efficient remote homology detection using local structure. *Bioinformatics*, 19:2294–2301, 2003.

[37] Y. Hou, W. Hsu, M. Lee, and C. Bystroff. Remote homolog detection using local sequence structure correlations. *PROTEINS: Structure, Function, and Bioinformatics*, 57:518–530, 2004.

[38] J. Huan, W. Wang, D. Bandyopadhyay, J. Snoeyink, J. Prins, and A. Tropsha. Mining protein family specific residue packing patterns from protein structure graphs. In *Recomb*, pages 308–315, 2004.

[39] J. Huan, W. Wang, and J. Prins. Efficient mining of frequent subgraph in the presence of isomorphism. In *ICDM*, pages 549–552, 2003.

[40] J. Huan, W. Wang, J. Prins, and J. Yang. Spin: mining maximal frequent subgraphs from graph databases. In *KDD*, pages 581–586, 2004.

[41] E. Hunt, M.P. Atkinson, and R.W. Irving. A database index to large biological sequences. In *VLDB*, pages 139–148, 2001.

[42] A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *PAKDD*, pages 13–23, 2000.

[43] T. Jaakkola, M. Diekhans, and D. Haussler. A discriminative framework for detecting remote protein homologies. *Journal of Computational Biology*, 7:95–114, 2000.

[44] T. Kahveci, V. Ljosa, and A.K. Singh. Speed up whole-genome alignment by indexing frequency vectors. *Bioinformatics*, 20:2122–2123, 2004.

[45] T. Kahveci and A. Singh. An efficient index structure for string databases. In *VLDB*, 2001.

[46] K. Kailing, H. Kriegel, and P. Kroger. Density-connected subspace clustering for high-dimensional data. In *ICDM*, pages 246–257, 2004.

[47] W. Kiyne and V. Prelog. Description of steric relationships across single bonds. *Journal of Molecular Modeling*, 4:344–346, 1998.

[48] G.J. Kleywegt. Recognition of spatial motifs in protein structures. *Journal of Molecular Biology*, 285:1887–1897, 1999.

[49] M. Kuramochi and G. Karypis. Discovering frequent geometric subgraphs. In *ICDM*, pages 258–265, 2002.

[50] M. Kuramochi and G. Karypis. An efficient algorithm for discovering frequent subgraphs. *IEEE Transaction of Knowledge Data Engineering*, 16:1038–1051, 2004.

[51] R. Lees. Cell biology tutorial. *http://www.biology-online.org/tutorials/1_cell_biology.htm*, 2001.

[52] C. Leslie, E. Eskin, J. Weston, and W. Noble. Mismatch string kernels for svm protein classification. *Neural Information Processing Systems*, 15:1441–1448, 2003.

[53] H. Li, H. Marsolo, S. Parthasarathy, and D. Polshakov. A new approach to protein structure mining and alignment. In *4th Workshop on Data Mining in Bioinformatics*, pages 1–10, 2004.

[54] L. Liao and W.S. Noble. Combining pairwise sequence similarity and support vector machines for remote protein homology detection. In *RECOMB*, pages 225–232, 2002.

[55] B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In *KDD*, pages 80–86, New York, 1998.

[56] B. Liu, Y. Xia, and P.S. Yu. Clustering through decision tree construction. In *CIKM*, pages 20–29, 2000.

[57] B.T. Logan, U. Karaoz, P.J. Moreno, Z. Weng, and S. Kasif. Protein seer: A web server for protein homology detection. Technical Report HPL-2004-131, HP Labs, 2004.

[58] G. Lu. Top: A new method for protein structure comparisons and similarity searches. *Journal of Applied Crystallography*, 33:176–183, 2000.

[59] B. Ma, J. Tromp, and M. Li. Patternhunter: faster and more sensitive homology search. *Bioinformatics*, 18:440–445, 2002.

[60] U. Manber and G. Myers. Suffix arrays: a new method for on-line string search. *SIAM Journal on Computing*, 22:935–948, 1993.

[61] Elsevier MDL. Mdl chime. *http://www.mdli.com/chemscape/chime.*

[62] C. Meek, J.M. Patel, and S. Kasetty. Oasis: An online and accurate technique for local-alignment searches on biological sequences. In *VLDB*, pages 910–921, 2003.

[63] 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.

[64] R. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. In *VLDB*, pages 144–155, 1994.

[65] National Institutes of Health and National Institute of General Medical Sciences. The structures of life. *http://publications.nigms.nih.gov/structlife/structlife.pdf*, 2000.

[66] C.A. Orengo, A.D. Michie, S. Jones, D.T. Jones, M.B. Swindells, and J.M. Thornton. Cath-a hierarchic classification of protein domain structures. *Structure*, 5:1093–1108, 1997.

[67] L. Parsons, E. Haque, and H. Liu. Evaluating subspace clustering algorithms. In *Workshop on Clustering High Dimensional Data and its Applications, SDM*, pages 48–56, 2004.

[68] F. Pearl, D. Lee, J.E. Bray, I. Sillitoe, A.E. Todd, A.P. Harrison, J.M. Thornton, and C.A. Orengo. Assigning genomic sequences to cath nucleic acids research. *Nucleic Acids Research*, 28:277–282, 2000.

[69] F. Pearl, A. Todd, I. Sillitoe, M. Dibley, O. Redfern, T. Lewis, C. Bennett, R. Marsden, A. Grant, D. Lee, A. Akpor, M. Maibaum, A. Harrison, T. Dallman, G. Reeves, I. Diboun, S. Addou, S. Lise, C. Johnston, A. Sillero, J. Thornton, and C. Orengo. The cath domain structure database and related resources gene3d and dhs provide comprehensive domain family information for genome analysis. nucleic acids research. *Nucleic Acids Research*, 33:D247–D251, 2005.

[70] W.R. Pearson. Rapid and sensitive sequence comparisons with fastp and fasta. *Methods in Enzymology*, 183:63–98, 1995.

[71] W.R. Pearson, T. Wood, Z. Zhang, and W. Miller. Comparison of dna sequences with protein sequences. *GENOMICS*, 46:24–36, 1997.

[72] P.A. Pevzner. *Computational Molecular Biology.* The MIT press, 2000.

[73] PSRAST. What is genetic engineering? a simple introduction. *http://www.psrast.org/whatisge.htm.*

[74] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[75] K. Sadakane and T. Shibuya. Indexing huge genome sequences for solving various problems. *Genome Informatics*, pages 175–183, 2001.

[76] F. Schwenker. Hierarchical support vector machines for multi-class pattern recognition. In *KES*, 2000.

[77] T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *Molecular Biology*, 147:195–197, 1981.

[78] S. Srinivasa and S. Kumar. A platform based on the multi-dimensional data model for analysis of bio-molecular structure. In *VLDB*, 2003.

[79] D.J. States and P. Agarwal. Compact encoding strategies for dna sequence similarity search. In *ISMB*, pages 211–217, 1996.

[80] R. Szklarczyk and H. Heringa. Tracking repeats using significance and transitivity. In *ISMB*, pages 311–317, 2004.

[81] A.C. Tan, D. Gilbert, and Y. Deville. Integrative machine learning approach for multi-class scop protein fold classification. In *GCB*, 2003.

[82] Z. Tan, X. Cao, B.C. Ooi, and A.K.H. Tung. The ed-tree: An index for large dna sequence databases. In *SSDBM*, pages 151–160, 2003.

[83] Z. Tan and A.K.H. Tung. Clustering substructures from sequential 3d object data sets. In *ICDE*, pages 634–645, 2004.

[84] Z. Tan and A.K.H. Tung. Finding association rule groups on 3d structures for protein classification. Technical report, School of Computing, National University of Singapore, 2005.

[85] Z. Tan and A.K.H. Tung. scluster: A substructure clustering approach for sequential 3d object dataset. Technical report, School of Computing, National University of Singapore, 2005.

[86] Z. Tan and A.K.H. Tung. Mining frequent 3d sequential patterns. In *SSDBM*, 2006.

[87] E. Ukkonen. Approximate string matching with q-grams and maximal matches. *Theory of Computer Science*, 92:191–212, 1992.

[88] H. Wang, W. Wang, J. Yang, and P. S. Yu. Clustering by pattern similarity in large data set. In *SIGMOD*, pages 394–405, 2002.

[89] S. Wang, C. Chen, and M. Hwang. Classification of protein 3d folds by hidden markov learning on sequences of structural alphabets. In *RECOMB*, pages 65–72, 2005.

[90] W. Wang, C. Wang, Y. Zhu, B. Shi, J. Pei, X. Yan, and J. Han. Graphminer: a structural pattern-mining system for large disk-based graph databases and its applications. In *SIGMOD*, pages 879–881, 2005.

[91] H.E. Williams and J. Zobel. Indexing and retrieval for genomic databases. *IEEE Transactions on Knowledge and Data Engineering*, 14:63–78, 2002.

[92] X. Yan and J. Han. gspan: Graph-based substructure pattern mining. Technical Report R-2002-2296, UIUC-CS, 2002.

[93] X. Yan and J. Han. Closegraph: Mining closed frequent graph patterns. In *KDD*, pages 286–295, 2003.

[94] X. Yan, X. Zhou, and J. Han. Mining closed relational graphs with connectivity constraints. In *ICDE*, pages 357–358, 2005.

[95] J. Yang, W. Wang, H. Wang, and P.S. Yu. delta-cluster: Capturing subspace correlation in a large data set. In *ICDE*, 2002.

[96] J. Yang, W. Wang, P.S. Yu, and J. Han. Mining long sequential patterns in a noisy environment. In *SIGMOD*, pages 406–417, 2002.

[97] C.S. Yu, J.Y. Wang, P.C. Lyu, C.J. Lin, and J.K. Huwang. Fine-grained protein fold assignment by support vector machines using generalized npeptide coding schemes and jury voting from multiple-parameter sets. *Proteins*, 19:531–536, 2003.

[98] M. Zaki and C. Hsiao. Charm: An efficient algorithm for closed association rule mining. In *SDM*, 2002.

[99] Z. Zhang, W.R. Pearson, and W. Miller. Aligning a dna sequence with a protein sequence. In *RECOMB*, 1997.