Approximate Matching in Genomic Sequence Data

Xia Cao

NATIONAL UNIVERSITY OF SINGAPORE
2006

Approximate Matching in Genomic Sequence Data

Xia Cao

Master of Computer Engineering, Wuhan University, China

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

SCHOOL OF COMPUTING

NATIONAL UNIVERSITY OF SINGAPORE

2006

Acknowledgement

This thesis is the result of a collaboration with a very talented group of people. I consider myself extremely fortunate to have received such excellent training and education as well as tremendous support and encouragement at the National University of Singapore.

First, I would like to express my appreciation to my supervisors Prof. Ooi Beng Chin and Dr. Tung Kum Hoe for their invaluable tutoring, advice, perspective, and encouragement through all the years of my Ph.D study. I have learned a lot from them about how to do and present research work. This work could not have been completed without their insight and encouragement.

I am thankful to the members of my thesis evaluation committees for going through my thesis and giving me valuable feedback. They are Prof. Tan Kian-Lee and Dr. Ken Sung.

I also wish to thank Prof. Tan Kian-Lee for his valuable suggestions and help. A big part of the great and enjoyable experience here at the School of Computing came from working in the Database Group and the Computational Biology Group. I am deeply indebted to Li Shuaicheng and Tan Zhenqiang for their very helpful

ideas and discussions. I would like to thank Zhang Zong Hong, Yang Xia, Yang Jing, Cong Gao, Zhang Zhenjie, Dai Bingtian, Lin Dan, Li Hanyu, Cui Bin, He Qi, Li Yingguang, Guo Shuqiao, Zhang Rui and Yang Rui for their friendship and support.

I could not have achieved this degree without the support and encouragement of my family. Many thanks go to my parents and sisters, who have always encouraged me to pursue my education and provided often a helping hand. Finally, I wish to thank my husband Xuewen Chen for his love, support and understanding while this thesis was being written.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Summary

Increasing interest in genetic research has resulted in the creation of huge genomic databases and approximate sequence matching in genomic sequence databases has become a basic operation in computational biology. In this thesis, we shall design several models and algorithms for approximate sequence matching in the context of DNA sequence similarity search, DNA sequence similarity join, and protein sequence subcellular localization prediction.

To efficiently support similarity search in very large DNA sequence databases, we present an efficient hash-based model for DNA sequences. In this model, only certain segments of a DNA sequence database called "piers" need to be accessed during search, unlike other approaches, where a full scan of the biological sequence database is required. To further improve search efficiency, the piers are stored in a specially designed hash table, which helps avoid expensive alignment operations. The hash table is small enough to reside in main memory, hence avoiding I/Os in the search steps. We investigate the effect of parameter settings on the performance of the proposed hash-based pier model. We also compare the proposed approach with the latest version of BLAST11, and show theoretically and empirically that

our approach can efficiently detect biological sequences that are similar to a query sequence with acceptable accuracy. Moreover, the idea of "pier" can be used on any kind of sequence indexing structures as a means to select interesting segments for indexing.

To facilitate similarity search in a DNA database and sidestep the need for linear scan of the entire database, we propose a novel two-level index method for indexing long seeds efficiently based on q-grams of DNA sequences. At the first level, a hash table is built on the sequences in terms of qClusters, which are a group of clusters generated on the q-grams. At the second level, a novel data structure called c-trees is proposed to organize c-signatures for sequence similarity search. The c-signatures of a DNA sequence are generated according to the occurrence of q-grams in the sequence. The proposed data structures allow the quick detection of sequences within a certain distance to the query sequence. We present the results of experiments conducted to evaluate the performance of the proposed two-level index against hash-based pier model and the latest version of BLASTn.

To perform DNA sequence approximate join efficiently without false dismissal, we propose a filter-and-refine sequence join algorithm for DNA sequences. While the filtering phase can rapidly prune away sequences that are not joinable, the refinement phase employs a comprehensive algorithm to remove the remaining false alarms. The efficiency of the proposed scheme lies in the use of the precedence count matrix (PCM) for approximating the edit distance between two sequences. With PCM, the time complexity of sequence comparison is bounded by a constant. We have evaluated the proposed sequence join algorithm based on PCM, and our study shows that it outperforms the known techniques.

To effectively predict the subcellular localization of proteins, q-gram frequency vectors, q-gram wavelet vectors, q-gram similarity vectors, and q-gram TF.IDF

vectors based on q-grams for protein sequences are proposed, and the Support Vector Machine (SVM) is used to predict the subcellular localization of proteins based on these q-gram vectors in the sequences. The experimental results show that the q-gram based features represent the protein sequence well, and they are very effective for the prediction of the subcellular localization of proteins. Since there is no single method of prediction which can achieve high prediction accuracy, precision or recall for all the subcellular localizations for proteins, the contribution of our proposed prediction method is substantial and useful in practice.

We believe that our contributions have successfully addressed some of the issues of approximate sequence matching in genomic sequences. Our contributions include the proposal of an efficient search model for DNA sequences [27], a novel indexing structure of DNA sequences [28], a filter-and-refine algorithm for DNA sequence approximate join [30] and some q-gram based subcellular localization prediction methods for proteins [29]. We have conducted extensive performance studies, and the experimental results show that the proposed methods are effective and efficient for the problems addressed in this thesis.

The publications that have arisen from the material described in this thesis are listed in the reverse chronological order as follows.

- Xia Cao, Beng Chin Ooi, Kian-Lee Tan, Anthony K.H. Tung. *The q-gram Based Protein Subcellular Localization Prediction.* Technical Report: School of Computing, National University of Singapore, 2005.

- Xia Cao, Shuai Cheng Li, Anthony K.H. Tung. *Indexing DNA Sequences Using q-grams.* In Proc. of the 10th Int. Conf. on Database Systems for Advanced Applications, 2005.

- Xia Cao, Shuai Cheng Li, Beng Chin Ooi, Anthony K.H. Tung. *Piers: An*

Efficient Model for Similarity Search in DNA Sequence Databases. In ACM Sigmod Record, 33(2):39-44, 2004.

- Xia Cao, Anthony K.H. Tung, Beng Chin Ooi, Kian-Lee Tan, Shuai Cheng Li. *String Join Using Precedence Count Matrix.* In Proc. of the 16th Int. Conf. on Scientific and Statistical Database Management, 2004.

CHAPTER 1

Introduction

Sequence data naturally arises in many real-world applications such as genomic data, web data and event sequences. There is frequent need to conduct sequence similarity search, sequence approximate join and sequence mining to locate some useful information in a sequence database. These applications in sequence data involve sequence approximate matching. In contrast to the simpler exact matching problem, which consists of locating all exact matches between a query or pattern and a target database, sequence approximate matching includes recognizing all approximate matches with respect to a certain measure of similarity or distance. Furthermore, the sequence approximate matching problem can be classified into two groups: full sequence approximate matching and subsequence approximate matching. In this thesis, we confine our attention to sequence approximate matching in the aspect of subsequence matching since the subsequence approximate matching problem is a general case of the full sequence approximate matching problem. This, however, does not mean that we forget the role of exact matching; rather,

we consider exact matching problems to be subproblems in large-scale sequence comparison, database search and other biologically important applications. For the approximate sequence matching problem, there is a need to measure the difference or distance between two sequences in the study of biological sequences. One common and simple formalization, called edit distance, focuses on transforming (or editing) one sequence to the other by a series of edit operations on individual characters [50].

This thesis presents our research in three important problems in the area of approximate subsequence matching: DNA sequence similarity search in a sequence database, DNA sequence approximate join, and protein subcellular localization prediction.

1.1 Background of Genomic Sequence Approximate Matching

There exist a number of practical applications for approximate sequence matching including signal processing, text retrieval, optical character recognition and pattern recognition. "Approximate" means some errors of various types are acceptable in valid matches. The particularly important recent application for approximate sequence matching is genome research. The growing interest in genome research has resulted in the creation of huge genomic databases and significant breakthroughs have already been achieved with the aid of the analysis of approximate matching in genomic databases. Databases holding genomic sequences are firmly established as central tools in current molecular biology, and electronic databases are becoming the lifeline of the field [50]. In the following, we survey the background knowledge to genomic databases and introduce the three problems investigated in this thesis:

similarity search in DNA sequence database, DNA sequence approximate join, and protein sequence subcellular localization prediction which are all related to sequence approximate matching in genomic databases.

1.1.1 Genomics and Genomic Databases

Genetic material, or DNA is the basic blueprint of life, and its structure can be viewed as a simple but very long sequence over the four-letter alphabet of A, C, G and T. Some nucleotide sequences are responsible for the production of the protein. Such DNA sequences are transcribed to RNA, which is a one-strand sequence similar in structure to DNA. Triplet combinations of the nucleotide bases from the mRNA, known as codons, are used to specify amino acids. Since there are four kinds of bases in DNA sequence, there are 64 possible nucleotide triplets. However, there are only 20 amino acids to specify since different triplet can correspond to the same amino acid. A protein sequence is a chain of amino acids.

A genomic database is a database of genetic sequences. Genomic databases assist molecular biologists in understanding the biochemical function, chemical structure and evolutionary history of an organism [121]. Due in part to the development of molecular biology, large numbers of DNA, RNA and protein sequences have been determined in the past two decades. In recent years, statistics show that the size of the collective genomic database doubles every 15 months [17].

There are several public DNA sequence databases. DNA sequence databases were first assembled at the Los Alamos National Laboratory (LANL) in New Mexico by Walter Goad and his colleagues who worked on the GenBank database, and at the European Molecular Biology Laboratory (EMBL) in Heidelberg, Germany, where the EMBL database was assembled [77]. GenBank is now maintained by the National Center for Biotechnology Information (NCBI). Currently,

the large and well-known DNA sequence databases include GenBank, EMBL and the DNA Database of Japan (DDBJ) [101, 77]. GenBank, EMBL, and DDBJ have now formed the International Nucleotide Sequence Database Collaboration (http://www.ncbi.nlm.nih.gov/collab). The three databases are similar in structure, and are updated every day to guarantee their data consistency.

Though DNA is the basic blueprint of life, protein sequences are the first sequences to be collected into a database instead of DNA sequences. Margaret Dayhoff and her colleagues were the pioneers to assemble the databases of these protein sequences, and the collection eventually becomes known as the Protein Information Resource (PIR) [3]. The SWISS-PROT Protein Knowledgebase [90, 19] is an annotated protein sequence database established in 1986 and maintained collaboratively by the Swiss Institute for Bioinformatics (SIB) and the European Bioinformatics Institute (EBI).

In the experiments reported in this thesis, we use the biological sequence database in GenBank for DNA sequence processing, and the protein sequences from SWISS-PROT for protein sequence subcellular localization prediction.

1.1.2 Similarity Search in Genomic Sequence Database

In biological sequences (DNA, RNA, or protein sequences), high sequence similarity usually implies significant functional or structural similarity. Understanding the relationship of a query DNA or protein genomic sequence to the known sequences in genomic databases allows molecular biologists to assign functions to poorly understood sequences. Therefore, similarity search in genomic databases is an important function in genome research as it is useful for discovering the location of functional sites, searching novel repeats and conducting comparative analysis of different genomic sequences. To cater for evolutionary mutations in genomic sequences and

noise in the sequence data, approximate sequence matching is preferred to exact matching from the biologists' point of view when similarity search in genomic databases is conducted.

Many approaches have been developed for approximate sequence matching. The most fundamental is the Smith-Waterman alignment algorithm [108], which is a dynamic programming approach that seeks optimal alignment between a query and the target sequence in $O(mn)$ time, m and n being the length of the two sequences. These methods are not practical for long sequences in the megabases range due to the time complexity of $O(mn)$.

Effort spent improving the efficiency of approximate sequence matching results in the common idea of filtering by discarding regions of low sequence similarity. Many approaches have been proposed to perform approximate sequence matching with respect to the idea of filtering. A well known approach is to scan biological sequences and find short seed exact matches which are subsequently extended into longer alignments. This method detects similar regions without using dynamic programming, and is used in programs such as FASTA [96] and BLAST [8], which are the most popular tools among biologists. However, the dilemma of this approach is that increasing seed size decreases search sensitivity whereas decreasing seed size leads to too many random search results. An alternative approach is to build an index on the data sequences and conduct the search on the index. Various index structure models have been proposed for this purpose. In these index structures, the suffix tree and the suffix array are the popular data structures for sequence similarity search, as seen in algorithms such as QUASAR [25] and the disk-based suffix tree structure used in [57]. Suffix trees and suffix array provide efficient string operations but are not well suited to handling insertion and deletion (gap) in either sequence. Furthermore, the structure of the suffix tree and the suffix array devours

very large amounts of memory. For example, an index file of 2GB is built for a DNA sequence of the size 20.5M when a suffix tree with links is used. Even if the suffix tree is used without links as proposed in [57], the suffix tree structure index is still nearly 10 times the size of the original sequence database. There also exist some other index structures for biological sequence databases [47, 61, 88, 121, 110, 91]. Though these proposed index structures can support genomic sequence search efficiently, they suffer either a large index structure or low sensitivity for similarity search.

1.1.3 Genomic Sequence Approximate Join

The join operation is one of the most useful operations for relational databases and the most commonly used way to combine information from two or more relations based on common attributes [99]. Likewise, in the area of computational biology, join on sequences is very useful for combining sequences, but it is based on similar sequence values.

Sequence join, which is a computationally expensive operation on sequences, combines data from two sequence datasets with *similar* sequence values on the join attribute. The similarity (or distance) between two sequences is typically determined by the *edit distance*, which is computed by using the standard dynamic programming approach [50]. Two sequences are said to be joinable if the prefix of one sequence is similar to the suffix of another with respect to the edit distance. For every ordered pair of sequenced sequences S_1 and S_2, we would compute the longest suffix of S_1 that approximately matches a prefix of S_2. In the context of genomic applications, such as *sequencing by hybridization* or *sequence assembly*, a sequence is assembled from a set of smaller and overlapping subsequences. In sequence assembly, the first step is to find how much a suffix of the first sequence

matches a prefix of the second. Sequencing errors are a reality (even if they are only in the 1-5% range) and suffix-prefix matching must allow for approximate matches [50]. We design an algorithm to find the longest suffix-prefix match which allows for approximate match for every pair of sequences.

To find the longest suffix and prefix match within a certain distance between two sequences S_1 and S_2 with length m and n respectively, standard dynamic programming can be used [50], and the time complexity to compute the longest suffix-prefix matches is $O(mn)$. However the computation of the longest best suffix-prefix matches becomes a bottleneck in *sequencing by hybridization* or *sequence assembly* when the length of sequences is very large. Many heuristic approaches have been subsequently proposed to speed up the sequence join by skipping the dynamic programming computation for unattractive pairs. Chen and Skiena [31] proposed a method called in-depth examination of exact matching with false dismissals based on suffix trees and suffix arrays. Their test established that the approach achieves a 1,000 speedup over dynamic programming while sacrificing 1% quality in sequence join. The approach finds 99% of the significant overlaps found by using dynamic programming. A method that computes the length of the longest common subsequences was presented to speed up sequence join as well since two sequences which have sufficient overlap should have at least one significant long common subsequence [50]. The idea is that we can recognize and exclude many pairs of sequences which are unlikely to be overlapping pairs in the full sequence [50]. Cohen [34] presented a framework for approximate sequence matching using the vector space model of similarity. However, the similarity metric for sequence joins is TF.IDF term weighting [1], rather than edit distance. Since TF.IDF between

[1]The term frequency / inverse document frequency (TF.IDF) is commonly used to weight each word in a text document. The TF.IDF approach can capture the relevancy among words, text documents and particular categories.

two sequences does not correspond well with actual edit distance, a larger number of false dismissals may occur in genomic sequence join.

The q-grams, which have been well used in text retrieval, could be used to generate the candidates of approximate sequence joins. Gravano et al. [48] used the concept of q-grams in approximate sequence joins in relational databases by augmenting a database with q-grams information, which is needed to run approximate sequence join. However, the filter rate of this method is still not efficient enough for sequence join for genomic data. Jin et al. [59] proposed a two-step process for sequence join. Their approach can support any distance measure between sequences, but it suffers from a large number of false dismissals during the processing of sequence join.

1.1.4 Protein Subcellular Localization Prediction

Advances in proteomics and genome sequencing are generating an enormous amount of data on genes and proteins at an accelerating rate. Mining the DNA, RNA and protein data to extract significant information is essential in genome processing. The significant information may refer to motifs, functional sites, clustering and classification rules [118].

The development of automated systems for the annotations of protein structure and function has become extremely important. Subcellular localization is a key function characteristic of potential gene products such as proteins [39], and the specific knowledge of subcellular localization allow biologists to decide if further experimental studies of proteins are required [65]. Therefore, it is very important to use automated annotation systems to identify or predict subcellular localization of proteins.

We assume there are two protein sequence datasets: a positive dataset and a

negative dataset. For a localization L, positive sequences are the protein sequences that locate in localization L, and negative sequences are the protein sequence that do not locate in localization L. The problem of predicting protein subcellular localization can be stated as follows: Given an unlabeled protein sequence S, and a known subcellular localization L, we want to determine if the sequence S locates in the localization L. Several methods have been proposed during the last decade for the prediction or classification task of protein localization. Since 1991, a number of systems have been developed to support the automated prediction of subcellular localization of proteins using different approaches. In these systems, machine learning methods such as Artificial Neural Networks, the k-nearest neighbors method, and the Support Vector Machine (SVM) have been applied on different features extracted from protein sequences.

The existing methods may be grouped into three categories. The first category of methods use similarity search to assign functions including the subcellular localization site of a protein. Subcellular localization tends to be evolutionarily conserved, thus homology to a protein of known localization can be a good indicator of a protein's actual localization site [79]. However, this method fails when the query sequence and target protein sequence are not significantly similar. The second group of methods use sequence motifs such as peptide signals, or nuclear localization signals, which are short subsequences with a length of three to 70 amino acids [40]. The problem of this method is that sometimes it is very difficult to find universal motifs for a group of protein sequences. The third group of methods are based on amino acid composition, where some machine learning classifiers are used to implement the prediction. The biological experiments show that the information needed to direct a protein to any localization site is mainly encoded in its amino acid sequence. For example, NNPSL [100] uses artificial neural nets (ANN), and SubLoc

[55] uses SVM as classifier based on amino acid composition. This approach may not capture the information on sequence order and the inter-relationships between amino acids.

The previous research on protein subcellular localization prediction clearly indicates that no single method of prediction can achieve high prediction accuracy, precision or recall for all subcellular localizations of proteins. The observation indeed provides us with the motivation to propose novel approaches to predict the subcellular localization of proteins.

1.2 Motivation and Objectives

Sequence similarity search, sequence approximate join, and sequence mining are important applications of sequence processing in molecular biology. While they may differ in functionalities, they share certain underlying operations, and they are common underlying operations, such as sequence approximate matching and sequence alignment, that determine their efficiency and effectiveness. To process approximate matching, the approximation metric must be specified, and there are several ways to formalize the notion of distance between sequences. One common and simple formalization called *edit distance* focuses on editing one sequence into the other by a series of edit operations on individual characters. Though edit distance is one common and simple approximation metric for sequence approximate matching, the time complexity and space complexity of computing edit distance are both $O(mn)$ for two sequences with length m and n respectively when using standard dynamic programming [108]. Obviously, the computation of edit distance is very costly in terms of both time and space when sequences in the database are very long.

To speed up approximate sequence matching, filtering is an efficient means to quickly discard irrelevant parts of a sequence database by means of filtering criteria. Useful parts are retained for further checking with the edit distance computed using dynamic programming. Several filtering techniques have been developed for efficient sequence approximate matching of DNA sequences, and they require reasonable amount of memory and disk space.

In this thesis, we set out to achieve three goals:

1. First, we seek to develop efficient index structures and design the corresponding algorithms for efficient comparison of many short DNA query sequences with a very large genomic database. To measure the new proposed structures and algorithms, we have devised the following criteria that the similarity search method should meet.

    - The index data structure should be a compact and approximate representation of a large genomic sequence database, and the size of the index structure is within an acceptable range compared to the original sequence database.
    - The filtering approach based on the index structure must be very efficient for sequence similarity search. It must also ensure there will be no false dismissals in sequence approximate matching. *False dismissals* are subsequences that are within a specified distance from query subsequences but are discarded wrongly as dissimilar subsequences. Sensitivity analysis for the search method must be conducted to guarantee that the search method is comparable in accuracy to existing popular systems in identifying answers.
    - The system must be fast and scalable with query rate and database size.

2. Second, we seek to design an approximate measurement of edit distance with the aim of decreasing the computational cost of deriving edit distance by standard dynamic programming. To this end, a DNA sequence is first transformed to a numeric vector which can be denoted as a point in high-dimensional space, and an algorithm is then developed for approximating the edit distance of two sequences in the new transformed data space. The edit distance approximation algorithm must satisfy the following principles:

    - The space of the transformed data vector should be small as we need to reduce the space requirement for approximating the edit distance between two sequences.
    - The distance function between two vectors defined in the new transformed spaces should be the lower bound of the actual edit distance between the two corresponding sequences. This principle is meant to be a guarantee against false dismissal in sequence approximate matching.
    - The approximation of edit distance should be sufficiently tight so that the number of false positives is small and the cost of refining results for final outputs is kept low.

3. Third, we seek to extract useful and significant information from protein subcellular location sequences. These extracted features should be "relevant" [118] in the sense that there should be high mutual information between the features and the classification label, which is the subcellular localization in this case. Moveover, for protein sequences, the extracted features should capture both the global and local similarity of the sequences. In all, the proposed feature extraction method should be very effective in capturing information in protein sequences that is useful and critical for sequence prediction, for

example, protein subcellular localization prediction.

1.3 Contribution

To achieve the objectives outlined in Section 1.2, we define each problem and study its related work, and subsequently propose novel sequence filtering techniques and sequence feature extraction methods for more efficient and effective sequence approximate matching in genomic databases. To study the effectiveness and efficiency of our proposals, we provide theoretical analysis and conduct extensive experiments using real datasets, comparing our methods against existing methods. We now summarize the contributions of this thesis:

First, we propose an efficient similarity search model for DNA sequences. From observation, we note that only some extracted DNA segments called "piers", need to be accessed from the DNA sequence database; there is no need to search the entire database. Based on the model, we construct a hash table on the extracted piers to further improve search efficiency and avoid unnecessary dynamic programming computation. The piers model is a general model for reducing the segments to be indexed by the indexing structures while keeping higher sensitivity.

Second, we propose a two-level index to organize DNA sequences efficiently based on q-grams. The purpose of the index is to allow similarity search in a DNA database, sidestepping the need for linear scan of the entire database. The two-level index structure is composed of two parts: a hash table built on the q-Clusters of DNA segments, and a novel data structure, c-trees, constructed on the q-grams of the DNA segments. The filter principle of the two-level index structure should guarantee efficient sequence search while keeping sensitivity high.

Third, we design an effective and efficient filter-and-refine sequence join algo-

rithm to conduct DNA sequence approximate join efficiently. The proposed scheme employs the precedence count matrix (PCM) to compute the edit distance between two DNA sequences efficiently.

Finally, to predict protein subcellular localization, we propose q-gram frequency vectors, q-gram wavelet vectors, q-gram similarity vectors, and q-gram TF.IDF vectors based on q-grams for protein sequences to extract useful information from a protein sequence. The sequence representation feature vectors can be trained on SVMs to predict the subcellular localization of proteins.

1.4 Thesis Organization

The thesis is organized as follows.

- Chapter 2 provides an introduction and overview of state-of-the-art research works that are closely related to this thesis. First, the backgrounds of molecular biology, genomic databases, and techniques for practical sequence comparison are introduced and described. Second, the core research problems of this thesis are defined, and related work are reviewed and discussed. They provide the necessary background for this thesis.

- In Chapter 3, an efficient hash-based pier model is presented for similarity search in very large DNA sequence databases. In this model, only certain segments in a DNA sequence database called "piers" need to be accessed during search, as opposed to other approaches which require a full scan of the biological sequence database. We compare our proposed approach with the latest of BLAST, and show theoretically and empirically that the proposed approach can efficiently detect biological sequences that are similar to a query sequence with very high sensitivity. The idea of "pier" is also applicable to

any kind of sequence indexing structures since it acts as a tool for selecting "useful" segments of a database for indexing.

- In Chapter 4, a novel method for indexing DNA sequences efficiently based on q-grams is proposed to facilitate similarity search in a DNA database and avoid the need for linear scan of the entire database. A two-level index is proposed based on the q-grams of DNA sequences. The proposed data structures allow the quick detection of sequences within a certain distance to the query sequence. We present experimental studies that evaluate the performance of the proposed two-level index against the proposed hash-based pier model and the latest version of BLASTn.

- In Chapter 5, we propose a filter-and-refine sequence join algorithm. While the filtering phase can rapidly prune away sequences that are not joinable, the refinement phase employs an efficient algorithm to remove the remaining false positives. The efficiency of the proposed scheme lies in the use of the PCM for computing the edit distance between two sequences. We also evaluate the proposed sequence join algorithm, and our performance study shows that it outperforms known techniques such as the q-grams method [48] and the frequency vector method [61].

- In Chapter 6, we devise several sequence features generated based on the q-grams for protein sequences: the q-gram frequency feature, the q-gram wavelet feature, the q-gram similarity feature, and the q-gram TF.IDF feature. SVM is used to predict the subcellular localization of proteins based on these proposed q-gram based features generated from sequences. The experimental studies show that q-gram based features can represent a protein sequence well, and they are very effective for the prediction of subcellular localization

of proteins.

- We conclude in Chapter 7 with a summary of our contributions, and discussion on some limitations of our work and some suggestions for future work.

CHAPTER 2

Background and Related Work

This chapter first gives an overview of concepts in molecular biology that are essential to computational biology. It then introduces the background of genomic sequence databases and addresses the importance of sequence comparison in molecular biology. Subsequently, the standard dynamic programming algorithm for computing the edit distance between two sequences is introduced. Finally, we present three research problems studied in this thesis for approximate genomic sequence matching in the area of molecular biology, and review the existing work related to these research problems.

2.1 Basic Concepts of Molecular Biology

Modern science has shown that life started some 3.5 billion years ago, shortly after the Earth itself was formed [33, 36]. Both complex and simple organisms are similar in molecular chemistry or bio-chemistry. The main actors in the chemistry of life

are molecules called proteins and nucleic acid. In general, proteins are responsible for what a living being is and does in a physical sense. Nucleic acids, on the other hand, encode the information necessary to produce the proteins and are responsible for passing along this "recipe" to subsequence generation.

The "central dogma" of information flow in biology states that information flows from DNA to RNA to protein; since a protein's functionality is determined by its unique three dimensional structure, it follows that the one-dimensional sequence information in DNA determines the three-dimensional structure of the corresponding protein [33].

The central dogma states that once "information" has passed into a protein it cannot get out again. The transfer of information from nucleic acid to protein may be possible, but transfer from protein to protein, or from protein to nucleic acid is impossible. Information here means the precise determination of sequence, either of bases in the nucleic acid or of amino acid in the protein [36].

The following depicts information flow in biology:

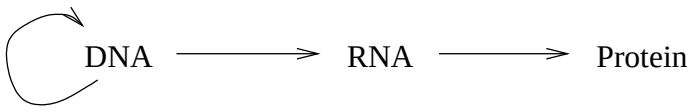

Figure 2.1: Information Flow

2.1.1 Genome and Chromosome

A genome is all the DNA contained in an organism or a cell, which includes the chromosomes plus the DNA in mitochondria (and DNA in the chloroplasts of plant cells)[1]. In other words, all the genetic information in an organism is referred to collectively as a "genome". A chromosome is one of the threadlike "packages" of

[1]definition from the National Human Genome Research Institute (NHGRI): Glossary of Genetic Terms.

genes and other DNA in the nucleus of a cell. Different kinds of organisms have different numbers of chromosomes. Humans have 23 pairs of chromosomes, 46 in all: 44 autosomes and two sex chromosomes. Each parent contributes one chromosome to each pair, so children get half of their chromosomes from their mothers and half from their fathers. An example of chromosome is given in Figure 2.2.

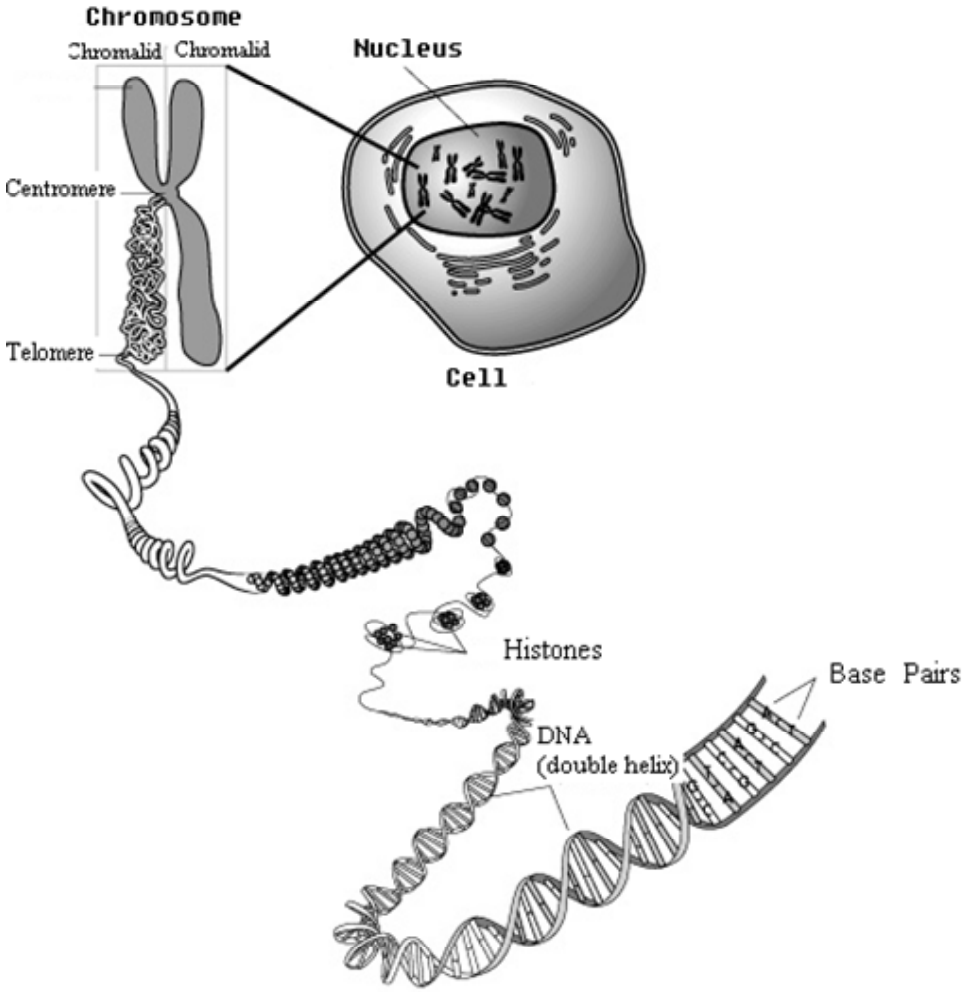

Figure 2.2: Chromosome (Image from[1])

2.1.2 Nucleotide, DNA and RNA

A nucleotide is one of the structural components, or building blocks, of DNA and RNA. A nucleotide consists of a base (adenine, thymine, guanine, and cytosine) plus a molecule of sugar and one of phosphoric acids [54].

Genetic material, or DNA is the basic blueprint of life, and its structure can be viewed as a simple but very long sequence. Both DNA and RNA are polymers, which are composed of nucleotides. DNA is composed by four bases adenine(A), cytosine(C), guanine(G), and thymine(T). DNA exists as a double-strand molecular, formed by hydrogen bonds between hydrogen bonds between complementary bases: A with T, and C with G, the so-called $Watson\text{-}Crick$ rules. Double-strand DNA forms a helix – two strands line up anti-parallel to each other but are oriented in opposite directions. DNA stores the instruction required by a cell to perform the daily life function. The information in DNA is used like a library. Then information in genes is read, maybe millions of times in the life of an organism, but the DNA itself is never used up.

In contrast to DNA, RNA is single-stranded. In RNA, the thymine is replaced by uracil (U). While DNA serves only the function of information storage, RNA serves certain catalytic functions through its complex three-dimensional form.

2.1.3 Genes

Genes, in the form of DNA, are embedded in a cell's chromosomes. A gene is the functional and physical unit of heredity passed from parent to offspring. Genes are pieces of DNA, and most genes contain information for making a specific protein or an RNA. Genes comprise two non-coding regions, whose functions may include providing chromosomal structural integrity and regulating where, when and in what quantity proteins are made.

2.1.4 Proteins

Proteins perform a wide variety of activities in a cell. A protein is a large complex molecule made up of one or more chains of amino acids. Amino acids are a group of 20 different kinds of small molecules that link together in long chains to form proteins. Therefore, amino acids are often referred to as the "building blocks" of proteins. The types of amino acids found in proteins are shown in Table 2.1 [104, 33]. There are altogether 20 amino acids in proteins, although a few nonstandard amino acids might also be present in them [104].

Protein synthesis begins in the cell's nucleus when the gene encoding a protein is copied into RNA. RNA then functions to convert the nucleic acid sequence into the amino acid sequences of proteins. The process of transferring the gene's DNA into RNA is called transcription. Transcription helps magnify the amount of DNA by creating many copies of RNA that can act as the template for protein synthesis [33]. The RNA copy of the gene is called the messenger RNA (mRNA).

Translation is the actual synthesis of a protein under the direction of mRNA [104, 33]. During this process the nucleotide sequence of an mRNA is translated into the amino acid sequence of a protein. The nucleotide sequence of the mRNA is composed of four different nucleotides whereas a protein is built up from 20 amino acids. To allow the four nucleotides to specify 20 different amino acids, the nucleotide sequence is interpreted in codons, groups of three nucleotides. These codons have their corresponding anticodon in the transfer RNA (tRNA). Furthermore each anticodon is linked to one particular amino acid. Thus, each codon specifies one amino acid.

A protein is not only a linear sequence of amino acids. The sequence is known as primary structure, and proteins also fold in three dimensions, which present secondary structure, tertiary structure and quaternary structure. In our work, as

	One-letter code	Three-letter code	Name
1	A	Ala	Alanine
2	C	Cys	Cysteine
3	D	Asp	Aspartic Acid
4	E	Glu	Glutamic Acid
5	F	Phe	Phenylalanine
6	G	Gly	Glycine
7	H	His	Histidine
8	I	Ile	Isoleucine
9	K	Lys	Lysine
10	L	Leu	Leucine
11	M	Met	Methionine
12	N	Asn	Asparagine
13	P	Pro	Proline
14	Q	Gln	Glutamine
15	R	Arg	Arginine
16	S	Ser	Serine
17	T	Thr	Threonine
18	V	Val	Valine
19	W	Trp	Tryptophan
20	Y	Tyr	Tyrosine

Table 2.1: The Twenty Amino Acids Found in Proteins [104, 33]

the protein is concerned, note that we only focus on processing the primary protein sequences.

2.2 Background of Genomic Sequences and Sequence Comparison

Understanding the relationship of an unknown DNA or protein genomic sequence to well understood sequences in a genomic database allows molecular biologists to assign function to poorly understood sequences. Indeed, in computational biology, one of the goals of sequence analysis is to determine sequence function, structure, and role from inspection and querying with a character string representation (or linear sequence) of a genomic sequence [50]. In this section, we start by illustrating

the development of genomic sequence database, and by describing the importance of genomic sequence comparison, and then present some techniques and algorithms for sequence comparison.

2.2.1 Genomic Databases

Genomic database is the database of the genetic sequences. Comprehensive databases holding DNA and protein sequences are firmly established as central tools in current molecular biology. Electronic databases have become the lifeblood of the field [117]. Genomic databases assist molecular biologists in understanding the biochemical function, chemical structure and evolutionary history of organisms [121]. Given the effectiveness of sequence comparison in molecular biology which will be discussed in Section 2.2.2, it is natural to systematically organize the genomic sequences to be compared.

Large number of DNA, RNA and protein sequences have been determined in the past decades. Some institutional sequence databases have been set up to harbor these sequences as well as wealth of associated data. The rate at which new sequences are being added to these databases is exponential. Historically, one of the most popular DNA sequence databases, GenBank had been doubling in size about every 18 months, but that rate has accelerated to doubling every 15 months due primarily to the enormous growth in data from expressed sequence tags (ESTs) [15, 16, 17], as shown in Figure 2.3.

- **Genomic Sequence Storage and Format.** Sequence information is stored in computers as simple rows of sequence characters. Each character is stored in a byte, which produces 255 possible combinations. The combinations are also called ASCII characters by convention. Each DNA or protein sequence database entry has much information, including an assigned accession num-

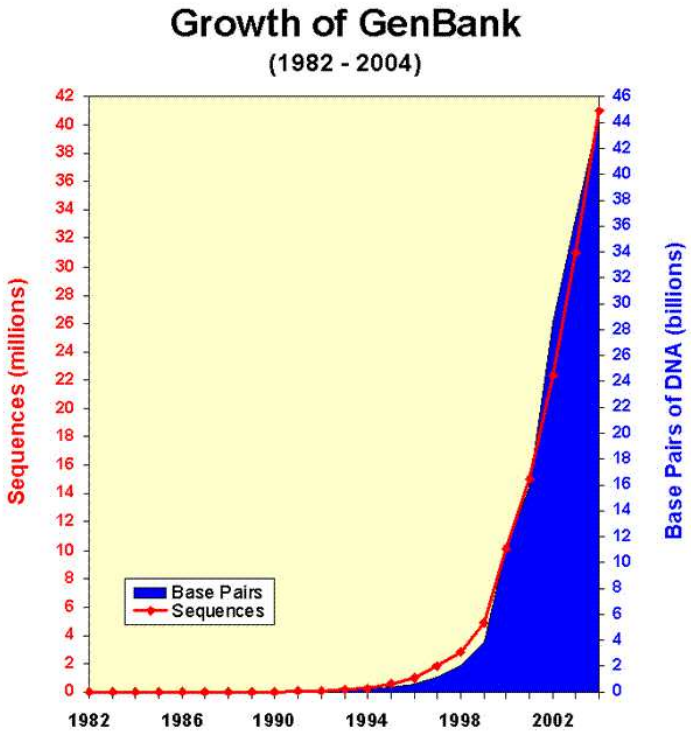

Figure 2.3: Growth of GenBank (1982-2004) [2]

ber; score organism; name of locus; reference; keywords that apply to sequences; features in the sequence such as coding regions, intro splice sites and mutations; and finally the sequence itself [77].

In genomic sequence database, though the formats of databases all are standard ASCII files, they would differ in the presence of certain characters and words that indicate where different types of information and the sequence itself are to be found. FASTA sequence format is one of the most popular sequence formats used for the genomic sequence databases. It includes three parts [77]: (1) a comment line identified by a '>' character in the first column followed by the name and origin of the sequence; (2) the sequence in standard one-letter symbols; and (3) an optional '*' which indicates end of sequence which may or may not be present.

- **DNA Sequence Databases.**

  DNA sequence databases were first assembled at Los Alamos National Laboratory (LANL), New Mexico, by Walter Goad and his colleagues in the GeneBank Database and at the European Molecular Biology Laboratory (EMBL) in Heidelberg, Germany. For DNA sequence in database, in addition to four nucleotide bases, there are eleven standard wildcard characters used to represent different possible substitution in a nucleotide sequence [68]. The most common wildcard is 'N' which represents any base, and some sequences contain thousands of consecutive occurrence of 'N' that represents poorly understood regions of a certain length [121].

  There are several public DNA sequence databases. The larger databases are GenBank [15], the DNA Database of Japan (DDBJ) [4], the European Molecular Biology Laboratory database (EMBL) [101], and Genome Sequence Database (GSDB). The four databases share information between them and they have similar structure and they are also updated daily to incorporate the most recently available sequence data from all sources.

  GenBank stores sequence data generated through the US human genome initiative, which not only focuses on the human genome, but also on model organisms [35]. Historically, the database has roughly doubled in size every 21 months since 1984, however GenBank is now doubling in size every 15 or 16 months. The average sequence length is around 700 bases, with sequences ranging from a few bases to 300,000 bases in length; several sequences are longer than 300,000 bases, but have been stored as separate records according to GenBank guidelines. GenBank contains amino-acid translations for many coding nucleotide sequences, however several solely protein databases also exist.

- **Protein Sequence Databases.** In nature, 20 different amino acids are found in protein sequences. These 20 common amino acids are listed in Table 2.1, while a few nonstandard amino acids might also be present in protein sequence databases. Protein databases are typically well-managed and less redundant than nucleotide databases, commonly including classification of sequences into related families and, in some cases, superfamilies of families. On the protein side, such databases include SWISS-PROT [10, 90, 19] in Europe and Protein Information Resource (PIR) [3]. SWISS-PROT contains cross-references and data from around twenty smaller databases that investigate special organisms and protein types. There also exist some special protein databases, such as the Portable Mouse Genome Database [122].

2.2.2 The Importance of Sequence Comparison in Molecular Biology

The importance of sequence comparison in molecular biology can be traced back to a few earliest researches. The first success story in sequence comparison was to establish the link between cancer-causing genes and a gene involved in normal growth and development [38, 119, 102].

After comparing some sequences, they discovered that some of the sequences, v-sis oncogene, are very similar to the platelet-derived growth factor. At that time the function of v-sis oncogene is still unknown. Based on the similarity of two sequences, they suggested that the function of both v-sis oncogene and platelet-derived growth factor is the same. Indeed, scientist later found that these two genes share similar functions. Another research work [102] also resulted in the same conclusion. They made use of multiple sequence alignment to understand the cystic fibrosis gene. Since all the compared sequences are similar, they share

similar function.

Sequence comparison is one of the most important primitive operations in computational biology as a basis for many other and more complex operations, and it has become essential in modern molecular biology due to the development on the systematic collection, sequence assembly and search of databases containing biomolecular sequences and emergence of many other important applications in molecular biology. In essence, sequence comparison is an operation of finding which parts of the sequences are alike and which parts differ.

In biological sequences (DNA, RNA or protein sequences), high sequence similarity usually implies significant functional or structural similarity [50]. In the genome of a single species and across a very wide spectrum of divergent species, the same and related molecular structures and mechanisms repeatedly occur.

The following quote is from Eric Wieschaus, cowinner of the 1995 Nobel Prize in medicine. Wieschaus says in an Associated Press article of October 9, 1995: "We didn't know it at the time, but we found out everything in life is so similar, that the same genes that work in flies are the ones that work in humans."

And also from a book review on DNA repair [109], it is said that "Throughout the present work we see the insight gained through our ability to look for sequence homologies by comparison of the DNA of different species. Studies on yeast are remarkable predictors of the human system!".

We can obviously see that redundancy and similarity are key phenomena in molecular biology. But there are lots of limits for similarity - humans and mice are different. These differences make conserved similarity even more significant, which in turn makes comparison and analogy very powerful tools in biology [50].

2.2.3 Sequence Alignment and Edit Distance

Mutation in DNA is a natural evolutionary process. DNA replication errors cause substitutions, insertions, and deletions of nucleotides, leading to "editing" of DNA texts. Similarity between DNA sequences can be a clue to common evolutionary origin (say the similarity between globin genes in humans and chimpanzees), or a clue to common function (say the similarity between the v-sys oncogene and the growth-stimulating hormone).

In this section, the definition of two notations will be introduced. One is distance (similarity) of two sequences which gives a measurement on how similar are the two sequences. The other is the alignment of two sequences, which is a way of placing one sequence with another to make clear correspondence between similar characters or subsequences from the sequences.

In 1979, Sims and his colleagues [106] analyzed similar regions from the DNA of two bacteriophages. An alignment was presented between regions from the DNA of the H-gene from phages St-1 and G4 containing 11 matches, which means that there are equal characters in 11 columns in both sequences. They described the procedure of alignment as the "insertion of occasional gaps to maximize number of identities". However, in [107], Smith and his colleagues produced an alignment with 12 matches for the same sequences with the aid of computer. It shows that the use of computer can help detect the intriguing similarity that is easy to be neglected by human beings. We shall introduce the two important concepts: edit distance between two sequences and sequence alignments in the following.

Edit Distance

We start with a detailed examination of the most classic sequence approximate matching problem solved by using dynamic programming, the edit distance prob-

lem. Edit distance focuses on transforming one sequence into the other by a series of edit operations on individual characters, which are insertion, deletion ad replacement. The edit distance between two sequences is defined as the minimum number of edit operations needed to transform the first sequence into the second [50]. Edit distance is also referred to as Levenshtein distance since it was firstly discussed in Levenshtein's paper properly in 1966 [66]. Currently most of genomic sequence comparison algorithms still use this operation or a slight variation of it. In this thesis, we use edit distance instead of Levenshtein distance for consistency and simplicity.

Formally, the definition of edit distance is given as follow:

Definition 2.2.1 Edit Distance

The edit distance between two sequences is the minimum number of edit operations (i.e., insertions, deletions, and substitutions) of single characters needed to transform the first sequence into the second.

The edit distance is a symmetric distance measurement in two sequences, and a deletion in the first sequence can be seen as an insertion in the second, and vice versa. The edit distance problem is to compute the edit distance between two given sequences, along with an optimal edit transcript that describes the transformation.

Sequence Alignment

From the mathematical view of point, sequence alignment is equivalent to the edit transcript between two sequences. Sequence alignment is the procedure of comparing two (pair-wise alignment) or more (multiple sequence alignment) sequences by searching for a series of individual characters or character patterns that are in the same order in the sequences. Sequence alignment is useful for discovering functional, structural, and evolutionary information in genomic sequences. It is

A	T	-	C	-	T	G	A	T
-	T	G	C	A	T	-	A	-

Table 2.2: Sequence Alignment of Sequence s_1 and s_2

important to obtain the best possible or so-called "optimal" alignment to discover this kind of information. In alignment, identical or similar characters are placed in the same column, and nonidentical characters can either be placed in the same column as a mismatch or opposite a gap in the other sequence. The definition of sequence alignment given in [50] is shown as follows:

Definition 2.2.2 Sequence Alignment

A (global) alignment of two sequences s_1 and s_2 is obtained by first inserting chosen spaces (or dashes), either into or at the ends of s_1 and s_2, and then placing the two resulting sequences one above the other so that every character or space in either sequence is opposite a character or a space in the other sequence [50].

There are two kinds of alignments: global alignment and local alignment. The term "global" gives regard to the fact that for each sequence, and the entire sequence is involved in the alignment. Sequences that are quite similar and approximately the same length are suitable candidates for global alignment. As a global alignment, it considers the alignment of the sequences s_1 : "$ATCTGAT$" and s_2 : "$TGCATA$" shown in Table 2.2. A local alignment between s_1 and s_2 is an alignment between a subsequence of s_1 and a subsequence of s_2.

In the early research on sequence alignment, researchers focused on finding the similarity between two entire sequences, i.e. global alignment. Global alignment of sequences is often meaningful when the two sequences are members of the same sequence family, such as the similar DNA or protein family. However, in molecular biological applications, local similarity (local alignment) is far more meaningful in

some sense than global similarity (global alignment).

When the long stretches of unknown sequences are compared, only some internal segments of the two sequences are similar, especially for some protein sequences. In a case of this, local alignment is a more appropriate way to compare sequences. For example, *homeobox genes*, which regulate embryonic development, are present in a large variety of species. Though homeobox genes are very different in different species, only one region of them, *homeodomain* is highly conserved. It means that we need to find the conserved region but ignore the regions with little similarity.

2.2.4 Algorithm of Calculating Edit Distance and Generating Sequence Alignment

Then we should turn to the question of how to compute, via dynamic programming, the edit distance of two sequences along with the sequence alignment together. This algorithm has been discovered and re-discovered many times in different applications from speech processing [115] to molecular biology [85]. Though the details are slightly different with the different algorithms, they have dynamic programming as basis in essence.

The dynamic programming approach consists of three essential components, which are the recurrence relation, the tabular computation, and the traceback [50]. To compute the edit distance of two sequences s_1 and s_2 with length m and n, respectively, the dynamic programming approach will build an $m * n$ matrix and compute the value for every cell in the matrix. The time complexity is $O(mn)$. The basic algorithm for computing the edit distance between two sequences $s_1[1..m]$ and $s_2[1..n]$ is as follows [104]:

A two-dimensional matrix, $M[0..m, 0..n]$ is used to store the edit distance values in matrix. The value of $M(i, j)$ is the edit distance of $s_1[1..i]$ and $s_2[1..j]$. For the

edit distance problem, the base conditions are

$$\begin{aligned} M(i,0) = &\quad \textstyle\sum_{k=1}^{i} d(s_1[k],-), \quad i = 1..m \\ M(0,j) = &\quad \textstyle\sum_{k=1}^{j} d(-,s_2[k]), \quad j = 1..n \end{aligned} \tag{2.1}$$

where a dash("-") denotes a space inserted in a sequence; $d(s_1[k],-)$ denotes the distance weight between character $s_1[k]$ and the inserted space, and $d(-,s_2[k])$ the distance weight between the inserted space and character $s_2[k]$.

The general recurrence is described as below:

$$\begin{aligned} M(i,j) = &\quad min(M(i-1,j)+1, M(i,j-1)+1, \\ &\quad M(i-1,j-1)+d(s_1[i],s_2[j])), \quad i = 1..m, j = 1..n \end{aligned} \tag{2.2}$$

In the above equation, $d(s_1[i],s_2[j])$ denotes the distance weight value of two characters $s_1[i]$ and $s_2[j]$, which can handle insertion, deletion, substitution and equality, where

$$d(s_1[i],s_2[j]) = \begin{cases} 1, & s_1[i] \neq s_2[j]; \\ 0, & otherwise; \end{cases}$$

An alternate way for edit distance in formalizing the relation of two sequences is to measure their similarity rather than their distance. The similarity of two sequences is related with the edit distance, but it depends on the specific scoring matrix. Numerous character scoring matrices have been suggested for DNA and proteins. In protein sequence, the famous scoring matrices are PAM (Point Accepted Mutation) or BLOSUM (Blocks Substitution Matrix) matrix. Scoring matrix is often used in sequence alignment.

When focusing on similarity, the language of alignment is usually more convenient than the language of edit transcript. Every alignment of s_1 and s_2 corresponds

to a path in the matrix of sequence s_1 and s_2. The global alignment problem corresponds to finding the path between cell $[0, 0]$ and cell $[n, m]$ in the matrix.

Smith and Waterman [108] in 1981 proposed a clever modification of dynamic programming that solves the local alignment problem. The algorithm of computing the local edit distance or alignment between two sequences s1 and s2 is similar to the one of computing the global edit distance mentioned above. Also the three steps are necessary, and the time complexity is also $O(mn)$ when computing the local edit distance between $s_1[1..m]$ and $s_2[1..n]$. Obviously, the computational cost is high when the length of sequence is long. Therefore, it is important to find a more cost effective method especially for large database.

In the biological literature, global alignment is often referred to as a Needleman-Wunsch [85] alignment after the authors first discussed global alignment. Local alignment is often referred to as Smith-Waterman [108] alignment since the authors introduced local alignment.

2.3 Research Problems: Genomic Sequence Search, Join and Classification

A couple of examples which describe a list of problems in computational biology were given in [104]. Here we shall cite them to give us a clear picture about the basic operations for genomic sequences in computational biology.

1. We have two sequences over the same alphabet, both about the same length (ten of thousands of characters). We know that the sequences are almost equal, with a few isolated differences such as insertions, deletions, and substitutions of characters. The average frequency of these differences is low. We want to find the places where the difference occur.

2. We have two sequences over the same alphabet with a few hundred characters each. We want to know whether there is a prefix of one which is similar to a suffix of the other. If the answer is yes, the prefix and the suffix involved must be produced.

3. We have the same problem as in (2), but now we have several hundred sequences that must be compared (each one against all). In addition, we know that the great majority of sequence pairs are unrelated, that is, they will not have the required degree of similarity.

4. We have two sequences over the same alphabet with a few hundred characters each. We want to know whether there are two subsequences, one from each sequence, that are similar.

5. We have the same problem as in (4), but instead of two sequences we have one sequence that must be compared to thousands of others.

Problem (1) arises in the situation when we want to compare the results sequenced by two different labs on the same genes, or we want to detect the typing errors when the same long sequence is input to computer. Problem (2) and Problem (3) are about sequence approximate join which is needed to be solved in fragment assembly to conduct the DNA sequencing. Meanwhile, Problem (4) and Problem (5) are mainly about local similarity search in large biological sequence databases.

In addition, advances in proteomics and genome sequencing are generating enormous numbers of genes and proteins at an accelerating rate. Mining these DNA, RNA, protein sequence data to extract significant information is essential in genome processing. In the following, we shall introduce some existing work closely related to the research problems addressed in this thesis.

2.3.1 Genomic Sequence Similarity Searches

Database similarity searches have become a mainstay of computational biology or bioinformatics. Searching a sequence database for sequences that are similar to a query sequence is the most common type of database similarity search. A common reason for performing a database search with a query sequence is to find a related gene or protein in another organism. The sequence of the gene of interest is compared with every sequence in sequence database, and the similar ones are identified, which may provide a clue as function of this query gene with unknown function. Whenever a new gene or a new protein is cloned or sequenced in the laboratory, searching the related database is the next step. Sequence databases and the software to search them have become important and vital tools in modern molecular biology. Database organization and searching have become industries, and "discoveries based solely on sequence have become routine" [94]. It would be hard to find a recent DNA-based discovery that didn't use these tools [14]. The problem description of similarity search in the sequence databases can be defined formally as follows:

Problem 2.3.1 *Given a query sequence Q and a target sequence database D, find a set of subsequences of Q such that each subsequence Q' in the set is highly similar to some subsequence D' of D. The similarity between Q' and D' is computed as a function of the* **edit distance**, *$edit(Q', D')$.*

Sequence similarity search programs are versatile tools for the molecular biologist, frequently able to identify possible DNA coding regions and to provide clues to gene and protein structure and function. A great amount of work has been done to improve search efficiency and effectiveness, and we now review some existing work of homology search in genomic sequence databases.

BLAST – The Standard Tool

This section first examines BLAST – the standard tool that is being used by biologists – before discussing other related work.

The BLAST programs [8, 9] are among the most frequently used to search sequence databases worldwide since BLAST programs are available on the World Wide Web through a large server at the NCBI (http://ncbi.nlm.nih.gov), and at many other mapping sites. BLAST has been a dominant searching engine for the biological sequence databases since 1990.

BLAST is the standard tool that molecular biologists use to search for sequence similarity in genomic databases, and is known to be one of the more efficient tools available. It is a local similarity search algorithm that incorporates heuristics to prune the search space in order to keep the computational cost low. As a result, it may miss the (global) maximum segment pair (MSP).

BLAST is actually a collection of programs, which are used to deal with different problems in sequence database search. However, the basic principle of BLAST is the same. BLASTN focuses on searching DNA sequence database, while BLASTP is to search protein sequence database, which uses PAM or BLOSUM matrix to compute similarity scores.

The operation of matching a query sequence against a library of data sequences is carried out in three steps. First, BLAST generates a list of query windows of length W (W-tuples), where W is a tunable parameter, by placing a window of size W over every position of the query.

In the second step, BLAST scans through every database sequence to see if it contains a W-tuple that can match with one of the query W-tuples to produce a *seed* with a score greater than or equal to a predetermined threshold T. The sequential scan of the entire sequence database employs a finite state machine algorithm.

Moreover, BLAST encodes every database sequence into bit representation, and this step searches the bit-encoded data rather than the original database sequences.

Sample Query and Database sequence	
Query	GCAACGGGCAATATGTG
Data	ATTGAAACGGGCAATATGTC
Settings	k = 11 (default for BLAST), T = 0
Step 1: Create query windows	GCAACGGGCAA CAACGGGCAAT AACGGGCAATA ACGGGCAATAT CGGGCAATATG GGGCAATATGT GGCAATATGTG
Step 2: Search seeds	AACGGGCAATA ACGGGCAATAT CGGGCAATATG GGGCAATATGT
Step 3: Extend seed	AACGGGCAATATGT
Step 4: Align sequences	---GCAACGGGCAATATGTG ATTGAAACGGGCAATATGTC

Figure 2.4: Illustration of BLAST Search Steps

Finally, in the third step, the seeds are extended in an attempt to locate longer similar segment pairs. Each seed is extended on the left and right to determine if it is contained within a longer segment pair whose score is greater than or equal to S. The algorithm stops extending the seed in one direction when that causes the score to fall a certain distance below the best score found so far. In BLAST-2 [9], seed extension is restricted to sequences that share at least two non-overlapping seeds, with the same relative offset and within a certain distance of one another. This

stricter criterion trades sensitivity for speed. BLAST-2 also tries to produce gapped alignment from multiple local MSPs (maximum segment pairs) for each sequence, whereas the original BLAST program often finds several alignments involving a single database sequence.

BLAST permits a tradeoff between speed and sensitivity, by adjusting the settings for the threshold T and the length of W-tuple. A higher T or W value yields greater speed by avoiding spurious short seeds arising by chance in data sequences that cause unnecessary comparisons. However, it may increase the probability of missing weak similarities (i.e., less sensitive). For DNA queries, the default settings are $T = 0$ and $W = 11$.

Figure 2.4 illustrates how BLAST works. Here, $W = 11$, $T = 0$, and the query sequence has a length of 17 base pairs leading to 7 query windows in step 1. In step 2, four of these windows are identified as having matching segments in the data sequence. Step 3 then extends the first of these seeds to give the alignment shown in step 4.

BLAST has a few popular implementations, including the NCBI-BLAST [5] from NCBI, and WU-BLAST [7] from Washington University. We evaluated NCBI-BLAST because its source code is freely available for download. To understand the cost breakdown, we recompiled the source codes of BLAST with the profiling option and ran the program with a profiler, gprof [6] to generate statistics on the time spent on each function. These statistics are grouped to produce the time spent on database retrieval, W-tuple matching, and seed extension.

We ran BLAST on a DNA database of 487 KB sequences with 1.7 GB letters to study the effect of query length on BLAST. For each query length, we generated 100 random queries and obtained the average running time. The results, plotted in Figure 2.5, show that the seed searching time dominates the total response time

for BLAST. Database read time and seed extension time are just a small portion of the total response time.

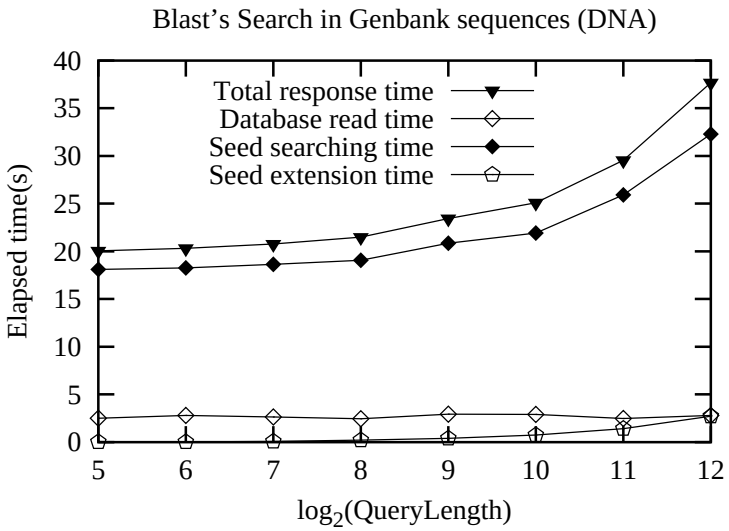

Figure 2.5: Breakdown of BLAST's Search Time

In all, BLAST is a heuristic method for finding similar regions between two genomic sequences. It regards the exact match of W contiguous bases as candidates which are then extended along the left side and the right side to obtain the final alignments. Unfortunately, BLAST faces the dilemma of DNA homology search: increasing the seed size W decreases sensitivity while decreasing the seed size results in too many random results.

Sequence Scan Based Similarity Search

The widely used sequence similarity finding programs include Needleman-Wunsch [85], Smith-Waterman [108], FASTA [96, 95] and BLAST [8, 9].

The Needleman - Wunsch algorithm performs global sequence alignment using a dynamic programming algorithm. Its computational complexity is $O(mn)$. The

Smith-Waterman algorithm [108] is a heuristic approximation to the Needleman-Wunsch [85] algorithm using local sequence alignment to find the similar sequence from the sequence database. The computational complexity is still $O(mn)$, with a coefficient of complexity smaller than Needleman-Wunsch. It performs local sequence alignment between a query sequence and the target sequence using a dynamic programming algorithm.

The FASTA algorithm [96, 95] identifies regions of local sequence similarity by first identifying candidate similar sequences based on the shared W-tuples. FASTA is a tool for the analysis of protein and DNA sequence similarity that achieves a tradeoff between the selectivity and sensitivity. The search algorithm used by FASTA includes four steps in computing the score for pair-wise sequence similarity [96, 95].

1. In the first step of comparison, FASTA uses a lookup table to locate all identities between two DNA or protein sequences, and find the best diagonal regions.

2. In the second step of comparison, FASTA evaluates these regions found in the first step and rescores them using a scoring matrix, which allows conservative replacements and runs of identities shorter than W-tuples to contribute more score to the final similarity score between the two pair-wise sequences. A certain sub-regions, say 10 sub-regions, with higher scores will be found, which are called "initial regions".

3. In the third step of comparison, FASTA will check the "initial region" identified in the second step to see whether several initial regions may be joined together. FASTA also computes the final alignment score of the combination of several initial regions and ranks the database sequences according to the

final scores.

4. In the fourth step of comparison, the highest scoring database sequences are aligned again using a modification of the optimization method described by Needleman and Wunsch, Smith and Waterman. This step considers all possible alignments of the query and database sequence that fall within a band centered around the highest scoring initial region. The final optimized alignments are reported as the results.

FASTA also considers local similarity analysis since molecular biologists are often interested in the detection of similar subsequences within longer sequences. Statistical significance of an alignment is also evaluated in the tool of FASTA by providing additional tool, which compares one sequence with randomly permuted versions of the potentially related sequences. A simple Monte Carlo shuffle analysis can be used in the evaluation of the statistical significance.

Both FASTA and BLAST have formed the basis for their latest versions that provide very powerful search tools for the molecular biologists, and are freely available to run on desktops.

PatternHunter [70] is an improvement on BLAST both in speed and sensitivity through the use of non-consecutive W characters as model, where W is the weight of this model. Though it seems that non-consecutive W seed is a very simple change, it has a very surprisingly large effect on sensitivity. They denote a model by 0-1 string, where the 1-positions represent the matches for the "care" part, while the 0-positions are the part of "do not care". According to the reported experiment results by Bin Ma et al. [70], PatternHunter outperforms BLAST both in speed and sensitivity. However, in essence, PatternHunter's basic principles in algorithm are similar to those of BLAST. Therefore, it also suffers the dilemma of BLAST.

Category	**Method**
Suffix structure	QUASAR [25] Oasis [75]
Multi-dimensional indexing approaches	Wavelet-Based Method [61] Four different transformation methods [91]
Other tree-structured index structures	SPINE [86] *ed*-tree [110] SST [47]
Inverted file index	CAFE [121]
Hash structure	LSH-ALL-PAIRS [23] RAMdb [44] FLASH [26] SSAHA [88] BLAT [63]

Table 2.3: Genomic Sequence Indexing Based Similarity Search Methods

Sequence Indexing Based Similarity Search

The above-mentioned similarity search methods are based on sequence scan. The alternative to exhaustive search is to use an index-based approach. In order to improve the efficiency of similarity search, the researchers in the area of computer science have also proposed many index structures for genomic sequence databases. These index structures are summarized in Table 2.3. We review these schemes in detail subsequently.

The family of suffix tree and suffix array are well-studied indexing structures for facilitating string-related retrieval [120, 74, 71, 72]. QUASAR [25] applies a modification of q-gram filtering on top of a suffix array. It pre-computes the positions of the hit lists in the suffix array A for all possible q-grams and stores them in a search array of size $|\Sigma|^q$, where $|\Sigma|$ is the number in the alphabet. QUASAR is faster in detecting the regions with strong similarity, however, its performance deteriorates dramatically if the compared sequences are weakly similar. Also, the resultant index structure based on the suffix array and suffix tree is large compared

to the size of the sequence database. For example, an index file of 2GB is built for a DNA sequence of 20.5M when a suffix tree with links is used. Even if the suffix tree is used without links as proposed in [57], the suffix tree structure index is still nearly 10 times the size of the original sequence database. In [78], Muthukrishnan and Sahinalp proposed an index structure for approximate nearest search for sequences. The index structure is based on a suffix array, and the resulting index structure is still four times the size of the sequence database.

Oasis [75] is a novel search algorithm which uses a dynamic programming A*-search driven by a suffix tree index built on the DNA sequence dataset for fast and accurate local alignment search on sequences. The study shows that Oasis is often an order-of-magnitude or more faster than Smith-Waterman algorithm for the short query sequence. And it also can be used in online mode as it returns the results in decreasing order of the matching scores. Since Oasis is driven by a suffix tree, it also suffers the weakness of the suffix tree, say large size of index structure.

For long genome sequences, constructing suffix tree is very time-consuming, and the size of the suffix tree is typically too large to fit in memory. Consequently, many proposals [42, 57, 111, 112, 12, 13] have focused on reducing the time and space for suffix tree construction. In [111, 112], the authors proposed an approach for efficiently constructing large suffix trees on disk. The approach takes into account the effects of buffering policies during the suffix-tree construction. [111, 112] demonstrated that a suffix tree can be built on the entire human genome in 30 hours based on the proposal. In [12, 13], Bedathur et al. developed a low overhead buffer management policy called TOP-Q to exploit the pattern of access over the suffix tree during its construction. They also evaluated the impact of buffering and internal node implementation choices on the construction of a suffix tree on disk. In [71, 72], Manber and Myers proposed suffix arrays as a space-saving alternative

to suffix trees and presented the first algorithms for suffix array construction and use. Subsequent research on compressed suffix arrays [49, 64, 103, 53] and the FM-index [43] has revealed that self-indexing structures are possible, which can search for and report matches without the need for the original sequences to be stored. Because of the non-sequential access patterns exhibited by current suffix array algorithms, all the work with such algorithms assume that their structures can fit in memory. In [64], for a human DNA which comprises about 2.8 billion characters, a compressed suffix array occupies about 2GB; moreover, the compressed suffix array for a human genome can be constructed in 2 hours, which is faster than suffix tree. [64] also demonstrated that searching on compressed suffix array is much faster than searching on hard disk based suffix tree.

In [86], Neelapala et al. proposed the SPINE index data structure, a carefully engineered horizontally-compacted tire index. In SPINE, a set of forward and backward edges are employed to ensure that all suffixes of the data string are captured in index structure. The algorithms are also designed for both online construction of the SPINE index structure and performing complex searching based on SPINE. The authors demonstrated that SPINE requires significantly less space than standard implementation of suffix tree, and SPINE takes less time for both construction and searching compared to suffix tree. But in SPINE, it only considers the occurrences of a matching pattern in data strings instead of approximate matches.

Recently, some attempts have been made to transform DNA sequences into numerical vector spaces to allow the use of multi-dimensional indexing approaches for sequence similarity search. The distance function defined in the new spaces should be the lower bound of the edit distance between the related sequences corresponding to the two vectors. A wavelet-based method is proposed to map the subsequences of a database into a 2σ dimensional integer vector where σ is the

alphabet size of the sequence in [61]. The coefficients of the integer vector are then indexed by MRS-Index. Range queries and nearest-neighbor queries are performed based on the index structure. Though this method avoids false dismissals and offers very fast filtering, its approximation of edit distance is not sufficiently tight, and as a result, the overhead of refining results for final output can be very high. MRS-Index returns only approximate answers for sequence approximate match [86], and it is designed to find the global alignment for the whole query sequence, but not for the subsequence of the whole query sequence. This is thus different from what we are trying to achieve in this thesis.

Four different transformation domains and five different distance functions are investigated in [91]. Three strategies, namely scanning access, the R-tree based indexing and scalar quantization-based indexing are built on top of the transformed vectors. The distance in this new transformed space is not expected to keep the exact same relationships as the edit distance of the originating strings.

A new index for DNA sequences, called the *ed*-tree [110] is proposed to support probe-based homology search in DNA sequence databases. The tree-structure index is constructed by sliding a window on the DNA data sequence with the skip intervals. In each window, the subsequences are segmented in terms of a segment length vector. The paper showed that query using the *ed*-tree is up to six times faster than BLAST. But the size of the tree-structure index is larger than the sequence database and as well as it is very time-consuming to build the *ed*-tree for DNA sequences.

In SST [47], 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 has been shown to be much faster than BLAST when searching for highly similar

sequences. Unfortunately, since the distance between sequences in the transformed vector space does not correspond well with the actual edit distance, a larger number of false dismissals may occur if the similarity between the query sequence and the target sequence is not sufficiently high.

Inverted file index in text processing has also been applied to a biological sequence database. Williams et al. [121] proposed a search algorithm in a research prototype system, CAFE, which uses an inverted index to select a subset of sequences that display broad similarity to the query sequence. CAFE is based on a partitioned search approach, where a coarse search using an inverted index is used to rank sequences by similarity to a query sequence, and a subsequent fine search is used to locally align only a subset of database sequences with the query. The CAFE index consists of three components: a search structure, which contains the index terms or distinct intervals, that is, fixed-length overlapping subsequences from the collection being indexed; inverted lists, which are a carefully compressed list of ordinal sequence numbers, where each list is an index of sequences containing a particular interval; and a mapping table that maps ordinal sequence numbers to the physical location of sequence data on disk. A compression scheme is employed to make the index size more manageable. CAFE evaluates a query by representing it as a set of intervals, retrieving the list for each interval, then using a ranking structure to store a similarity score of each database sequence to the query. CAFE bears the additional overhead of uncompressing the index at runtime for a query. The experiments conducted in [121] show that CAFE is faster but also less sensitive than BLAST when searching for very similar sequences.

Buhler [23] proposed a method, LSH-ALL-PAIRS, 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 may miss some short alignments in a collection of sequences.

RAMdb (Rapid Access Motif database) [44] is a system for finding short patterns called *motifs* in genome databases. Each genome sequence is indexed by its constituent overlapping intervals in a hash table structure. For each interval, an associated list of sequence numbers and offsets is stored. This allows a quick lookup of any sequences that match a query sequence. A long query sequence is split into shorter non-overlapping motifs that are used to query the database. RAMdb is best suited for query motifs whose length is equal to or slightly longer than the indexed interval length, and has been shown to produce up to an 800-fold speedup in search time over exhaustive approximate pattern matching schemes. It requires a large index that is twice the size of the original flat-file database including the textual descriptions and suffers from a lack of special-purpose ranking schemes for identifying initial match regions. In addition, the non-overlapping interval of query motifs can lead to false dismissals.

The FLASH search tool redundantly indexes genome data based on a probabilistic scheme [26]. For each interval of length n, the FLASH search structure stores, in a hash-table, all possible similarly-ordered contiguous and non-contiguous subsequences of length m that begin with the first base in the interval, where $m < n$. As an example, for a nucleotide sequence ACCTGATT the index terms for the first $n = 5$ bases, where $m = 3$, would be ACC, ACT, ACG, ACT, ACG and ATG with each of the permuted strings beginning with base A, the first base in the interval of length $n = 5$. The hash-table then stores each permuted m-length subsequence, the sequences that contain the permuted subsequences, and the offsets within each sequence of the permuted subsequence. The key idea is that the permuted scheme

approximates a reasonable number of insertions, deletions, and substitutions in the genome sequences. The authors found that FLASH was of the order of ten times faster than BLAST for a small test collection, and was superior in accuracy and sensitivity in determining homologies. Unfortunately, the hash-table index is uncompressed and impractically large – For a nucleotide collection of around 100 MB, the index requires 18 Gb on disk, around 180 times the collection size.

Another algorithm, Sequence Search and Alignment by Hashing Algorithm (SSAHA) [88], performs fast searches on databases containing multiple gigabases of DNA sequences. It hashes the non-overlapping W-tuples in the data sequences. Search for a query sequence is done by obtaining from the hash table the "hits" for each W-tuple in the query sequence and then performing a sort on the results. The final similar regions can be created by joining together the exact matchings sufficiently close to one another. Though it is fast, it also suffers from the same dilemma of BLAST.

BLAT [63] constructed an index of nonoverlapping W-tuples and their positions in the sequence database. It then looks up each overlapping W-tuple of query sequence in the index. BLAT is fast due to the index of all overlapping W-tuples in the genome. The index can be fit inside memory for fast access. BLAT also returns each region of homology between the data sequence and query sequence as separate alignment and stitches these regions together into a larger alignment. In addition, instead of using perfect matches of a W-tuple seed to trigger an alignment, BLAT also provides an option (denoted as near-perfect match) to allow one mistake in a seed hit. Every possible W-tuple that matches in all but one position. However, for a given sensitivity, the near-perfect match criteria runs 15 time slower than the perfect match criteria in BLAT [63].

2.3.2 Genomic Sequence Approximate Join

The standard approaches to assembling DNA sequence from randomly located and randomly oriented reads use "overlap-layout-consensus" [50, 58, 97, 98]. In this section, we discuss some of the related work in sequence approximate joins which are relevant to the first step of DNA sequence assembly, overlap detection, and also relevant to sequence by hybridization.

To find the most similar suffix and prefix match, we can use the standard dynamic programming recurrences to compute the similarity between s_1 and s_2 [50]. For the sequence s_1 and s_2 with the length m and n respectively, the best suffix-prefix match of the pairs takes $O(mn)$ time. Therefore, the computation of the best longest suffix-prefix matches of two sequences is actually a bottleneck in sequence assembly when the length of sequences is very large.

Hence many researchers have started to consider some approaches to speed up the first step of assembly by skipping the dynamic programming computation for those unattractive pairs. In [50], a method that computes the length of the longest common substrings is presented for this purpose since two strings which have the sufficient overlap should have at least one significant long common substring. Chen and Skiena in [31] proposed a method called in-depth examination of exact matching with dismissal. This method speeds up the dynamic programming with the cost of decline in quality.

Cohen [34] presented a framework for the approximate string matches using the vector space model of similarity. But the similarity metric for the string joins is TF.IDF term weighting, rather than the edit distance used in our method.

Another method to generate the candidates of approximate string joins is to use q-grams. In [48], Gravano et al. used the concept of q-grams in approximate string joins in relational database by augmenting a database with q-grams information,

which is needed to run the approximate string joins. Three filtering techniques, count filtering, position filtering and length filtering are used to get a set of candidate answers for string joins efficiently.

Jin, Li and Mehrotra [59] also proposed a two-step process for string join. Firstly, the strings are mapped into the points in a multi-dimensional vector space by *FastMap* algorithm so that the mapped space preserves the distance over strings. In the second step, the high-dimensional similarity join algorithm proposed in [52] is used to conduct a multi-dimensional similarity join. This approach can support any distance measure between strings, but it cannot guarantee to find all the results.

A great amount of work has been conducted to design the efficient and effective solutions to the approximate sequence joins problem. A good survey on the current techniques for approximate string matching can be found in [84].

2.3.3 Protein Subcellular Localization Prediction

A number of systems have been developed to support the prediction of subcellular localization based on amino acid sequences. The pioneering work was done by Nakai and Kanehisa in [81, 82]. They developed an expert system that makes use of various kinds of knowledge organized as "if-then" rules for predicting localization sites of proteins based on the N-terminal sorting signals from the amino acid sequence. Von Heijine et al. [116] and Nielsen et al. [87] proposed to use the signal peptides, mitochondrial targeting peptides and chloroplast transit pepetides for the protein subcellular localization prediction. As an improvement, TargetP [40] is an integrated prediction system built from two layers of neural networks based on combining these individual sorting signal predictions. TargetP has a relatively well-working cleavage site prediction capability for all involved target sequences. The

prediction accuracy highly depends on the quality of protein N-terminal sequence alignment. However, the prediction accuracy based on sorting signals is not stable since annotating the N-terminal using the existing gene identification methods is often unreliable.

In [83], sequences of intracellular and extracellular soluble proteins were analyzed statistically in terms of amino acid composition and residue-pair frequencies. Moreover, the authors have declared that introcellular and extracellular differ significantly in their amino acid composition. Reinhardt and Hubbard [100] used neural networks to predict the subcellular location of proteins in prokaryotic or eukaryotic cells from their amino acid composition. This study also examines whether the differences in amino acid composition between other subcellular locations is strong enough to establish a prediction method. A method based on the amino acid composition should be comparatively stable when the leader sequences are missing or only partially included. In [32], Chou and Elrod proposed a covariant discriminant algorithm to predict the subcellular location of a query protein according to its amino acid composition. In [55], SVM was introduced to predict the subcellular localization of proteins from amino acid composition, and Hua and Sun [55] examined three locations in prokaryotic organisms: cytoplasmic, periplasmic and extracellular; and four locations in eukaryotic cells: cytoplasmic, extracellular, mitochondrial, and nuclear, and they concluded that the amino acid composition based prediction method can be a complementary method to other existing methods based on sorting signals. A set of SVMs were trained to predict the subcellular location of protein based on its amino acid, amino acid pair, and gapped amino acid pair composition in [92], and a voting scheme was proposed to combine the prediction of these different compositions. The weakness of this method is that it may not catch the information on sequence order and the inter-relationships

between the amino acids.

Dipeptide composition was presented in [56], which is another representative form of proteins incorporating neighborhood information. A fuzzy k-NN method based on the protein's dipeptide composition was proposed for prediction of subcellular localizations. Fourier analysis and tri-peptide frequency are proposed in [65] as features to capture the information about long-range correlations and global symmetries in proteins which are missed by the global amino acid composition. The SVM was also used as the classifier based on the proposed features.

Instead of using amino acid sequence alone, Lu et al. [69] explored the use of database text annotations from homologies and machine learning to substantially improve the prediction of subcellular location. Each feature of the query protein is a Boolean value corresponding to the presence or absence of a token (word or phase) from certain field of the homologous sequence Swiss-Prot dataset entries. These features were not obtained from the training dataset, and they were computed automatically from Swiss-Prot database.

Based on the concepts of frequent patterns in data mining, frequent patterns are used as the basis for designing accurate and efficient prediction algorithms for outer membrane proteins [105]. A frequent subsequence is a consecutive subsequence of amino acids that occurs in outer membrane proteins. Two methods were presented by using frequent subsequences to predict outer membrane proteins: one method uses frequent subsequence to construct classification rules, the other uses frequent subsequence as features for an SVM. The methods are only applied on the prediction of outer membrane proteins.

In order to improve the prediction accuracy, an SVM-based hybrid module [18] was developed by using many features of a protein, which consisted of an input vector of 458 dimensions (400 dipeptide compositions, 33 properties, 20 amino

acid compositions of the protein and 5 from PSI-BLAST output). The hybrid module can capture more information about a protein that is crucial for detecting subcellular localization of proteins. In essence this method is a hybrid module which integrates some existing subcellular localization prediction methods.

For bacterial location prediction, the most widely used tool is PSORT-B [46], which combines several methods, including SCL-BLAST, identification of motifs and sorting signals and machine learning methods. PSORTb v2.0 [45], the updated version of PSORTb tool was extended by including Gram-positive organisms and increasing its predictive coverage to improve significantly upon the original program.

2.4 Summary

In this Chapter, we have introduced the foundation required for the research issues being addressed in this thesis. In particular, we defined the problems and discussed related work of these research issues.

CHAPTER 3

Piers: An Efficient Model for Similarity Search in DNA Sequence Databases

3.1 Introduction

Increasing interest in genetic research has resulted in the creation of huge genomic databases, and similarity search in genome sequence databases has become a basic operation in computational biology. Many algorithms have been developed, and one of the most fundamental is the Smith-Waterman alignment algorithm [108], which is based on dynamic programming concept for finding an optimal alignment between a query and its target sequence in $O(mn)$ time, where m and n being the length of the two sequences respectively. Obviously, the overhead is high for large m and n. The work has however generated interest and followup work in developing faster algorithms and tools.

One of the most common approaches adopted in efficient retrieval from large

and complex database is the two step filter-and-refine approach. An algorithm based on such principle consists of two phases. In phase 1, it reduces the number of negatives quickly as cost effectively as possible. In phase 2, it goes through each member in the retrieved result set and performs a more detailed and costly check to remove the false positives that are not filtered out earlier. Likewise in computational biology, many proposals have adopted this approach [96, 8]. In this context, the sequences database is first broken into short segments and matched against the query sequences. Then segments with low similarity are first filtered off while more complex computations are done on the remaining high similarity segments to form the final result. The efficiency and sensitivity of such approaches are highly dependant on the choice for the length of the segments and how regular the segments are being sampled from the database sequences. For example, in the case of BLASTn [8], a segment length of 11 characters is usually used and segments are obtained from every position in the database sequence.

In this chapter, we propose a hash based model for efficient and sensitive DNA sequence search. It introduces the notation of "pier", which is defined as a segment with length ℓ_p and located at position *pos* in a data sequence[1]. The proposed technique focuses on effective filtering in the first phase of the search since the performance in the second phase is typically similar across most of such approaches.

During pre-processing, the piers are randomly picked from a data sequence S based on the principle that at least one pier is fully contained within any subsequence of S having a minimum length. These piers are then stored in a hash table for efficient access. Such an approach gives us a much lower pre-processing time compared to BLASTn and other index building approaches.

During query time, by picking each query pattern (or seed) generated from

[1]The name "pier" is selected since we believe that these small set of selected segments should be enough to "support" highly sensitive similarity search for the whose database sequence.

the query sequences and enumerating its neighbors (i.e. segments of the same length that are within a small edit distance from the query pattern), candidate buckets can be located in the hash table very efficiently. Using the algorithm, we can enumerate all the neighbors which are potential candidates without searching through the whole hash space.

In order to achieve high efficiency in searching, we have to solve the insertion collision in hash table. We propose to pre-compute a global penalty matrix (called GPM), with size $4^{2\omega}$ for all the possible DNA segments, where ω is the length of suffix of pier segment. We keep length ω small to ensure that GPM is small enough to reside in memory. By looking up GPM, we can obtain the set of pier candidates which are similar to query pattern without computing the edit distance.

After completing the search for the pier candidates which are similar to the query pattern q_i, we use these pier candidates to detect the regions in sequence database which are similar to the query subsequence. All the subsequences which include the candidate piers are needed to be aligned and verified to see whether they are really similar to the query subsequence which include query pattern q_i. We then extend the candidate into both directions for the final alignments between data sequence and query sequence.

In summary, this chapter has the following contributions.

- We propose to use the extracted piers, not the whole sequence database for efficient sequence similarity search with acceptable sensitivity. The issue of sensitivity of our proposed pier model is also addressed. The piers are the segments of data sequence which meet the principle of pier extraction. The proposed pier model is applicable to other similarity search methods for selecting some segments as index terms.

- A hash table is built on the extracted piers for efficient access. The hash key

of the pier is the encoded integer of the prefix of pier in our method. Every pier is inserted into one of the buckets in the hash table according to the hash key of the pier.

- The algorithm of enumerating all the neighbors of the query pattern is also devised. By using the algorithm of neighbor enumeration, the candidate buckets can be located in hash table very efficiently without scanning all the hash keys in it.

- In order to reduce the computation cost of verifying the piers in the collision list in a candidate bucket, a method is presented to solve it. The method is to only look up the pre-built GPM for detecting the pier candidates.

- We show theoretically and empirically that the proposed approach can efficiently detect biological sequences that are similar to a query sequence with very high sensitivity.

The rest of the chapter is organized as follows. In Section 2, we shall provide some definitions and formally give a problem statement discussed in this chapter. In Section 3, the pier model is proposed for DNA sequence search and the sensitivity of the proposed pier model is also analyzed theoretically. In Section 4, a hash-based pier model is presented for efficient sequence search in DNA sequence database. Section 5 shows how sequence similarity search can be efficiently processed using the proposed hash-based pier model with analysis on the space and time complexity of the method. The test data and experimental results are presented in Section 6. Section 7 concludes the chapter.

3.2 Notations and Problem Statement

In this section, we shall give some definitions and provide a formal problem statement.

3.2.1 Notations and Definitions

The most commonly used distance measurement for two sequences is referred to as edit distance. As we have stated in Chapter 2, it is a simple but fairly accurate measure for the evolutionary proximity of two DNA sequences [50]. The edit distance between two sequences is the minimum number of edit operations (i.e., insertions, deletions, and substitutions) of single characters needed to transform the first sequence into the second. Edit distance is used as a distance measurement for two DNA sequences in this chapter.

In this chapter, small segments from a sequence database are referred to as *piers*. Formally, a pier is defined as a tuple $\langle p, pos \rangle$, where pier sequence p is a segment of length ℓ_p extracted from a DNA sequence, and pos is the list of positions for the pier sequence p occurring in the data sequences.

For notation simplicity, we shall use p to refer to as both the segment with the corresponding positions, and the segment itself. It should be clear in the context.

Intuitively, by using piers, we are able to avoid indexing the whole sequence, while selecting the appropriate number of piers and the distribution of the piers to achieve desirable sensitivity of search. Formally, a pier has the following definition.

Definition 3.2.1 Pier

A pier is defined as a pair $\langle p, pos \rangle$*. Pier sequence* p *is a segment extracted from DNA sequence of length* ℓ_p*.* pos *is the list of positions for the pier sequence* p *occurring in the sequence database.*

Based on the definition of piers, we define *span* to be the segment between two adjacent piers in the proposed pier model.

Definition 3.2.2 Span

The span is the segment between two adjacent piers in the proposed pier model.

The length of span ℓ_s must meet the pier extraction principle which will be described later. Here, candidate is referred to as the set of the target subsequences that we want to search further in the sequence database, and it is stated as follow formally.

Definition 3.2.3 Candidate

Given the query sequence Q, the pier p can be regarded as the candidate if it satisfies the formula: $edit(p, q_i) \leq \theta$, where q_i is the query pattern with length ℓ_p in query sequence Q.

The notations to be used in this chapter are summarized in Table 3.1.

3.2.2 Problem Statement

The approximate sequence match problem can be classified into two categories: whole sequence matching and subsequence matching [41]. Since the subsequence matching problem is a generalization of the whole matching problem, we confine our attention to a subsequence match problem. The sequence similarity search problem can be described as follows:

Problem 3.2.1 *Given the length l and edit distance ϑ, find all subsequences S in $\mathcal{D}$ which have length $|S| \geq l$ and $edit(S, Q') \leq \vartheta$ for subsequence Q' in query sequence Q.*

Notation	Description
D	a DNA sequence database
$\|D\|$	the size of DNA sequence database D
S	the data sequence in D
$\|S\|$	the length of sequence S
$S[i, j]$	the subsequence of S from i to j
$s \subset S$	s is a subsequence of sequence S
p_i	the ith pier segment along D
ℓ_p	the length of a pier
λ	the length of prefix of pier segment
ω	the length of suffix of pier segment, $\lambda + \omega = \ell_p$
ℓ_s	the length of a span
Q	a query sequence
q_i	the ith query pattern in Q with length ℓ_p
θ	edit distance threshold allowed for pier candidate and query pattern
ℓ_{min}	the minimum length of the high similarity region
$edit(S, Q)$	the edit distance between two sequences S and Q

Table 3.1: The Notations

We adopt the filter-and-refine approach, and according to the analysis of search cost for BLASTn in Chapter 2, we know that the seed searching time dominates the total response time for BLASTn, while database read time and seed extension time are just a small portion of the total response time. Therefore, in this chapter, we shall focus on the filtering problem defined as follows:

Problem 3.2.2 *Given edit distance threshold θ, find all candidates p, $p \subset S$ in data sequence S, $S \in D$ for each query pattern $q_i \subset Q$, where $edit(p, q_i) \leq \theta$.*

3.3 The Proposed Pier Model

This section describes our pier model for biological sequence similarity search. The main assumption in the pier model is that users are only interested in high similarity region that is of length greater than a minimum length l_{min}. Based on this

assumption, we define piers in a biological sequence database as some segments in the data sequences, which meet Property 3.3.1 defined below.

Property 3.3.1 Pier Extraction Principle

The Pier Extraction Principle *states that at least k piers should be fully contained in any subsequence with length no less than the threshold of minimum length* ℓ_{min}. *This means that the following formula must hold:*

$((k+1)\ell_p + k\ell_s) \leq \ell_{min}$.

Intuitively, the pier extraction principle simply ensures that consecutive piers are selected in the data sequences such that at least k of the piers will be fully contained in any candidate subsequence of length no less than l_{min}. In the pier model, the pier sequences can be extracted randomly or periodically from the data sequence as long as the pier extraction principle (3.3.1) is satisfied. As explained later, this is done to reduce the probability of a high similarity region being lost in a worse case scenario. For ease of discussion, we extract the piers from the sequence database periodically.

3.3.1 Generation of the Piers

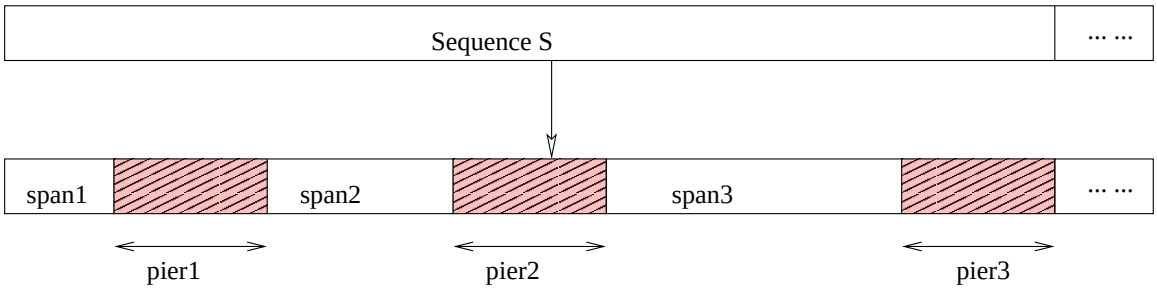

Figure 3.1: An Example of the Piers Extracted from DNA Sequence

In Algorithm 1, we describe how the piers are extracted from the sequence database. For each sequence S in database, startOffset is the start position of

the first pier in S. After the first pier is extracted, the current offset o is set as startOffset. The start position of next pier will be specified from the interval $[o+\ell_p, o+\ell_p+\ell_s]$ since it must satisfy the pier extraction principle. This extracting process will be executed until the end of the last sequence in the sequence is reached. For each pier, the pier sequence and its corresponding position in the sequence database are stored. Figure 3.1 gives a visual description on how the piers are extracted from the data sequence S. In our implementation, we extract the piers from data sequences periodically, which means that the start position of next pier is $o + \ell_p + \ell_s$ if the position of current pier is o.

Algorithm 1 Pier Generation Algorithm
Input: DNA sequence database D, Pier length ℓ_p, Span length ℓ_s.
Output: A set of piers $P = \{p_1, p_2, ..., p_n\}$.
Method:

```
1: P ← ∅, i ← 1;
2: for each sequence S_i ∈ D do
3:    o ← startOffset;
4:    while |S_i| − o ≥ ℓ_p do
5:       p_i ← ⟨S_i[o, o + ℓ_p − 1], o⟩;
6:       P ← P ∪ {p_i};
7:       o ← a random number ∈ [o+ℓ_p, o + ℓ_p + ℓ_s];
8:       i ← i + 1;
9:    end while
10: end for
```

In next section, we shall analyze how our pier model is theoretically effective and sensitive enough for sequence search.

3.4 Sensitivity Analysis

In approximate search, if a candidate is similar to the query sequence, then the edit distance of the corresponding parts in the alignment between query and candidate is small as well. Traditional methods are based on a more restrictive conclusion of

this: if the distance between two sequences is short, then they have at least one q-length segment that is exactly the same. In BLASTn, the segment is referred to as seed, and in others, it may be referred to as q-gram, or pattern. Approaches adopted have the constraint that seeds cannot be too long, otherwise the index structure will be large and it will also lead to low sensitivity. The seed length cannot be small as well, otherwise it makes filtering ineffective. BLASTn looks for matches of w consecutive letters as seeds. In the case of BLASTn, the seed length is typically set to 11. If the edit distance is allowed to be $10\% - 20\%$ of the matched subsequence, then it is possible that BLASTn may miss some of the candidates. Gapped seed is used by PatternHunter [70] to reduce the missing candidates. The pier here is in some sense a more flexible version of gapped seeds. We use pier segment with error tolerances as seed. For example, we set the length of pier as 15, and set edit distance allowed in pier segment as 3 in our pier model.

In order to investigate the sensitivity of BLASTn and pier model theoretically, we follow the same prototype as in PatternHunter [70]. In sensitivity model, we represent an alignment between two sequences as a 0-1 string, where 1 positions represent matches, while 0 positions represent mismatches. Suppose a substring s_1 from query sequence is similar to a same-length subsequence s_2 from the data sequence. Let $a = a(s_1, s_2)$ be a 0-1 string indicating the alignment between s_1 and s_2, i.e. $a[i] = 1$ if and only if $s_1[i_1] = s_2[i_2]$, where $s_1[i_1]$, and $s_2[i_2]$ are the corresponding letters in s_1 and s_2 in terms of the position i of alignment a. So there are in total 2^L binary strings of length L: $R_0, ..., R_{2^L-1}$. The sensitivity of a seed model is computed as $\sum_{i=0}^{2^L-1}(Pr(R_i \text{ has a seed match}))$. If we count the hit probability one by one for all $2^L - 1$ binary strings, it takes exponential time. In the following, we shall discuss how to compute the sensitivity of BLASTn and pier model in polynomial time.

3.4.1 Theoretical Sensitivity Analysis for BLASTn

For BLASTn, its default seed is 11111111111, or 11 consecutive matches to generate a hit. In order to compute the exact sensitivity of BLASTn much more efficiently, we use the algorithm of dynamic programming [62, 67, 70].

Algorithm 2 Algorithm: Dynamic Programming
Input: A seed A, similarity level p and length L.
Output: The probability that A hits a p-random region of length L.
Method:

```
1: compute the compatible suffix set B;
2: for i from 0 to L do
3:    for b in B from longest to shortest do
4:       if i < |b| then
5:          f[i, b] = 0;
6:       else
7:          f_0 = f[i − |b| + |b'|, 0b'], where 0b' = B(0b);
8:          if A hits 1b then
9:             f_1 = 1;
10:         else
11:            f_1 = f[i, 1b];
12:         end if
13:         f[i, b] = (1 − p) × f_0 + p × f_1;
14:      end if
15:   end for
16: end for
17: output f[L, ϵ]
```

In PattenHunterII [67], they use the Algorithm of dynamic programming (Shown in Algorithm 2) to compute the hit probability for k seeds. Let R be a random binary string of length L with similarity level p. Each bit independently is 1 with probability p. A is a seed or a set of seeds. [2]

For a binary string b and $|b| \leq i \leq L$, $f(i,b) = Pr(A \text{ hits } R[0:i] | b \text{ is a suffix of } R[0:i])$. The hit probability of A on R is equal to $f(L, \epsilon)$, where ϵ is the empty string. For any $i > |b|$, $f(i,b) = (1-p)f(i,0b) + pf(i,1b)$. So $f(i,b)$ is computed

[2] We consider a single seed hit in this Chapter. And the BLASTn's default seed is 11111111111.

in terms of other $f(i', b')$ which has been computed earlier, and the set of b's we need to consider is limited in the process [67]. The set B is needed to be computed before Algorithm 2 is run. B is the set of binary strings that are not hit by seed A but compatible with A [3].

Therefore, by using Algorithm 2, we can compute the exact sensitivity (hit probability) of BLASTn's default seed in polynomial time.

3.4.2 Theoretical Sensitivity Analysis of the Pier Model

Based on the observation that two similar sequences would have similar subsequences, we arrive at the following property:

Property 3.4.1 *If two sequences Q and C have edit(Q, C)$\leq \zeta$, then for each segment $s \subset C$, there exists a segment $s^{'} \subset Q$ such that edit$(s, s^{'}) \leq \zeta$.*

By Property 3.4.1, we can simply index a partial set of segments of the database, and when given each query segment s', search only for segments s in the index structure that has $edit(s, s^{'}) \leq \zeta$. Further, in the above property, $edit(s, s^{'})$ is rarely near to ζ when the segment length or pier length is much smaller than the length of C or Q. Subsequently, we can state the property as follows:

Property 3.4.2 *If sequences Q and C have edit(Q, C)$\leq \zeta$, then for each $s \subset C$, there exist a segment $s^{'} \subset Q$ with high probability such that edit$(s, s^{'}) \leq \zeta^{'}$ for some $\zeta^{'} \leq \zeta$.*

Intuitively, if two sequences C and Q have very few differences between them, then the probability of these differences being clustered in the same region should be low. Instead, the differences are expected to be scattered in most cases. By losing some of the rare cases, the computation cost can be reduced dramatically.

[3] In [67], a suffix b of a region R is compatible with seed A if $b[|b|-j] = 1$ whenever $A[|A|-j] = 1$ for $0 < j \leq min(|A|, |b|)$.

Good Seeds Selection

A dilemma for a BLASTn type of search is that large seeds lose distant homologies while small ones create too many random hits [70]. BLASTn looks for matches of w (default w=11) consecutive letters as seeds. The choice of the seed length w is determined by the tradeoff between search speed and sensitivity [62]. Larger w reduces chance hits and smaller w finds weaker similarities. Compared to BLASTn's consecutive seed model, our proposed pier model uses **longer seeds with error tolerance** to have a higher probability of a hit in a similarity region. For example, in pier model, we set the length of pier as 15, and set error tolerance (edit distance) allowed in pier segment as 3. In the following, we will investigate the sensitivity of the pier model in a more theoretical way.

Let R be a random binary string of length L with similarity level p. Each bit independently is 1 with probability p. In pier model, we use pier segment of length ℓ_p with error tolerance (edit distance is θ) as a seed. We use the following formula to compute the sensitivity (hit probability) if region R contains only a single pier seed. We use A denotes the event that a pier seed is fully contained by the random region R.

$$P(A) = Pr(\ell_p, \theta, p) = \sum_{i=0}^{\theta} \begin{pmatrix} \ell_p \\ i \end{pmatrix} p^{(\ell_p - i)}(1-p)^i \tag{3.1}$$

Based on the sensitivity of single pier seed, we use inclusion-exclusion principle to compute the sensitivity of pier model when more than one pier is fully contained in region R. The inclusive-exclusive principle can be used to compute the probability of exactly k occurrences of events A_1, A_2, ..., A_n, where k is the number of pier seeds that are fully contained by the random region R, and A_i is the event that a pier seed is fully contained by the random region R. The inclusion-exclusion

principle can be used in probability where it becomes:

$$Pr(\bigcup_{i=1}^{k} A_i) = \sum_{i=1}^{k} P(A_i) - \sum_{i,j:i<j} P(A_i \bigcap A_j) + \sum_{i,j,k:i<j<k} P(A_i \bigcap A_j \bigcap A_k) - ... \pm P(\bigcap_{i=1}^{k} A_i). \tag{3.2}$$

By using inclusion-exclusion principle, we can compute the sensitivity (hit probability) of k pier seeds for a random region.

3.4.3 Comparison of Sensitivity of BLASTn and Pier Model

We studied the sensitivity of seeds used in pier model and BLASTn, and we also compared the sensitivity of pier model with BLASTn. In our pier model, piers are extracted from data sequence in terms of Property 3.3.1 instead of all the sliding segments in data sequence. For example, three piers are extracted from a region of length L=64, and the length of each pier is 15. According to the parameter setting used in [70, 67] [4], we somewhat arbitrarily chose a region of length 36 and 64 because in practice homologies are typically of size 20-200 bases. In the following study of sensitivity analysis, we use three region lengthes of 36 and 64 for both pier model and BLASTn method.

For BLASTn search method, we use its default value 11 for seed length. For pier model, we use pier segment with error tolerances as seed. We investigated the sensitivity of pier model with different parameters when the edit distance (error tolerance) θ allowed in the pier segment is set to 1, 2 or 3. The parameters of the experiments for the sensitivity analysis of pier model are summarized in Table 3.2.

We plot the figure of Similarity vs Sensitivity to investigate the sensitivity of pier model and compare it with BLASTn. In Figure 3.2 and 3.3 [5], for each similarity

[4] In PatternHunter, the authors arbitrarily chose a region length of 64.

[5] In both figures, we use (ℓ_p,k,θ) to denote pier seed for sensitivity analysis.

Parameter	Settings
length of pier: ℓ_p	11, 13, 15, 17, 18
number of piers: k	1, 2, 3, 4, 5
edit distance allowed: θ	1, 2, 3

Table 3.2: The Parameters Used for Sensitivity Analysis (Pier Model)

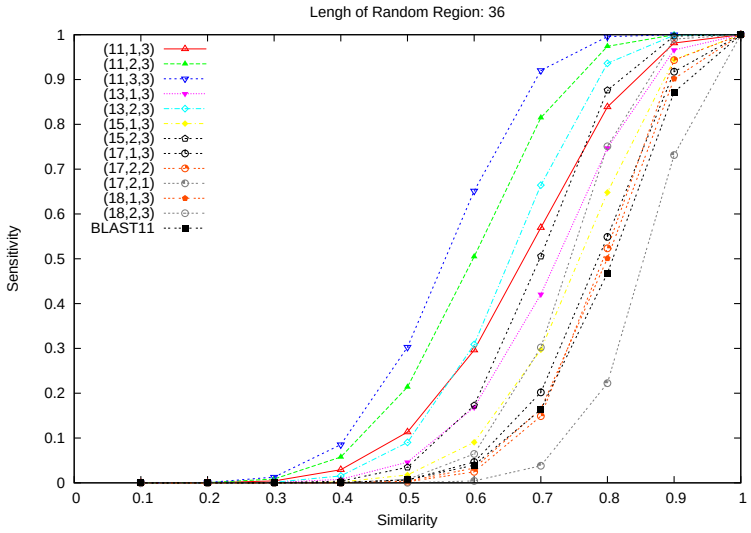

Figure 3.2: Similarity vs Sensitivity

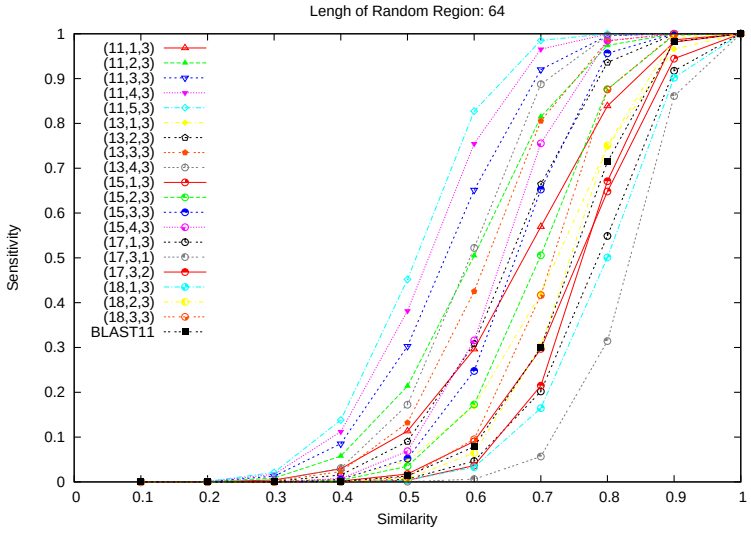

Figure 3.3: Similarity vs Sensitivity

percentage on the x-axis, the hit probabilities in the random region of length 36 and 64 are plotted on the y-axis as the sensitivity at that similarity level. We use the typical seed length 11 for BLASTn.

Figure 3.2 shows the sensitivity of random region of length 36. The results in Figure 3.2 show very clearly that pier model has much higher sensitivity than BLASTn with seed length 11 (denoted as BLAST11) for different level of similarity when we different parameter settings shown in Table 3.2. When the length of random region is set to 36, there are at most three piers of length 11 which can be fully contained in this region. Similarly, at most two piers can be fully contained when the length of pier is set to 13, 15 and 18, respectively. For example, we observed that in a random region of length 36 with 80% similarity, BLAST11 has a 46.7958% probability to get a hit, while our pier model with (15,1,3) has a 64.8162% probability to get a hit. Figure 3.3 shows the sensitivity of random region of length 64. The results show that pier model has higher probability than BLAST11 to get a hit for the seeds shown in Table 3.2, except for seeds (15,1,3), (17,1,3), (17,3,1) and (18,1,3).

In conclusion, there are three factors that affect the sensitivity of pier model. The first factor is the length of pier ℓ_p; the second one is the number of pier k; and the third one is the edit distance allowed in pier seed θ. In our model, we fix θ to 1, 2 and 3, and we observed from two above two figures that smaller ℓ_p, bigger k and bigger θ all imply higher sensitivity.

Note that in pier model, only some piers are extracted from data sequence. In comparison with BLASTn, the results show that our proposed pier model can achieve higher sensitivity even though only some extracted piers from data sequence are used to identify the hits instead of all the slidings.

3.5 The Hash-based Pier Model

After the piers are extracted from data sequences, we can just store them in a link list, or some other sequential data structure. For each pier, it will be aligned with the query patterns $q_1, q_2, ..., q_m$. If the edit distance between the pier and the query pattern is smaller or equal to θ, the pier sequence will be regarded as a candidate.

But we know that the sequential search algorithm works very slowly: the number of piers may be small, but all of them have to be accessed for each given query pattern and there are generally few hits in the whole set. The complexity of the sequential search is $O(mn\ell_p^2)$, where m is the number of query patterns and n is the number of piers. In order to find the candidates in D for the given query pattern much more efficiently, a hash-based pier model is proposed. Using a hash table constructed on the piers, we need only access the piers in the candidate buckets in the hash table, where candidate buckets refer to those buckets that have a high possibility of holding piers that are most possibly similar to the query patterns.

In the hash-based pier model, piers with length ℓ_p are first extracted from a sequence database and then hashed into the bucket of hash table $HTable$. In order to hash the piers to the pre-constructed hash table, it is necessary to encode the prefix of the pier with length λ, $\lambda \leq \ell_p$ into a 2λ bit integer. Each of the four possible nucleotides in a DNA sequence is encoded as two binary digits as follows:

$$f(a) = \begin{cases} 00, & a = \text{`}A\text{'} \\ 01, & a = \text{`}C\text{'} \\ 10, & a = \text{`}G\text{'} \\ 11, & a = \text{`}T\text{'} \end{cases}$$

Using the encoding function f, any λ-tuples of DNA sequence $s = b_1, b_2, ..., b_\lambda$ can be mapped uniquely to a 2λ bit integer by the encoding function:

$$encode(s) = \sum_{i=1}^{\lambda} 4^{i-1} f(b_i)$$

The *encode* function is compact and efficient to process, but there is no way to encode any characters apart from the four valid letters. In our implementation, invalid characters are transformed into one of the four valid ones ('A', 'C', 'G' and 'T') randomly in order to keep the positions of the matches found exactly as the positions in the original data sequence.

3.5.1 Construction of the Hash Table

After all the buckets in the hash table are initialized to empty, each pier p_i will be inserted into the corresponding bucket in the hash table using the encoding function $encode(p_i[0, \lambda-1])$ which maps the first λ-tuple of p_i into a 2λ bit integer. The hash table has a total of 4^{λ} buckets for DNA sequences.

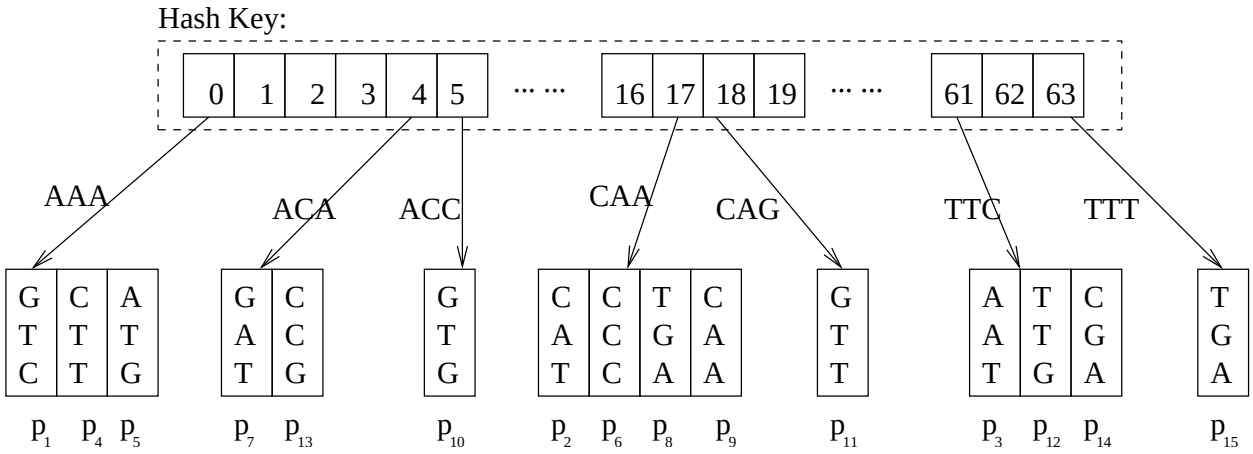

Figure 3.4: An Example of the Hash Table for Piers

The piers in the same bucket will share the same hashed prefix, i.e., the same first λ-tuple. On average, there are $|P|/4^{\lambda}$ piers in each bucket for DNA sequences,

where P is the set of piers extracted from data sequences. The algorithm for constructing the hash table on the piers is outlined in Algorithm 3[6].

Example 3.5.1 *For example in Figure 3.4,* $\ell_p = 6$, $\lambda = 3$, $P = \{p_1, p_2, \ldots, p_{15}\}$ *and the hash table has* $4^\lambda = 64$ *buckets. In this example, all the 15 piers are inserted into the hash table one by one. The piers* $p_1 : AAAGTC$, $p_4 : AAACTT$ *and* $p_5 : AAAATG$ *are inserted into the first bucket since the hash keys of these three piers are all* $encode(AAA) = 000000$.

Algorithm 3 Hash Table Generating Algorithm
Input: A set of piers $P = \{p_1, p_2, ..., p_n\}$, λ.
Output: Hash table $HTable$.
Method:

1: Initialize the HTable with size of 4^λ, with each bucket set to be empty;
2: **for** each $p_i \in P$ **do**
3: $\quad e_i \leftarrow encode(p_i[0, \lambda])$;
4: $\quad HTable[e_i] \leftarrow HTable[e_i] \cup \{p_i\}$;
5: **end for**

3.5.2 Collision Handling

To handle collision caused by the insertion of piers into the same bucket of a hash table, the trivial solution is simply to store the piers in a bucket into a link list, named *collision list*, or consolidate them into an array to save space and increase accessing efficiency since the hash table structure is relatively stable once it is constructed. If the piers in the same bucket are stored as a list, each pier in the bucket will be retrieved one by one and compared with the query pattern using dynamic programming. This will be inefficient when the number of piers in the candidate bucket is large. In our implementation, we choose λ to be 10, and

[6]In Algorithm 3, $HTable[e_i]$ stores the piers that share the same prefix which is encoded into e_i.

thus it means that there are over 10^6 buckets in the hash table. Given 10^9 piers, each bucket contains 1000 piers on average. To minimize the computation cost of obtaining candidates for sequence similarity search, we propose to use global penalty matrix (GPM) to handle the collision list of a hash table.

A $4^{2\omega}$ GPM, where $\omega = \ell_p - \lambda$, is built beforehand. All the possible segments in a DNA sequence with length ω are mapped into 2ω bit integers with the use of the encoding function. We compute the edit distance for each pair of $\langle i, j \rangle$, where $0 \leq i, j < 4^{\omega}$, and store the computed value into cell $\langle i, j \rangle$ in GPM. Table 3.3 illustrates the structure of GPM by an example for $\omega = 2$ in DNA sequences. Note that by carefully and systematically eliminating symmetric cases, we can reduce the size of the table dramatically.

When the piers in a candidate bucket need to be retrieved and checked during query processing, the edit distance of a pier and the query pattern does not have to be computed through dynamic programming. Instead, only the GPM is looked up to see whether the pier suffix segments and query pattern suffix are within a small edit distance ε. If it holds, we will say that the current pier p_i may be a candidate of the query pattern q_j since we know that the current pier and the query pattern share the similar prefix with length ℓ_p. By using the pre-computed GPM, the dynamic computation cost of verification can be reduced from $O(\ell_p^2)$ to $O(1)$.

3.6 Query Processing

In this section, we shall show how the hash-based pier model can be used to conduct efficient and effective similarity search in a biological sequence database. The space and time complexity are also discussed and analyzed. In our approach, the

	AA	AC	AG	AT	CA	CC	CG	CT	GA	GC	GG	GT	TA	TC	TG	TT
AA	0	1	1	1	1	2	2	2	1	2	2	2	1	2	2	2
AC	1	0	1	1	2	1	2	2	2	1	2	2	2	1	2	2
AG	1	1	0	1	2	2	1	2	2	2	1	2	2	2	1	2
AT	1	1	1	0	2	2	2	1	2	2	2	1	2	2	2	1
CA	1	2	2	2	0	1	1	1	1	2	2	2	1	2	2	2
CC	2	1	2	2	1	0	1	1	2	1	2	2	2	1	2	2
CG	2	2	1	2	1	1	0	1	2	2	1	2	2	2	1	2
CT	2	2	2	1	1	1	1	0	2	2	2	1	2	2	2	1
GA	1	2	2	2	1	2	2	2	0	1	1	1	1	2	2	2
GC	2	1	2	2	2	1	2	2	1	0	1	1	2	1	2	2
GG	2	2	1	2	2	2	1	2	1	1	0	1	2	2	1	2
GT	2	2	2	1	2	2	2	1	1	1	1	0	2	2	2	1
TA	1	2	2	2	1	2	2	2	1	2	2	2	0	1	1	1
TC	2	1	2	2	2	1	2	2	2	1	2	2	1	0	1	1
TG	2	2	1	2	2	2	1	2	2	2	1	2	1	1	0	1
TT	2	2	2	1	2	2	2	1	2	2	2	1	1	1	1	0

Table 3.3: An Example of the Global Penalty Matrix, $\omega = 2$

algorithm of sequence similarity search based on the hash-based pier model consists of three steps: generating the query pattern with size of ℓ_p from Q; searching for pier candidates among the hashed piers; and post-processing the candidates to concatenate adjacent candidates to form final alignments with a high alignment score. We focus our discussion on the second step since it is the main part of query processing.

3.6.1 Neighborhood Enumeration

Our search technique partitions the given query sequence Q into $|Q| - \ell_p + 1$ query patterns $q_1, q_2, ..., q_{|Q|-\ell_p+1}$. In the second step, the pier prefix segment with length λ, $q_i[0, \lambda - 1]$ is encoded to a hash key h_i, which is a 2λ bit integer. Then all the encoded neighbors $ngbr$ of this hash key h_i are enumerated, and the neighbors are those 2λ bit integers which are within a small edit distance from h_i. In our algorithm, for simplicity, λ is set to 10 or 12, and the edit distance allowed for

neighbors is set to 2. Our method for enumerating all neighbors within an edit distance of 2 from the given encoded query pattern is supported by Theorem 3.6.1. Note that each neighbor of the hash key can be enumerated in $O(1)$ amortized cost.

Theorem 3.6.1 Let S and Q be two sequences of length λ from the alphabet set Σ. If $edit(S, Q) \leq 2$, then one of the following cases is true:

- Case 1: $edit(S, Q) = 0$, i.e. the two sequences are exactly the same;
- Case 2: $edit(S, Q) = 1$, i.e. one replacement operation is needed in S to transform S to Q;
- Case 3: $edit(S, Q) = 2$, there will be three subcases to explain it:
  - Case 3.1 two replacement operations in S are needed to transform S to Q;
  - Case 3.2 one insertion and one deletion in S with order are needed to transform S to Q;
  - Case 3.3 one deletion and one insertion in S with order are required to transform S to Q.

We shall illustrate in detail how the neighbors of the given encoded sequence q of length λ are generated.

Proof:

1. Case 1 means that the neighbor of q is q itself.
2. In Case 2, the neighbor of q is enumerated with the replacement of the letter x in position i, $0 \leq i < \lambda$ with other letters in Σ. Each neighbor $ngbr$ enumerated in terms of Case 2 meets $edit(q, ngbr)$=1.

3. In Case 3.1, the neighbor is generated with the replacement of the letter x in position i, $0 \leq i < (\lambda - 1)$ with other letters except x, followed by the replacement of the letter y in position j, $(i + 1) \leq j < \lambda$ with other letters except y.

4. In Case 3.2, the neighbors are enumerated with the insertion of any letter in Σ in position $0 \leq i < (\lambda - 1)$ of q and the deletion of the letter in position j of q, $(i + 1) \leq j < \lambda$.

5. Similarly, in Case 3.3, the neighbors are generated with the deletion of a letter in position i, $0 \leq i < \lambda$ and the insertion of any letter in Σ in position j, $(i + 1) \leq j < \lambda$ of q.

6. Each neighbor *ngbr* enumerated in terms of Case 3.1, Case 3.2 and Case 3.3 meets $edit(q, ngbr)$=2.

□

Based on Theorem 3.6.1, the encoded neighboring 2λ bit integers are enumerated. There may exist several kinds of redundancy in neighbor enumeration, which means that the same neighbor may possibly be enumerated several times across the different cases in Theorem 3.6.1, or the neighbor generated may be the sequence itself.

Most of the redundancy can be avoided easily with little additional cost. For those redundancy that cannot be detected easily, we will exempt it by labeling those neighbors that have been enumerated already.

3.6.2 Sequence Similarity Search

The algorithm of sequence similarity search using the proposed hash-based pier model is presented in Algorithm 4. Once an encoded neighbor *ngbr* of the hash

key of query pattern q_i is enumerated, the piers in the bucket of the hash structure $HTable[ngbr]$ will be retrieved and checked to see if they are candidates, i.e., whether they are similar to the query pattern q_i by the $verify$ function. The $verify$ function is implemented using the GPM we mentioned in earlier section.

In order to find the local alignment between pier segment and query pattern, for a pier candidate, we allow the edit distance in its prefix and suffix, respectively. For its prefix, we use the edit distance threshold to enumerate neighbors of query pattern; for its suffix we use edit distant to compute the GPM beforehand. The two edit distances cannot be simply added up as the edit distance between pier p and query pattern q_i. In order to ensure a high sensitivity, suppose that the edit distance for enumerating neighbors is β and the one for the GPM is ε. By careful setting of β and ε, we can capture most cases of $edit(p, q_i) \leq \theta$. In our next experimental study, we discover the performance of query processing by setting β as 1 and 2, setting ε as 1, 2 and 3.

Algorithm 4 Hash-based Similarity Search Algorithm
Input: Hash table $HTable$, Query sequence Q, ℓ_p, λ.
Output: $Candidate$.
Method:

```
1: Candidate ← ∅;
2: for each query pattern q_j in Q do
3:    Enumerate the next neighbor ngbr of encode(q_j[0, λ]);
4:    for each pier p in the bucket HTable[ngbr] do
5:       verify(p, q_j);
6:       if p and q_j are similar then
7:          Candidate ← Candidate ∪ {⟨p, q_j⟩};
8:       end if
9:    end for
10:   Until all the neighbors of q_j are enumerated;
11: end for
```

3.6.3 Time and Space Complexity

We next look at the time and space complexity for our algorithm. First, for pier set P construction, we need to scan the database once; the time complexity for this is $O(|D|)$. A more effective way is to simply read the piers that we need since we can obtain the start position of each pier on the fly. To build a hash table for the piers, each pier can be inserted with time complexity $O(1)$ if the GPM is used. So the total time complexity for the construction of the hash table will be $O(|P|)$. For the space complexity of the hash table, we need $O(4^\lambda)$ for the table head. For the bucket of the table, each pier will contribute space $\Theta(\omega)$. Thus, the total size of the buckets will require space $O(\omega|P|)$. Also, we need space $\Theta(4^{2\omega})$ for the GPM. Thus, the total space complexity for the hash structure will be $O(4^\lambda + 4^{2\omega} + \omega|P|)$. Typically, if we set $\lambda = 10$ and $\omega = 5$, for a sequence with 3×10^9 letters, with span length $\ell_s = 7$, then the hash table size will be less than 200 mega-bytes. The structure is small enough to keep in the main memory to aid faster computation.

For each query pattern q of the query Q, the set of piers which are N neighbors of q will be enumerated. As we have explained, each neighboring pier can be generated with time amortized complexity $O(1)$. For each collision list we access, each item of the collision list will require time $O(1)$ when the GPM is used. Thus, the time complexity for the query is $O(\alpha|Q||N|)$, with the loading factor $\alpha = |P|/4^\lambda$. We can take the repetitive computation into consideration, such as in symmetric cases, the letter repetition in a sequence and the neighboring of the several query patterns, to reduce time complexity.

3.7 Experiments

In this section, we present the experimental results on the performance of homology search in DNA sequence databases to evaluate the efficiency and effectiveness of the proposed hash-based pier models. We investigate the impact of different parameter setting on performance of hash-based pier model. Meanwhile, we also compare the proposed search model to the latest version of BLAST (NCBI BLAST 2) for the efficiency and effectiveness of pre-processing and query processing.

3.7.1 Datasets

All the DNA sequence databases used in the experiment are real datasets downloaded from NCBI website. In the experimental study, all the six DNA sequence datasets: ecoli.nt (4.68MB), yeast.nt (12.3MB), human_est.fa (23.6MB), month.gss (286.2MB), patnt (702.1MB), other_genome (1.06GB) and other_genome+patnt (1.76GB), shown in Table 3.4, are used to evaluate the query performance of our proposed method.

The databases are composed from the characters 'A', 'C', 'G', 'T', 'N' with 'N' representing a wildcard, i.e. 'N' can be any one of the four characters. The character 'N' is randomly regarded as 'A', 'C', 'G', 'T'.

We have collected a query set from human genomic sequence database to evaluate the responses from the hash-based pier approach and BLASTn. The query sequences are segments randomly extracted from the human genome databases, and the reported performance results are averaged over 10 queries. Based on Property 3.3.1, the shortest length of queries allowed for both sets is set to be 100. The shortest query length is 100 and the longest is 2000 in our experiments.

Database	Size	No. of Sequences
ecoli.nt	4.68MB	400
yeast	12.3MB	16
human_est.fa	23.6MB	40000
month.gss	286.2MB	293031
patnt	702.1MB	1125356
other_genome	1.06GB	2638
other_genome + *patnt*	1.76GB	1127994

Table 3.4: The DNA Sequence Databases

3.7.2 Experimental Settings

This performance analysis contains two sets of experiments to investigate the performance of the proposed hash-based pier model. We first investigate the effect of parameter settings on the performance of the hash-based pier model. The six parameters for our hash-based pier model are ℓ_p, ℓ_s, λ, $\omega = \ell_s - \lambda$, θ, β. Subsequently we compare the performance of the hash-based pier model with BLAST11. Since the comparison of sensitivity analysis has been conducted in Section 3.4, here we shall focus on comparing the efficiency and accuracy of query processing between the hash-based pier model and BLAST11. The parameters used in the experiments are listed in Table 3.5. The experiments are conducted in C++ programming language, and are executed on a Linux server with 4 UltraSPARC-III+ CPU of 900MHz and 8GB free memory.

Parameter	Settings
length of pier: ℓ_p	11, 12, 13, 14, **15**, 16, 17, **18**
length of span: ℓ_s	0, **5**, 10, 15, 20, 25, 30
length of prefix: $\lambda(=\ell_p\text{-}\omega)$	5, 6, 7, 8, 9, 10, 11, 12, 13
length of suffix: ω	2, 3, 4, **5**, 6
θ	1, 2, **3**
β	0, **1**, **2**

Table 3.5: The Parameter Settings

3.7.3 Effect of Parameters

We study the effect of the parameters of pier length, prefix length, suffix length, span length and error tolerance on the performance of the hash-based pier model respectively. To do this, we construct the hash structure on dataset *month.gss* and perform similarity search for query length of 100 on *month.gss*. The size of dataset *month.gss* is 286.2MB as listed in Table 3.4, the number of sequence in *month.gss* is 293031. The efficiency of dataset preprocessing, efficiency of pier candidate (seed) detection, efficiency of candidate extension, and number of alignments returned are investigated and analyzed for the different parameter settings listed in Table 3.5. We use (ℓ_p, ℓ_s, λ, ω, θ, β) to denote the parameter setting for the hash-based pier model. In the following experiments, the default values for different parameters are high-lighted in bold in Table 3.5.

Effect of Pier Length ℓ_p and Prefix Length λ

We first study the performance of hash-based pier model with different pier length and prefix length while we fix the span length, suffix length, error tolerances θ and β to the default values, respectively. Table 3.6 shows the response time of preprecessing, response time of seed detection and extension, and the number of final alignments when pier length varies from 11 to 18. We use longer seed with error tolerances in the hash-based pier model compared to BLASTn with seed length 11. Since the size of hash structure is too big when the pier length is greater than 18, we therefore vary pier length from 11 to 18 to investigate the effect of pier length and prefix length on the performance of search.

As expected, the results in Table 3.6 show that the response time of preprocessing increases as both the length of pier (ℓ_p) and the length of its prefix (λ) increase. It takes more time to construct the hash structure for the bigger ℓ_p

Parameter Settings	**Pre-processing time (Sec)**	**Seed/Candidate Detection (Sec)**	**Seed/Candidate Extension (Sec)**	**No. of Alignments**
(11,5,6,5,3,2)	14.38	15.45	13.02	15766392
(12,5,7,5,3,2)	14.19	7.33	8.52	836337
(13,5,8,5,3,2)	14.8	2.46	4.45	256011
(14,5,9,5,3,2)	14.53	0.8	1.81	77659
(15,5,10,5,3,2)	15.9	0.27	0.64	23001
(16,5,11,5,3,2)	22.16	0.12	0.22	6891
(17,5,12,5,3,2)	31.51	0.08	0.06	2026
(18,5,13,5,3,2)	70.62	0.07	0.02	575

Table 3.6: Effect of Pier Length ℓ_p and Prefix Length λ (ω=5; Dataset:month.gss)

and λ because the size of the constructed hash structure becomes larger for longer pier length and prefix length. From Table 3.6, we also observe that the response time of both candidate detection and extension decrease rapidly, as the length of pier increases. Similarly, the number of final alignments between data sequences and query sequence decreases when ℓ_p and λ increase. This is because much more pier alignments are returned for smaller ℓ_p and λ with the same error tolerances θ=3 and β=2. For example, the number of alignments is 15,766,392 for ℓ_p=11, λ=6, θ=3, β=2; while the number of alignments is 575 for ℓ_p=18, λ=13, θ=3, β=2. The result is consistent with the one we observed from theoretical sensitivity analysis in Section 3.4. Longer piers lose distant homologies while short ones create too many random hits.

Effect of Suffix Length ω

We measure the effect of suffix length ω on the performance of hash-based pier model. Table 3.7 shows the response time of pre-precessing, response time of both seed detection and extension, and the number of final alignments for different value of span length when we fix the length of pier length to 15. Note that there are two portions for the response time of pre-processing: response time for constructing the

hash structure based on prefix length, and response time for initializing the GPM based on suffix length. When the pier length ℓ_p is fixed, decreasing suffix length will increase prefix length. When suffix length is small, the time of constructing the hash table dominates the whole response time of pre-processing. As shown in Table 3.7, we observed that the response time of pre-processing increases with the suffix length decreases from 5 to 2. We also investigate the effect of suffix length on the performance of the search when pier length is set to 18 in Table 3.8. The result shows that the pre-processing with (λ=12, ω=6) is faster than that with (λ=13, ω=5) and (λ=11, ω=7). It indicates that for a larger suffix length ω, the time incurred for initializing the GPM dominates the total response time of pre-processing.

Parameter Settings	**Pre-processing time (Sec)**	**Seed/Candidate Detection (Sec)**	**Seed/Candidate Extension (Sec)**	**No. of Alignments**
(15,5,10,5,3,2)	15.9	0.27	0.64	23001
(15,5,11,4,3,2)	22.03	0.12	0.6	20279
(15,5,12,3,3,2)	31.99	0.08	0.52	17236
(15,5,13,2,3,2)	71.57	0.07	0.41	13314

Table 3.7: Effect of Suffix Length ω (ℓ_p=15; Dataset:month.gss)

Parameter Settings	**Pre-processing time (Sec)**	**Seed/Candidate Detection (Sec)**	**Seed/Candidate Extension (Sec)**	**No. of Alignments**
(18,5,13,5,3,2)	70.62	0.07	0.02	575
(18,5,12,6,3,2)	45.72	0.08	0.03	642
(18,5,11,7,3,2)	362.76	0.12	0.02	716

Table 3.8: Effect of Suffix Length ω (ℓ_p=18; Dataset:month.gss)

In Table 3.7 and 3.8, we also observed that both the response time of candidate detection and the response time of extension increase as the length of suffix increases for both $\ell_p = 15$ and ℓ_p=18. With the same error tolerances of θ=3 and β=2, more alignments (or hits) can be found by decreasing the pier length ℓ_p and prefix length

λ; however, this also increases the response time of query processing: candidate detection and extension. More alignments lead to higher computation cost, and therefore slower speed.

Effect of Span Length ℓ_s

The effect of span length on the performance of hash-based pier model is summarized in Tables 3.9 and 3.10. The results demonstrate that the response time of pre-processing decreases with the span length increases from 0 to 20 for both ℓ_p=15 and 18. Meanwhile, the response time of candidate detection and extension also decreases when the span length increases. The reason is that more piers are extracted from the sequence dataset for smaller span length, and consequently more piers are indexed and searched in hash-based pier model. We also observed that the performance only changes slowly (not dramatically) when the span length is varied from 0 to 20.

Parameter Settings	**Pre-processing time (Sec)**	**Seed/Candidate Detection (Sec)**	**Seed/Candidate Extension (Sec)**	**No. of Alignments**
(15,0,12,3,3,2)	38.27	0.09	0.67	22976
(15,5,12,3,3,2)	31.99	0.08	0.52	17236
(15,10,12,3,3,2)	28.47	0.08	0.45	14068
(15,15,12,3,3,2)	25.96	0.07	0.37	11608
(15,20,12,3,3,2)	24.34	0.06	0.33	10044

Table 3.9: Effect of Span Length ω (ℓ_p=15; Dataset:month.gss)

Effect of Error Tolerances θ and β

Since we use longer seed with error tolerance in our hash-based pier model, we further study the performance of the query processing based on our proposed model using different values of error tolerances. The response time of candidate detection and extension for different values of θ and β is shown in Table 3.11 when pier

length ℓ_p is set to 18. The number of alignments (or hits) found by our model is also depicted. As we have introduced, β is the error tolerance for the prefix of pier segment; while $\varepsilon(=\theta\text{-}\beta)$ is the error tolerance for the suffix of pier segment. The results in Table 3.11 show that both the response times of candidate detection and candidate extension decrease rapidly when error tolerance β decreases. The same results can be observed when error tolerance θ decreases.

Correspondingly, as expected, the number of alignments or hits reduces when θ and β decrease. For example, when we use the default values ℓ_p=18, ℓ_s=5, λ=13 and ω=5, the number of alignments is 3556 for $\beta = 2$ and θ=4; while the number is decreased to 24 for β=0 and θ=4. The bigger β, the more neighbors of the query pattern are enumerated; consequently, more buckets of the hash structure are required to be checked to determine if the piers in the buckets are really similar to the query pattern. Thus this also increases the response time of candidate detection and extension. When we fix the error tolerance β, the number of alignments decreases when θ decreases. This is obvious since GPM is used to check if the edit distance of pier suffix and query suffix is within ε. The results in Table 3.11 also confirm that more candidates are returned for bigger ε.

3.7.4 Comparison of Hash-based Pier Model and BLAST11

In this subsection, we compare the performance of hash-based pier model and BLAST11. First, we explore the efficiency of pre-processing of both hash-based pier model and BLAST11. Next we conduct the experiments to investigate the effect of the length of the query sequence and the size of dateset on the performance of hash-based pier model in comparison to BLAST11. Finally, we discuss the results of alignments returned by hash-based pier model and BLAST11.

Efficiency in Pre-processing

We perform an experiment to evaluate the efficiency of data sequence pre-processing for sequence similarity search. We conduct the pre-precessing for hash-based pier model by using four sets of parameter settings. We use $(\ell_p, \ell_s, \lambda)$ to denote the hash-based pier model with different parameter settings. Figure 3.5 shows that pre-processing with our proposed hash-based pier model is always faster than with BLAST11 for different parameter settings. This is because our hash-based pier model simply extracts longer piers and hashes them rather than processes shorter segments in the sequence database as in BLAST11 and other methods [96, 70, 25]. For the hash-based pier model with different parameter settings $(\ell_p, \ell_s, \lambda)$, we achieve 2-10 times speedup compared to BLAST11 in the stage of pre-processing for all the seven sequence datasets. The result also shows that our method scales up sublinearly with respect to the size of DNA sequence dataset. From Figure 3.5, we also observe that there is a spike on the query processing of BLAST11, and it is possibly due to the characteristic of the dataset *other_genome* which containing many long sequences.

Varying Query Length

The next experiment is to investigate the impact of length of the query sequence on the performance of our method in comparison to BLAST11. For this experiment, we perform similarity search for query lengths of 100, 300, 500, 800, 1000, 1500 and 2000 on the datasets *month.gss* and *patnt*. For hash-based pier model, we conduct the similarity search with different parameter settings. In the following experiments, we use $(\ell_p,\ell_s,\lambda,\omega,\theta,\beta)$ to denote the parameter setting for the hash-based pier model. We use three parameter settings for hash-based pier model: (15,5,10,5,3,1), (16,5,11,5,3,1) and (17,5,12,5,3,1) to study the performance of sim-

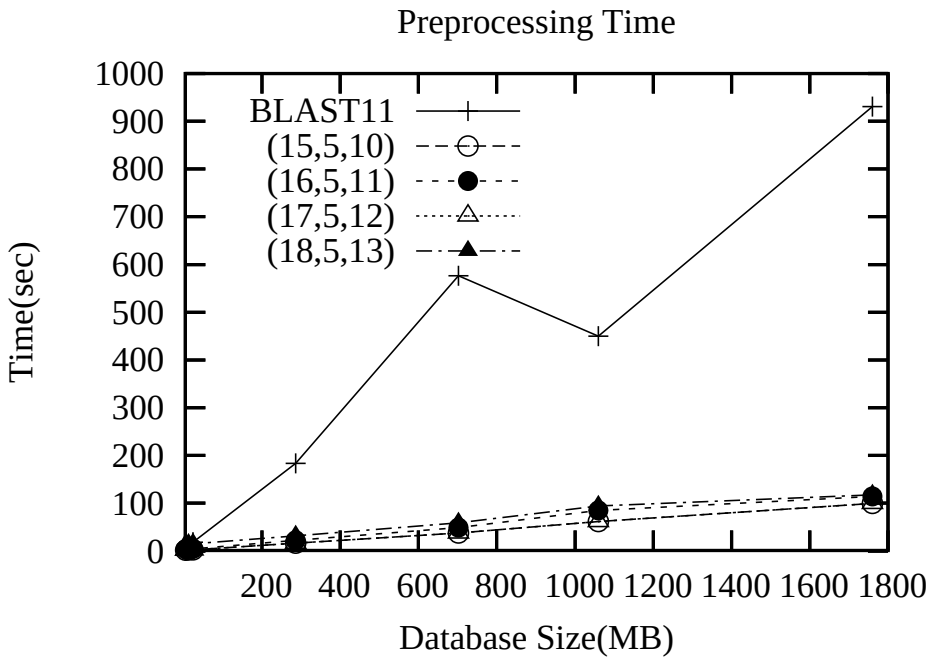

Figure 3.5: Pre-processing Time

ilarity search when query length is varied since the pier model with the three parameter settings can achieve as high as or much higher sensitivity than BLAST11. Figures 3.6 and 3.7 show that search speed of pier model is 2-15 times faster. The margin is widened as the query length increases. They also confirm that our method with the parameter setting of (17,5,12,5,3,1) is faster than our method with the other two settings, named (15,5,10,5,3,1) and (16,5,11,5,3,1) as we observed in the section of effect of parameters in Section 3.7.3, advocating that the search time decreases as the length of pier increases.

To investigate the effectiveness of both hash-based pier model and Blast11, in Table 3.12, we also plot the number of alignments that both methods can return. We observe that for both *month.gss* and *patnt* datasets, the hash-based pier model with the three parameter settings returns much more alignments than BLAST11. This is due to the seed we used in hash-based pier model. We use longer pier

Parameter Settings	**Pre-processing time (Sec)**	**Seed/Candidate Detection (Sec)**	**Seed/Candidate Extension (Sec)**	**No. of Alignments**
(18,0,13,5,3,2)	75.53	0.08	0.03	732
(18,5,13,5,3,2)	70.62	0.07	0.02	575
(18,10,13,5,3,2)	67.62	0.07	0.02	463
(18,15,13,5,3,2)	65.59	0.06	0.02	392
(18,20,13,5,3,2)	64.02	0.07	0.02	362

Table 3.10: Effect of Span Length ω (ℓ_p=18; Dataset:month.gss)

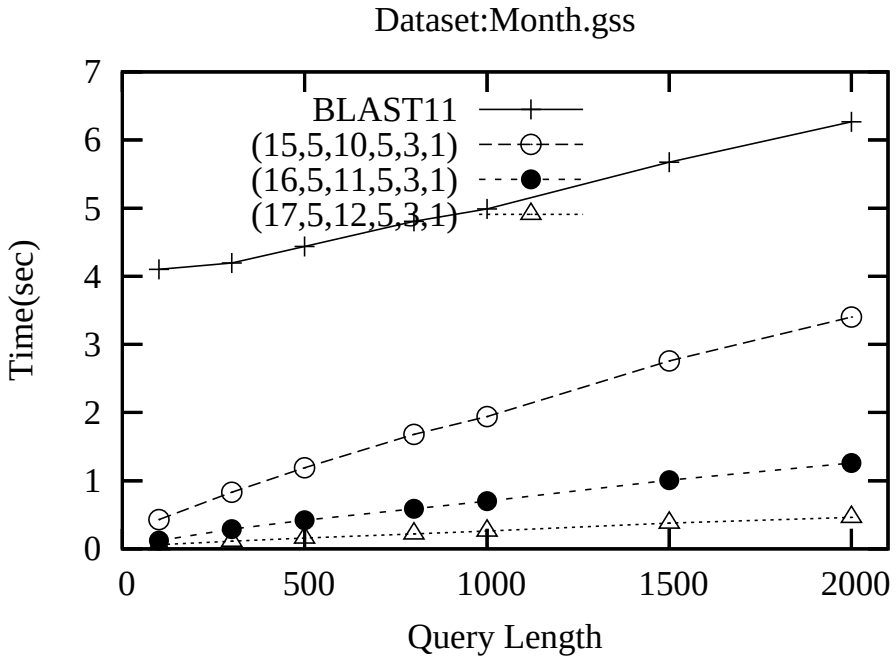

Figure 3.6: Query Time (Dataset:month.gss)

Parameter Settings	**Seed/Candidate Detection (Sec)**	**Seed/Candidate Extension (Sec)**	**No. of Alignments**
(18,5,13,5,4,2)	0.07	0.12	3556
(18,5,13,5,4,1)	0.01	0.02	520
(18,5,13,5,4,0)	0.02	0.01	24
(18,5,13,5,3,2)	0.06	0.03	576
(18,5,13,5,3,1)	0.01	0.02	169
(18,5,13,5,3,0)	0.01	0.01	14
(18,5,13,5,2,2)	0.07	0.02	55
(18,5,13,5,2,1)	0.01	0.01	23
(18,5,13,5,2,0)	0.01	0.01	4
(18,5,13,5,1,1)	0.01	0.01	2
(18,5,13,5,1,0)	0.01	0	1

Table 3.11: Effect of Error Tolerances θ and β (ℓ_p=18; Dataset:month.gss)

with error tolerance in our pier model. Consequently, we are able to detect more alignments and similar region between the query sequence and data sequence compared to BLAST11. Furthermore, in terms of both the response time and the number of returned alignments for similarity search, we will use the parameter setting (17,5,12,5,3,1) in the following experiments for comparing the performance of both methods with the effect of dataset size.

Query Length	**BLAST11**	**(15,5,10,5,3,1)**	**(16,5,11,5,3,1)**	**(17,5,12,5,3,1)**
100	21	9878	2703	792
300	9	26648	7261	1976
500	19	44338	12259	3357
800	39	72507	1999	5538
1000	12	89977	24838	6907
1500	19	143019	39472	10813
2000	27	189725	52289	14357

Table 3.12: Alignments Found (Dataset:month.gss)

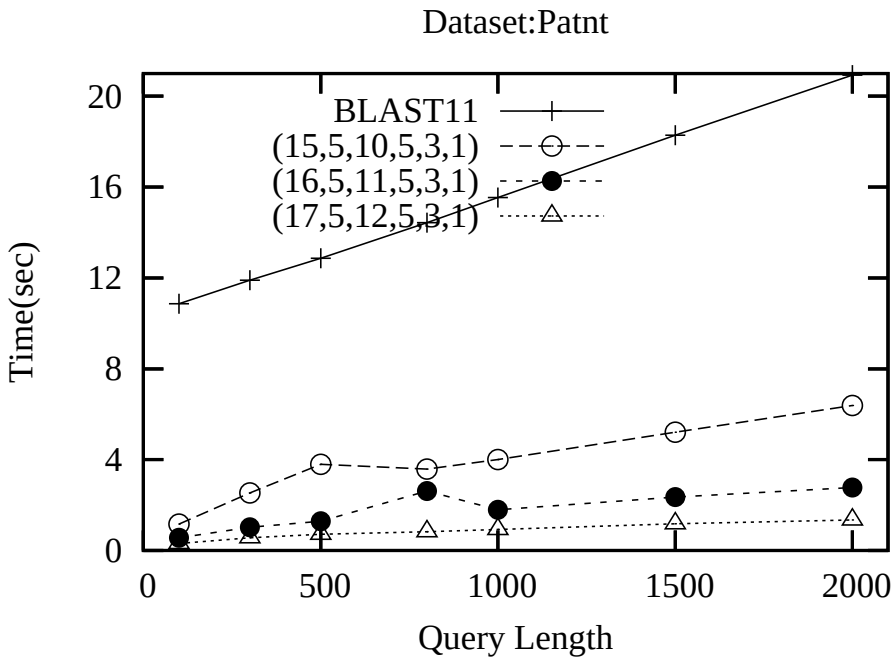

Figure 3.7: Query Time (Dataset:patnt)

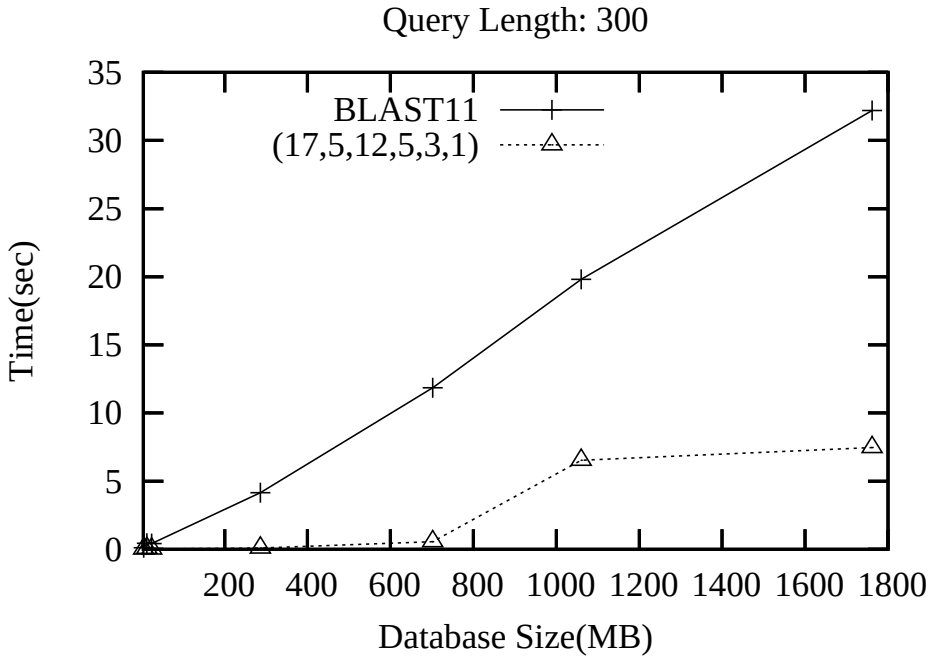

Figure 3.8: Query Time ($|Q| = 300$)

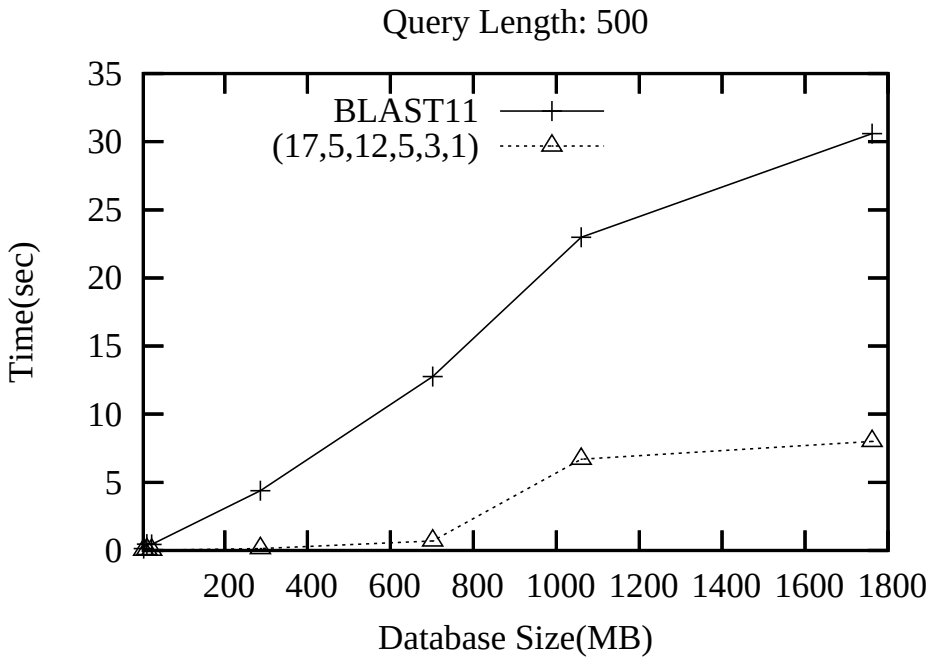

Figure 3.9: Query Time ($|Q| = 500$)

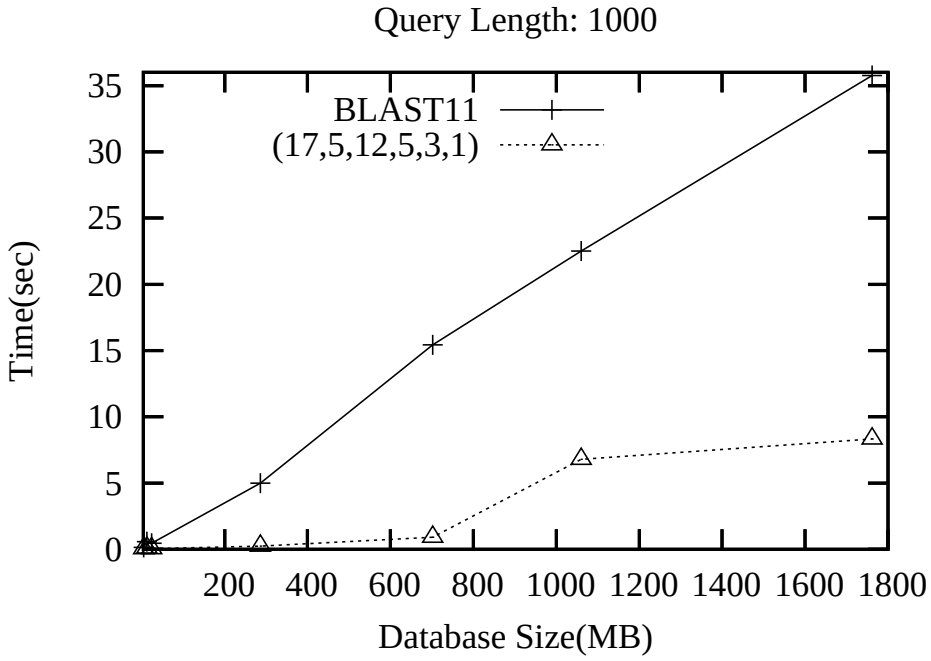

Figure 3.10: Query Time ($|Q| = 1000$)

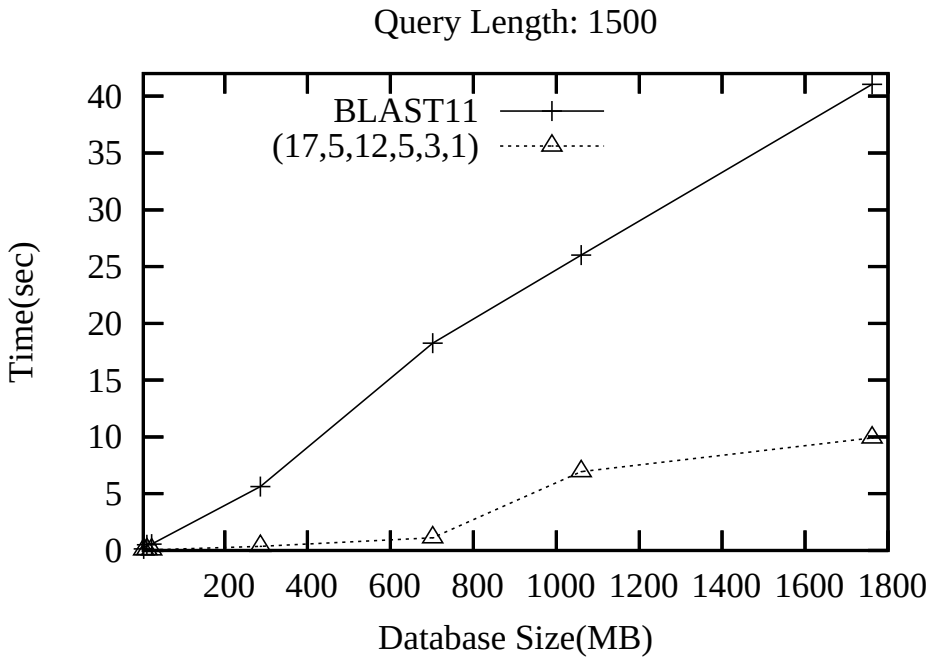

Figure 3.11: Query Time ($|Q| = 1500$)

Varying Dataset Size

For further evaluation of the efficiency of our method, we run query processing with query length of 300, 500, 1000 and 1500 on six datasets. The experiment evaluates the effect of database size on the search time of both hash-based pier model and BLAST11. As shown in Figures 3.8, 3.9, 3.10 and 3.11, our method with parameter setting (17,5,12,5,3,1) outperforms BLAST11 2-10 times when the size of datasets varies from 4.68MB to 1.76GB. The results exemplify that our method is very efficient for handling sequence similarity search in large DNA sequence databases.

Discussion on Returned Alignments (Hits)

Recently proposed methods such as [70, 67, 63, 24, 21] have reported that modifications of the basic approach proposed by BLAST [8, 9] lead to significant improvement in both sensitivity and running time of similarity search in DNA sequence

databases. Sensitivity can no doubt be increased by decreasing the length of the seed. However, this would increase the number of alignments returned (may decrease accuracy of results), and also the running time [21]. So researchers are looking for a better seed model to achieve both high accuracy and low running time. We propose to use a longer seed ("pier") with error tolerances to achieve both higher sensitivity and efficiency.

Query Length	**BLAST11**	**(17,5,12,5,1,1)**
100	21	8
300	9	27
500	19	43
800	39	65
1000	12	80
1500	19	129
2000	27	165

Table 3.13: Alignments Returned (Dataset:month.gss)

We investigate the number of returned alignments for the hash-based pier model with parameter (17,5,12,5,1,1) and BLAST11 when varying the length of query from 100 to 2000 on the dataset *month.gss*. Compared to using (17,5,12,5,3,1), the hash-based pier model using (17,5,12,5,1,1) can return comparable number of local alignment with BLAST11, while it is much faster than (17,5,12,5,3,1) and BLAST11. The numbers of returned alignments by both our hash-based pier model and BLAST11 are listed in Table 3.13. We observed that the pier model with (17,5,12,5,1,1) can return more local alignments than BLAST11.

Note that in this experiment, we generated the query sequence of length 100, 300, 500, 800, 1000, 1500 and 2000 from a single longer sequence. This means that the query sequence of length 100 is a subsequence of query sequence of length 300, and so forth. We make an interesting observation that for BLAST11, the number of returned alignments does not increase with the length of query sequence. It is

Data Sequence ID	Sequence Position	Data Segment	Query Position	Query Segment
162854	550-568	AGAGTGTTTGAAAACTGA	0-18	AGAGTGTTTCAAAACTGC
185360	22-40	GAATGCAAAAATCACAAG	50-68	GAATGCAAACATCACAAA
111133	748-767	CAAAGAAGTTTCTAAAAAG	64-83	CAAAGAAGTTTCTGAAAAT
24291	483-502	TCAAAGATGTTTCTGAAAG	63-82	ACAAAGAAGTTTCTGAAAA
129674	528-546	GAAGTTGCTGAAAATGCA	68-86	GAAGTTTCTGAAAATGCT
118605	197-216	AAACTTTCTGAAAATGCTG	68-87	GAAGTTTCTGAAAATGCTT
66538	550-568	TCTGAAAATCCTTCTGTG	74-92	TCTGAAAATGCTTCTGTC
86779	395-413	CTGACAATGCTTCTGTCT	75-93	CTGAAAATGCTTCTGTCT

Table 3.14: Eight Local Alignments (Dataset:month.gss, Query Length:100)

surprising that the number of returned alignment for query length of 300 is less than the number for query length of 100. For the hash-based pier model, as expected, we observed that the 8 returned local alignments for query length 100 are also part of the 28 returned results for query of length 300. After close examination, we also discover that we can find the local alignments which are really very similar, but cannot be detected by BLAST11. In Table 3.14, we show the 8 local alignments returned by the hash-based pier model with (17,5,12,5,1,1) for the query length of 100 on the dataset *month.gss*. In the next section, we will further examine investigate the results returned by both hash-based pier model and BLAST11 by using Smith-Waterman algorithm [108].

3.7.5 Search Accuracy Analysis

To compare and analyze the accuracy of the results of hash-based pier model and BLAST11, we define two search accuracy metrics: precision and recall as follow:

$$Precision = \frac{Number\ of\ pairs\ of\ similar\ sequences\ returned}{Number\ of\ total\ pairs\ of\ sequences\ returned}$$

$$Recall = \frac{Number\ of\ pairs\ of\ similar\ sequences\ returned}{Number\ of\ total\ pairs\ of\ similar\ sequences}$$

In the following experiments, we use 400,00 sequences in human EST dataset as data sequences; we randomly select 20 sequences from mouse EST dataset as query sequences. To obtain the total matching sequences, we performed all-against-all comparison on the data sequences and query sequences. The all-against-all comparison was conducted with the SSEARCH [93] implementation of the Smith-Waterman algorithm [108]. SSEARCH is a subprogram in the FASTA software, which can return the total pairs of similar sequences between the data sequences and query sequence. As stated in [67], the recall of SSEARCH is regarded as 100%.

In our experiments, for each randomly selected query sequence, we use SSEARCH to generate the top 1000 alignments between the query sequence and sequences in human EST dataset. We observed that the minimum score of all the alignments is 67, and the maximum score of all the alignments is 1660.

We next run hash-based pier model and BLAST11 on the same dataset and with the same query sequences. For a query sequence, if a sequence pair returned by the search method is also returned by SSEARCH, we regard the sequence pair as a similar sequence pair or a real alignment. That is, we compute the number of pairs of similar sequences returned by each search method for each query sequence, and their precision and recall based on SSEARCH results.

Table 3.15 shows the precision and recall of the search results for the hash-based pier model and BLAST11 for 20 randomly selected queries. For the hash-based pier model, we use two parameter settings (15,5,10,5,3,1) and (17,5,12,5,3,1) [7]. In Table 3.15, we observe that BLAST11 can achieve high precision and low recall, while hash-based pier model with parameter setting (15,5,10,5,3,1) can achieve high recall and low precision. And we also observe that hash-based pier model with parameter setting (17,5,12,5,3,1) has higher recall than BLAST11, and higher precision than

[7] We use (ℓ_p,ℓ_s,λ,ω,θ,β) to denote the parameter setting for the hash-based pier model.

QID	Pier(15,5,10,5,3,1)		Pier(17,5,12,5,3,1)		BLAST11	
	Precision	**Recall**	**Precision**	**Recall**	**Precision**	**Recall**
1	0.043507	0.134576	0.055556	0.014493	0.428571	0.012
2	0.121159	0.139113	0.26158	0.096774	0.895833	0.172
3	0.102703	0.019095	0.333333	0.003015	1	0.008
4	0.055982	0.073663	0.054264	0.007064	0.384615	0.005
5	0.069104	0.206827	0.144172	0.047189	0.1	0.001
6	0.041562	0.033199	0.034483	0.002012	0.358974	0.014
7	0.080276	0.151454	0.077519	0.01003	0.2	0.002
8	0.060278	0.039674	0.393939	0.013225	0.785714	0.011
9	0.051958	0.06599	0.147727	0.013198	0.363636	0.008
10	0.049052	0.119598	0.056034	0.013065	0.090909	0.001
11	0.039792	0.069556	0.057851	0.007056	0.375	0.003
12	0.337413	0.193387	0.649573	0.152305	1	0.202
13	0.054036	0.081818	0.070313	0.009091	0.478873	0.034
14	0.075862	0.022066	0.210526	0.004012	1	0.005
15	0.095926	0.07322	0.085714	0.006018	0.291667	0.007
16	**0.098661**	**0.449187**	**0.150746**	**0.410569**	**0.484**	**0.121**
17	0.099073	0.139418	0.289308	0.046138	0.815385	0.053
18	0.038486	0.123858	0.040541	0.012183	0.032258	0.001
19	0.122449	0.116923	0.3125	0.030769	0.324324	0.012
20	0.079077	0.161323	0.059701	0.008016	0.142857	0.002

Table 3.15: Precision and Recall of the Results (Dataset:human_est.fa, 20 Queries Randomly Selected from mouse_est.fa)

hash-based pier model with parameter setting (15,5,10,5,3,1). For example, for the query sequence of QID=16, the recalls of hash-based pier model with parameter settings (15,5,10,5,3,1) and (17,5,12,5,3,1) are 0.449187 and 0.410569 respectively, but the recall of BLAST11 is only 0.121 which are lower than hash-based pier model. In addition, we note that the corresponding precision for BLAST11 is 0.484, which is higher than the hash-based pier model with parameter settings (15,5,10,5,3,1) and (17,5,12,5,3,1). The results are consistent with the results of theoretical sensitivity analysis we observed in Section 3.4.

We also give an overall evaluation on the average precision and recall for both BLAST11 and hash-based pier model in Figure 3.12. The average precision and

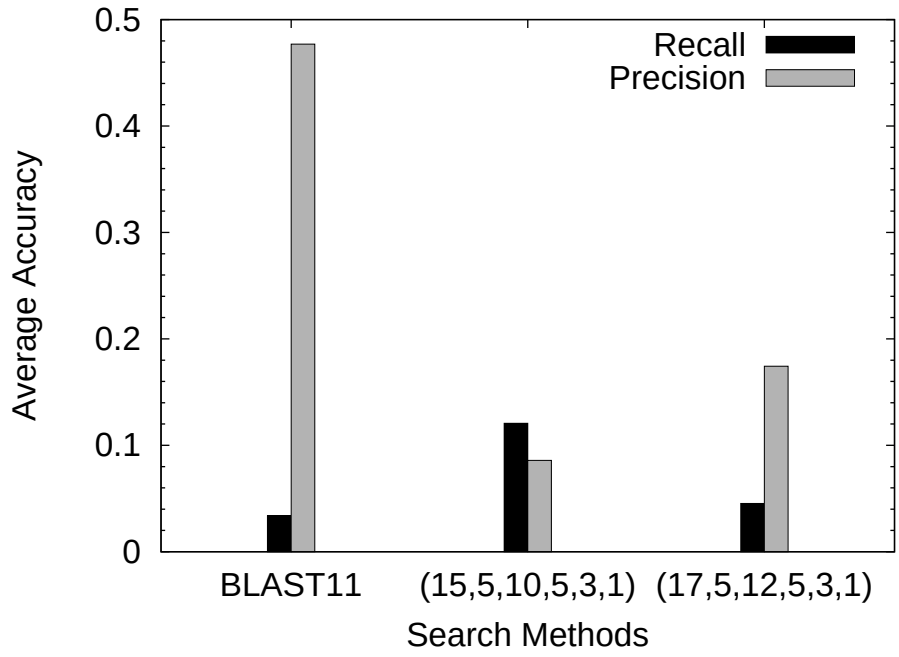

Figure 3.12: Average Accuracy (Dataset:human_est.fa, 20 Queries Randomly Selected from mouse_est.fa)

recall are computed based on the precisions and recalls (listed in Table 3.15) of the 20 randomly selected queries. The average precision of BLAST11 is 0.477, which is higher than that of hash-based pier method; the average recall of BLAST11 is 0.337, which is lower than that of the hash-based pier model [8]. The results shown in Figure 3.12 confirm that the hash-based pier model achieves higher recall than BLAST11, while sacrificing some precision.

3.8 Summary

There is a growing requirement for the efficient searching of genomic database since the current genomic databases are growing rapidly and they are queried about ten of thousands of times per day. In this chapter, for efficient similarity search, we

[8]The average recalls of the hash-based pier model with parameter settings (15,5,10,5,3,1) and (17,5,12,5,3,1) are 0.121 and 0.4531, respectively.

have proposed a new model, the hash-based pier model, and demonstrated its search efficiency, sensitivity, effectiveness and accuracy.

Before performing the similarity search in DNA sequence database, the piers are extracted from data sequences, and inserted into a hash table. Based on the hash table, the similarity search is done efficiently. We also proposed a method – the GPM (Global Penalty Matrix)-based method – to further improve search efficiency by reducing the computation cost of candidate verification. Compared to the most widely used biological database search tool, BLAST11, our method is faster, yet requiring smaller memory and space. Further, though it sacrifices some precision, hash-base pier model achieves better recall than BLAST11 .

CHAPTER 4

Indexing DNA Sequences Using q-grams

4.1 Introduction

We observed in Chapter 3 that our proposed hash-based pier model can efficiently detect the similar regions between data sequences and query sequence with high recall, but sacrifices some precision compared to BLAST11. Since we know that longer seeds lose distant homologies while small ones create too many random hits [70], we propose to use seeds with length greater than 17 [1] to improve the precision of the search. In this chapter, we present a novel method for indexing long seeds efficiently based on q-grams to facilitate similarity search in a DNA sequence database and sidestep the need for linear scan of the entire sequence database.

Our method is based on the observation that two sequences share a certain number of q-grams if the edit distance between them is within a certain threshold.

[1] We used seed length of 17 in Chapter 3, and we will use seed length of 30 in this chapter.

Moreover, since there are only four characters in the DNA alphabet, we know that the number of all combinations of q-grams in a DNA sequence is 4^q.

Based on the q-grams in DNA sequence, we propose a two level index to prune data sequences that are far away from the query sequence. The disjoint segments with the length ω are generated from the data sequence. At the first level of the indexing structure, a cluster (called qCluster) of similar q-grams in DNA sequence is generated; then a typical hash table is built on the segments with respect to the qCluster. At the second level indexing structure, the DNA segments are transformed into the c-signature based on their q-grams; then a new index called the *c-signature trees* (c-trees) is proposed to organize the c-signatures of all segments of the DNA sequences for search efficiency. The c-trees are dynamic trees built on the c-signatures as a technique to avoid the linear scan of the entire database.

At the first level of search, the ω-length sliding segment of query sequence is generated and encoded into the key in terms of the proposed coding function, and then the neighbors of this key will be enumerated. Thus a set of candidate segments of the data sequences will be extracted from the buckets pointed by the key and its neighbors, and inserted into the second index structure c-trees for subsequent filtering. At the second level of search, we only access the tree paths in c-trees that include possible similar data sequences in their leaf nodes with respect to the partial distance between the corresponding c-signature substrings of query segment and data segment. We also propose a similarity search algorithm based on the c-trees for query segments.

The contributions of this chapter can be summarized as follows:

- The proposed c-signature provides a compact and approximate representation of DNA sequence data in terms of its q-grams.
- A two-level filtering approach is proposed based on qCluster and c-signature.

- We present a new two-level index structure for c-signatures and qClusters, and further propose and describe an efficient search method based on the two-level index.

The rest of this chapter is organized as follows. In Section 2, we give a brief introduction of the problem definition, which has been described in Chapter 2. In Section 3, the concept of qClusters and c-signature is presented. The filtering principle based on q-grams is also described. In Section 4, we propose a two-level index scheme constructed on the q-grams for DNA sequences. In Section 5, an efficient similarity search algorithm is presented based on the proposed two-level index structure. The test data and experimental results are presented in Section 6. Section 7 summarizes the contributions of this chapter.

4.2 Problem Definition

Following the definition of similarity search problem in a DNA sequence database given in Chapter 3, we propose in this chapter a novel method for indexing the DNA sequences efficiently based on q-grams to facilitate similarity search in a DNA sequence database. The notations used throughout the chapter are summarized in Table 4.1.

Property 4.2.1 *S and Q are two sequences with the same length $L = |S| = |Q|$. S is divided into p disjoint ω-length segments $s_1, s_2, \ldots, s_p$, and then there exists an ω-length segment $q \in Q$ and $s_i \in S$ such that if $edit(S, Q) \leq \zeta$, then $edit(s_i, q) \leq \lfloor \zeta/p \rfloor$.*

With respect to Property 4.2.1, the region in data sequence which is similar to the query sequence can be detected by computing the edit distance between the disjoint segment s_i in S and the sliding query segment q in Q.

Notation	Description
D	a DNA sequence database D
$\|D\|$	the size of DNA sequence database D
S	the data sequence in D
$\|S\|$	the length of sequence S
$S[i:j]$	the subsequence from position i to j
$edit(S,P)$	edit distance of sequence S and P
ω	the length of segment
θ	distance threshold for data segments and query segment
$sig^1(S)$	q-gram signature of sequence S
$sig^c(S)$	c-signature of sequence S
$sig^c(S)[i,j]$	c-signature string from position i to j in $sig^c(S)$
$SDist(sig^c(S_1), sig^c(S_2))$	signature distance between $sig^c(S_1)$ and $sig^c(S_2)$
δ	number of the c-trees
T_i	the ith c-tree
ℓ	height of c-tree T_i, $0 \leq i < \delta$

Table 4.1: Notation Description

Since with high possibility there exists a similar segment pair $(s,q), s \in S, q \in Q'$ if S is similar to Q', we instead solve the following problem:

Problem 4.2.1 *Given the length ω and edit distance θ, find all the disjoint segments s_i with length ω generated from $\mathcal{D}$ which meet $edit(s_i, q_j) \leq \theta$ for the query segments q_j with length ω in Q.*

4.3 Preliminaries

Although the edit distance is a simple but fairly accurate measure of the evolutionary proximity of two DNA sequences, the computation complexity is $O(mn)$, m and n being the length of the two sequences. To speed up approximate sequence matching, filtering is an efficient way to quickly discard parts of a sequence database, leaving the remaining parts to be checked by the edit distance. Our proposed approach to sequence similarity search is based on q-grams, where the q-gram

similarity is used as a filter for similarity sequence search in DNA sequences.

4.3.1 The q-gram

Before we define *qClusters* and *c-signature*, we shall briefly review the definition of q-gram and the principle of q-gram based filtering. The intuition behind the use of q-grams as a filter for approximate sequence search is that two sequences would have a large number of q-grams in common when the edit distance between them is within a certain number.

Definition 4.3.1 **q-gram of Sequence**

Given a sequence S, its q-grams are obtained by sliding a window of length q over the characters of S. For a sequence S, there are $|S| - q + 1$ q-grams in it.

The q-gram based filtering introduced by Jokinen and Ukkonen [60] in 1991 is given as follows:

Lemma 4.3.1 Filtering based on q-grams (Jokinen and Ukkonen [60])

Let an occurrence of $Q[1 : w]$ with at most θ edit or hamming distance end at position j in sequence data S. Then at least $w + 1 - (\theta + 1)q$ of the q-grams in $Q[1 : w]$ occur in the subsequence $S[j - w + 1 : j]$. In the other words, there are at most θq q-grams in $Q[1 : w]$ which do not occur in $S[j - w + 1 : j]$, and vice versa. So obviously, the number of different q-grams between $Q[1 : w]$ and $S[j - w + 1 : j]$ is at most $2\theta q$.

Lemma 4.3.1 uses the number of common q-grams in two sequences as filter. Conversely, we can also use the number of dissimilar q-grams in two sequences as the filter and propose another lemma. Theoretically, the new lemma can therefore be deduced from Lemma 4.3.1 as below:

Lemma 4.3.2 Filtering Based on Different q-grams

Let an occurrence of $Q[1:w]$ with at most θ edit or hamming distance end at position j in data sequence S. Then we have the following property: $|(G_{Q[1:w]} \cup G_{S[j-w+1:j]}) - (G_{Q[1:w]} \cap G_{S[j-w+1:j]})| \leq 2\theta q$. G_S denotes the set of q-grams occurring in sequence S.

Proof: *Lemma 4.3.1 gives a necessary condition for $S[j-w+1:j]$, a subsequence of S to be a candidate for an approximate match with $Q[1:w]$: At least $w+1-(\theta+1)q$ of the q-grams are contained in $S[j-w+1:j]$. Hence, there exist at most $(w-q+1-(w-(\theta+1)q+1) = \theta q$ q-grams in $Q[1:w]$ which are different from the q-grams in $S[j-w+1:j]$. Therefore, the number of all the different q-grams in $Q[1:w]$ from the q-grams $S[j-w+1:j]$ is at most $2\theta q$.*

4.3.2 The qClusters and c-signature

The alphabet of the DNA sequence comprises four characters: $\Sigma = \{A, C, G, T\}$. It means there are in all $|\Sigma|^q = 4^q$ kinds of q-grams, and we may arrange them according to the lexicographic order, and use r_i to denote the ith q-gram in this order. All the possible q-grams are denoted as: $\Re = \{r_0, r_1, \ldots, r_{4^q-1}\}$. The q-gram clusters (qClusters) can be defined below:

Definition 4.3.2 q-gram Clusters (qClusters)

All the possible q-grams, $\Re = \{r_0, r_1, \ldots, r_{4^q-1}\}$ are divided into λ clusters (denoted as $qClusters$) $\{qCluster_1, \ldots, qCluster_\lambda\}$ by a certain principle. In this work, we simply cluster the m continuous q-grams $\{r_{(i-1)m}, \ldots, r_{im-1}\}$ together into $qCluster_i$, $1 \leq i \leq \lambda = \lceil \frac{4^q}{m} \rceil$.

The q-gram signature and c-signature of the DNA sequence are defined as follows:

Definition 4.3.3 ***q*-gram Signature**

The q-gram signature is a bitmap with 4^q *dimensions where ith bit corresponds to the presence or absence of the q-gram* r_i*. For a given sequence S, the ith bit is set as '1' if* $r_i \in \Re$ *occurs at least once in sequence S, else it is set as '0'.*

For brevity, we use 1-signature interchangeably with q-gram signature in our discussion. Using the definition of 1-signature, each DNA segment can be mapped into a 4^q-dimensional 1-signature vector, and we can therefore transform the segments of the DNA sequences in the database into their corresponding 1-signatures.

Definition 4.3.4 ***c*-signature**

Let $sig^1(S)=(a_0, \ldots, a_{n-1})$ *be a q-gram signature of the DNA segment S with* $n=4^q$*, then its c-signature is defined as:* $sig^c(S) = (u_0, \ldots, u_{k-1})$ *where* $k=\lceil n/c \rceil$*, and* $u_i = \sum_{j=ic}^{(i+1)c-1} a_j$*. Set* $a_j = 0$ *when* $n \leq j < ck$*. For sequence S and P, we define the distance between* $sig^c(S) = (u_0, \ldots, u_{k-1})$ *and* $sig^c(P) = (v_0, \ldots, v_{k-1})$ *as* $SDist(sig^c(S), sig^c(P))=\sum_{i=0}^{k-1} |u_i - v_i|$.

Clearly, we can generate c-signatures with different granularity according to the definition of the c-signature. For better understanding of the definition of q-gram signature and c-signature, we consider the following example:

Example 4.3.1 *For a DNA sequence P="ACGGTACT", its q-gram signature is* (<u>01</u> <u>00</u> <u>00</u> <u>11</u> <u>00</u> <u>11</u> <u>10</u> <u>00</u>) *with 16(=*4^2*) dimensions when* $q = 2$*. In sequence P, the q-gram 'AC' occurs twice in position 0 and 5 respectively, so we set the corresponding bit in position 1 in q-gram signature as '1'. As there is no occurrence of the q-gram 'AA' in sequence P, the corresponding bit in position 0 in q-gram signature is set as '0'. For c=2, the c-signature of sequence P, shown in Figure 4.1, is* (10020210) *with respect to the definition of the c-signature.*

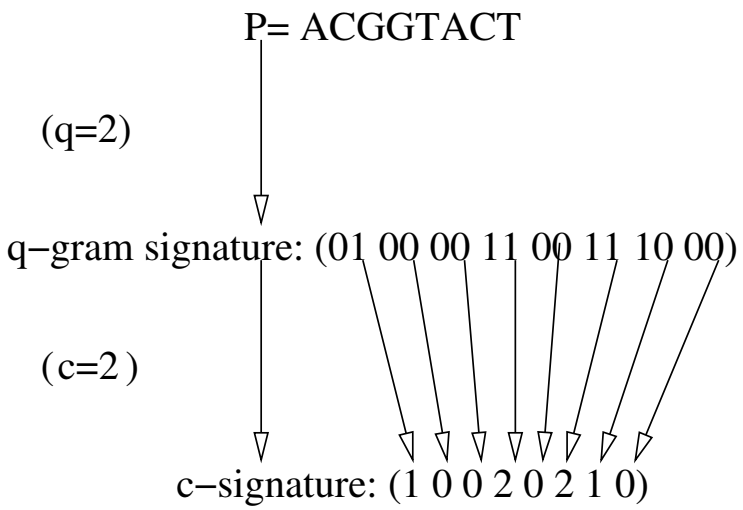

Figure 4.1: The c-signature of DNA Sequence P

With the property $|a| + |b| \geq |a + b|$, it is not difficult to obtain the following lemma for filtering in terms of c-signature:

Lemma 4.3.3 Filter Based on c-signatures

Given a sequence S, there is at most θ edit or hamming distance from another sequence P with $|S| = |P|$. Let $sig^1(S) = (a_0, a_1, \ldots, a_{n-1})$ and $sig^1(P) = (b_0, b_1, \ldots, b_{n-1})$ be the q-gram signatures generated for sequence S and P respectively. Denote the c-signatures of S and P as $sig^c(S) = (u_0, u_1, \ldots, u_{k-1})$ and $sig^c(P) = (v_0, v_1, \ldots, v_{k-1})$, $c > 1$, respectively. Then $\sum_{i=0}^{k-1} |u_i - v_i| \leq \sum_{j=0}^{n-1} |a_j - b_j| \leq 2\theta q$.

Proof: *In term of Lemma 4.3.1 and the definition of q-gram signature, $\sum_{i=0}^{n-1} |a_i - b_i| \leq 2\theta q$ holds. According to the definition of c-signature, $u_i = \sum_{j=ic}^{(i+1)c-1} a_j$ and $v_i = \sum_{j=ic}^{(i+1)c-1} b_j$. The following formula holds: $\sum_{i=0}^{k-1} |u_i - v_i| = \sum_{i=0}^{k-1} |\sum_{j=ic}^{(i+1)c-1} a_j - \sum_{j=ic}^{(i+1)c-1} b_j| \leq \sum_{i=0}^{k-1} \sum_{j=ic}^{(i+1)c-1} |a_j - b_j| = \sum_{j=0}^{kc-1} |a_j - b_j| = \sum_{j=0}^{n-1} |a_j - b_j| \leq 2\theta q$.*

4.4 An Indexing Scheme for DNA Sequences

In this section, we present a two-level indexing scheme to organize the segments in DNA sequence database and support the similarity search.

4.4.1 The Hash Table

In order to hash the DNA segments to a hash table with size 2^λ, it is necessary to encode the segment into a λ-bit integer. Given a segment s, we encode it into a λ bitmap $e = (e_1, e_2, \ldots, e_\lambda)$ with respect to qClusters=$\{qCluster_1, \ldots, qCluster_\lambda\}$. If there exists a q-gram $gram$ in s which meets $gram \in qCluster_i$, we set $e_i = 1$, else $e_i = 0$, where $1 \leq i \leq \lambda$. Following the encoding principle, any DNA segment s can be encoded into a λ-bit integer $(e_1, \ldots, e_\lambda)$ by the coding function:

$$coding(s) = \sum_{i=1}^{\lambda} 2^{i-1} e_i$$

The hash table has a total of 2^λ buckets for the qClusters $\{qCluster_1, \ldots, qCluster_\lambda\}$, and each segment s_i can be inserted into the corresponding bucket in the hash table with the use of the hash function $coding(s_i)$. Note that λ is set as 11 and 24 for q=3 and 4 respectively in the experimental studies to make the hash structure fit in memory, and furthermore, get the better performance on search, and we will not declare it again.

4.4.2 The c-trees

The c-trees are a group of rooted dynamic trees built for indexing c-signatures. The height of the trees, ℓ is set by the users. Given the c-signature of the segment s, $sig^c(s)$= $(v_0, v_1, \ldots, v_{k-1})$, there are $\delta = \lceil \frac{k}{\ell} \rceil$ trees in total. We denote these trees

as $T_0, T_1, \ldots, T_{\delta-1}$. Each path from the root to a leaf in T_i corresponds to the c-signature string $sig_i^c(s) = (v_{i\ell}, v_{i\ell+1}, \ldots, v_{(i+1)\ell-1})$ in the c-signature of sequence s. For ease of discussion, we shall assume without loss of generality that k is divisible by ℓ and thus $T_{\delta-1}$ also has a height of ℓ. For each internal node of the tree, there are at most $c+1$ children. Each edge in a tree of c-trees is labeled with the respective value from 0 to c.

The DNA segments are transformed into the c-signatures in order to build the c-trees on them. Note that it is not necessary to store the c-signatures themselves after the trees are constructed. To have a better understanding of the definition of c-trees, we shall present a straightforward algorithm to build c-trees for a group of c-signatures in Algorithm 5.

In Algorithm 5, $label[\langle N_x, N_y \rangle]$ denotes the label of edge $\langle N_x, N_y \rangle$ in the c-trees. For notation convenience, we define $S - S'$ as a suffix of S, where S' is a prefix of S, and the concatenation of S' and $S - S'$ is S. ϵ is used to refer an empty string. Also $lNode$ denotes the leaf node in the c-trees, and $E_0[lNode]$ is a group of segments in $lNode$ of the first tree T_0. Note that $E_0[*]$ will be constructed only for the tree T_0. For the other trees, the link from the c-signature to the leaf node will be constructed instead.

Given a group of c-signatures $\{sig^c(s_0), sig^c(s_1), \ldots, sig^c(s_m)\}$, the set of c-signature strings for the first c-tree T_0 is $\{sig_0^c(s_0), sig_0^c(s_1), \ldots, sig_0^c(s_m)\}$. We will show how the first tree T_0 can be constructed.

The tree T_0 is initialized with a root node T_0. In the algorithm, first the c-signature string $sig_0^c(s_0)$ is inserted into the tree. According to the function TreeInsert$(T_0, sig_0^c(s_0), s_0)$, since there are no other paths in the tree T_0, a new leaf node N_0 is created with the path labeled with $sig_0^c(s_0)$ from T_0 to N_0 under the root node T_0, and s_0 is added into $E_0[N_0]$.

Algorithm 5 Tree Construction
Input: c-signatures $sig^c(s_0), \ldots, sig^c(s_{|\mathcal{D}|-\omega})$.
Output: c-trees $(T_0, T_1, \ldots, T_{\delta-1})$.
Method:

1: $T_i \leftarrow NULL$, $0 \leq i < \delta$
2: **for** each c-signature $sig^c(s_j)$ **do**
3: **for** $i \leftarrow 0 \ldots \delta - 1$ **do**
4: TreeInsert(T_i, $sig_i^c(s_j)$, s_j)
5: **end for**
6: **end for**
7:
8: Function TreeInsert(N_x, sig, s)
9: **if** $sig = \epsilon$ **then**
10: insert(N_x, s, i) /*N_x is the leaf node*/
11: **else if** there exists an edge $\langle N_x, N_y \rangle$ where $label[\langle N_x, N_y \rangle]$ is a prefix of sig **then**
12: TreeInsert(N_y, $sig - label[\langle N_x, N_y \rangle]$, s)
13: **else if** there exists an edge $\langle N_x, N_y \rangle$ where $label[\langle N_x, N_y \rangle]$ shares a longest prefix pf with sig, $pf \neq \epsilon$ **then**
14: split $\langle N_x, N_y \rangle$ into two parts with a new node N_z, such that $pf = label[\langle N_x, N_z \rangle]$
15: create a new leaf $lNode$ with edge label $sig\text{-}label[\langle N_x, N_z \rangle]$ under N_z
16: insert($lNode$, s, i)
17: **else**
18: create a new leaf node $lNode$ under N_x with edge label $label[\langle N_x, lNode \rangle]$=$sig$
19: insert($lNode$, s, i)
20: **end if**
21:
22: Function insert($lNode$, s, i)
23: **if** i=0 **then**
24: $E_0[lNode] \leftarrow E_0[lNode] \cup \{s\}$
25: **else**
26: build the link from c-signature of s to $lNode$ in T_i
27: **end if**

For the following c-signature strings $sig_0^c(s_i)$, $1 \leq i \leq m$, they will be inserted into the growing tree one by one. For each $sig_0^c(t_i)$, the function TreeInsert($T_0, sig_0^c(s_i), s_i$) from Line 9 to 20 in Algorithm 5 will be executed recursively. The following case will be executed **in order**.

1. If there exists a path out of the root T_0 whose label is exactly equal to $sig_0^c(s_i)$, add s_i to the set $E_0[lNode]$, where $lNode$ is the leaf node of this path.

2. If there exists an edge or a path $\langle T_0, N_y\rangle$ with longest length which is the prefix of $sig_0^c(s_i)$, the function TreeInsert$(N_y, sig_0^c(s_i) - label[\langle T_0, N_y\rangle], s_i)$ will be recursively executed until s_i is put into $lNode$ of a path in T_0.

3. For the case that there is no above path or edge in the tree T_0, we will check if there is a path $\langle T_0, N_y\rangle$ in T_0 which shares the longest prefix $pf \neq \epsilon$ with $sig_0^c(s_i)$. If such a path exists, $\langle T_0, N_y\rangle$ will be split into two parts with a new node N_z such that pf=$\langle T_0, N_z\rangle$. One edge is labeled by pf, the other edge labeled with $sig_0^c(s_i)$-pf is created under the node N_z. A new leaf node $lNode$ is also created under the second edge and s_i is added to $E_0[lNode]$ as well.

4. If $sig_0^c(s_i)$ shares nothing with the edge or path in T_0, then a new path labeled with it will be created with $E_0[lNode] = \{s_i\}$, where $lNode$ is the new created leaf node.

The c-signature strings $sig_i^c(s)$ are inserted into the growing trees T_i $1 \leq i < \delta$ one by one by executing the function TreeInsert$(T_i, sig_i^c(s), s)$ recursively. We now demonstrate the c-trees construction constructed for DNA segments with the following example.

Example 4.4.1 *Consider the five DNA segments s_0="ACGGT", s_1="CTTAG", s_2="ACGTT", s_3="TAAGC" and s_4="GACGT". When we set q=2 and c=2, their c-signatures are: $sig^2(s_0)$=(1001 0200), $sig^2(s_1)$=(0101 0011), $sig^2(s_2)$=(1001 0101), $sig^2(s_3)$=(1100 1010), $sig^2(s_4)$=(1001 1100). If $\ell = 4$, we get $\frac{4^q}{c\ell}$=2 trees. The first tree T_0 is constructed from the c-signature strings $sig_0^2(s_i)$, $0 \leq i \leq 4$, and*

the tree T_1 is constructed from $sig_1^2(s_i)$, $0 \leq i \leq 4$. The c-trees (T_0, T_1) for the five DNA segments are shown in Figure 4.2.

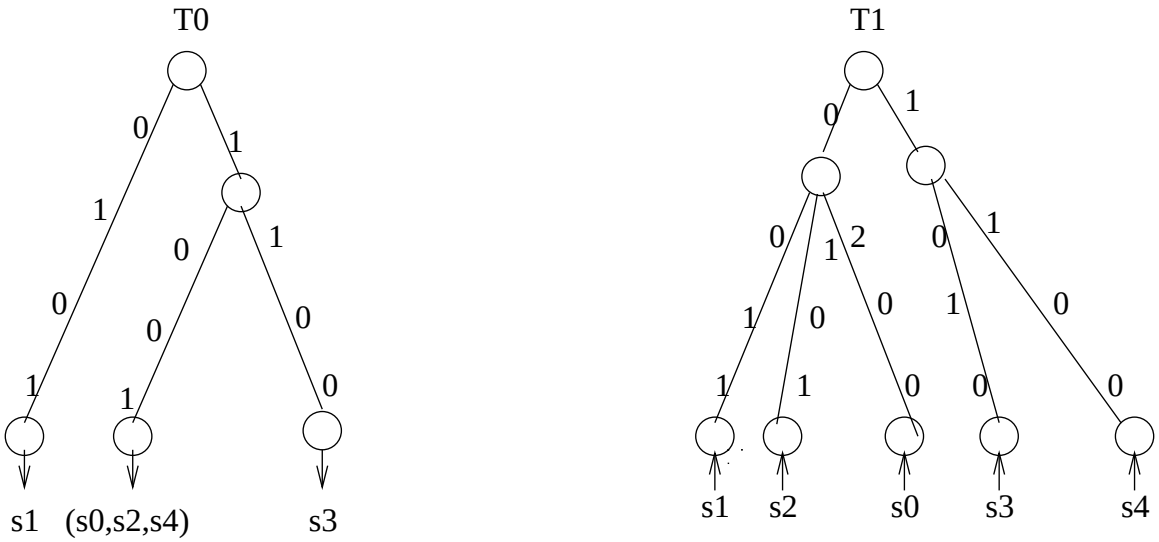

Figure 4.2: The c-trees for the DNA segments

An example is given below in detail about the tree T_0 construction.

Example 4.4.2 *In this example, the first half of the five 2-signatures are $sig_0^2(s_0) = 1001$, $sig_0^2(s_1) = 0101$, $sig_0^2(s_2) = 1001$, $sig_0^2(s_3) = 1100$ and $sig_0^2(s_4) = 1001$. The construction of the tree T_0 can be described as follows:*

1. *When the c-signature string 1001 for s_0 is to be inserted, T_0 is empty, so an edge with label 1001 is created under the root node of T_0;*

2. *When the c-signature string 0101 for s_1 is to be inserted, T_0 has one edge 1001 which shares no prefix with 0101. An edge with label 0101 is inserted under the root node of T_0;*

3. *For the third c-signature string 1001 of s_2, an identical edge (path) label with it is found in c-tree T_0. So the segment identifier s_2 is simply inserted under the existing leaf node with label 1001;*

4. *For the fourth c-signature string 1100 of s_3, since it shares a prefix 1 with an existing edge labeled with 1001, the edge 1001 is split into 1 and 001 at a new created internal node, and a new edge 100 is created and inserted under the new created internal node;*

5. *For the last c-signature string 1001 of s_4, as it is just the same as the first and third c-signature string for s_0 and s_2 respectively, the identifier of segment s_4 is simply inserted under the existing leaf node with label 1001.*

4.5 Query Processing

In this section, we present a two step filter-and-refine algorithm in performing the similarity search on DNA segments. In the first step, the two-level index is used to identify potential candidates by pruning data segments that are far away from the query sequence. In the second step, the dynamic programming is conducted to obtain the final alignments with high alignment score between the candidates and query sequence. Similar to BLAST, these candidates are then extended on the left and right to determine if it is contained within a longer segment pair whose score is greater than or equal to a certain threshold. The algorithm stops extending the seed in one direction when that causes the score to fall a certain distance below the best score found so far. As we have discussed in Chapter 2, the seed searching time dominates the total response time for BLAST. Database read time and seed extension time are just a small portion of the total response time. Therefore, in the following, we mainly consider how we can obtain these candidates which are similar to query segments by pruning those dissimilar data segments. The query processing in this work can be described as follows:

1. Preprocess the DNA sequence dataset and query sequence;

- Partition query sequence Q into $|Q| - \omega + 1$ sliding query patterns;
- Build hash table $HTable$ and c-trees on the DNA data segments;

2. Two-level Filtering
   - The first level: Hash table based similarity search;
   - The second level: The c-trees based similarity search;
3. Refining:

   Extend the candidates/seeds in both directions to obtain the final alignments with high alignment score between the data subsequence and query sequence.

4.5.1 The First Level Filter: Hash Table Based Similarity Search

After the hash table is constructed on the segments of DNA sequences, our search technique partitions the given query sequence Q into $|Q| - \omega + 1$ sliding query patterns $q_1, q_2, ..., q_{|Q|-\omega+1}$. Given the q-gram clusters $\{qCluster_1, \ldots, qCluster_\lambda\}$, the query pattern q_i is first encoded to a hash key h_i, which is a λ bit integer. Then all the encoded neighbors $ngbr$ of the hash key h_i are enumerated, and the neighbors are those λ bit integers encoded from the segments which are within a small edit distance from q_i.

In [27], an approach has been proposed to enumerate a segment's neighbors. The main idea is also applicable for our current case, but the difference is that we need to consider the impact on the q-grams to get the encoded neighbors when some edit operations are conducted on the segment. The d edit operations on segment s will result in at most dq q-grams which are different from those in s, and the new neighboring key will be computed in terms of the new group of q-grams by using the coding function.

Algorithm 6 illustrates the similarity search based on the first level indexing structure – hash table. For a query pattern q_i, once an encoded neighbor e_{ngbr} of q_i is enumerated, the segments in the bucket $HT[e_{ngbr}]$ of the hash structure HT will be retrieved as candidates and stored into the candidate set C_{ht}.

Algorithm 6 The First Level Filter Algorithm
Input: Hash structure HT, Query pattern q_i.
Output: Candidate C_{ht}.
Method:

1: $C_{ht} \leftarrow \emptyset$;
2: Encode the query pattern q_i to a hash key h_i in terms of encoding function;
3: Enumerate the next neighbor $ngbr$ of h_i;
4: **for** each segment s in the bucket $HT[ngbr]$ **do**
5: **if** s and q_i are similar **then**
6: $C_{ht} \leftarrow C_{ht} \cup s$;
7: **end if**
8: **end for**
9: **Until** all the neighbors of q_j are enumerated;

4.5.2 The Second Level Filter: The c-trees Based Similarity Search

The candidate segments C_{ht} generated from the first level filter will be further verified by the second level indexing structure, c-trees. According to the c-trees structure, the c-signature $sig^c(q)$ of query q is divided into δ c-signature strings which are $sig_i^c(q)$, $0 \leq i < \delta$. Algorithm 7 shows how to retrieve the segment s which satisfies the range constraint $edit(q, s) \leq \theta$ for a query segment q based on the second level indexing structure – c-trees. For clarity, threshold γ in Algorithm 7 is set to $2q\theta$ or a value smaller than $2q\theta$, where q is the q-gram length and θ is the edit distance allowed between the DNA data segment and query segment.

In Algorithm 7, $w_i[lNode]$ is used to denote the distance between $sig_i^c(q)$ and the path label $pl = label[\langle root_i, lNode\rangle]$ from the respective root $root_i$ to $lNode$ in

T_i, namely $w_i[lNode] = SDist(sig_i^c(q), pl)$. We use $score[s]$ to denote the partial distance for segment s during similarity search. Also for notation simplicity, we use $sig_i^c(s)$ as its corresponding path label for a leaf node in $T_i, 0 < i < \delta$, since each $sig_i^c(s)$ can only be mapped to one path or one leaf node in T_i.

Algorithm 7 The Second Level Filter Algorithm
Input: The c-trees $(T_0, T_1, ..., T_{\delta-1})$ on $\mathcal{D}$, query c-signature $(sig_0^c(q), ..., sig_{\delta-1}^c(q))$, Candidate segments C_{ht}, distance γ.
Output: Candidate set C.
Method:

```
1: C ← ∅
2: for lNode ∈ T_0 do
3:    if  w_0[lNode] < γ then
4:       E_0'[lNode] ← E_0[lNode] ∩ C_ht;    C ← C ∪ E_0'[lNode]
5:       for each s ∈ E_0'[lNode] do
6:          score[s] ← w_0[lNode]
7:       end for
8:    end if
9:    return Search({T_1, ..., T_{δ-1}}, C)
10: end for
11:
12: Function: Search(TSet, C)
13: if TSet = ∅ then
14:    return C
15: else
16:    T_i ← first entry in TSet
17:    for each s ∈ C do
18:       if w_i[sig_i^c(s)] + score[s] ≤ γ then
19:          score[s] ← w_i[sig_i^c(s)] + score[s]
20:       else
21:          C ← C − {s}
22:       end if
23:    end for
24:    return Search(TSet-{T_i}, C)
25: end if
```

During query processing, for each leaf node $lNode$ in tree T_0, the distance $w_0[lNode])$ between the path label of $lNode$ and $sig_0^c(q)$ are computed. And the initial candidate set C includes those segments in $E_0[lNode] \cap C_{ht}$ where

$w_0[lNode] \leq \gamma$. For the other trees $\{T_1, \ldots, T_{\delta-1}\}$, candidates will be pruned based on the partial distance gradually by executing the function $Search(TSet, C)$ recursively. For each candidate s in C, we can find its corresponding leaf node $lNode$ with label $sig_i^c(s)$ in tree $T_i (i \neq 0)$ in time O(1) by links constructed during tree construction, and the partial distance $score[s]$ can be computed as well. For the tree T_i, if the new partial distance for candidate segment s, $w_i[sig_i^c(s)] + score[s] \leq \gamma$, then the partial distance $score[s]$ for segment s is modified as $w_i[sig_i^c(s)] + score[s]$, else the segment s is removed from candidate set C since segment s is not a candidate. The function $Search(TSet, C)$ is executed until $TSet$ is empty.

4.5.3 The Space and Time Complexity Analysis

In this section, the space and time complexity are analyzed for the two-level index structure. For the space complexity of the hash table, we need $O(2^\lambda)$ for the table head. For the bucket of the table, DNA segments will contribute space $\Theta(|\mathcal{D}|/\omega)$. Thus, the total space complexity for the hash structure will be $O(2^\lambda + |\mathcal{D}|/\omega)$. Each neighborhood of the segment can be generated with time amortized complexity $O(1)$. Thus, the time complexity for the query is $O(|q|)$ for the first level filtering.

Essentially, the space complexity for the c-trees can be divided into two portions: the c-trees themselves, and space occupied by $E_0[*]$ for the first tree T_0 and the links for the other trees. As in the algorithm, $E_0[*]$ must be stored for the first tree, thus they require $O(|\mathcal{D}|/\omega)$ space. The height of each tree is bounded by $O(4^q/(\delta c))$, thus for each tree the storage required for the edge labels is bounded by $O((c+1)^{4^q/(\delta c)} \log(c+1))$. Besides, we also need to maintain the links for the other trees. The space required by links highly depends on the data distribution. Note there are many zeros in c-signatures, thus a lot of links will point to a dummy leaf (by dummy we mean that the path label is 0); and compression may therefore

be beneficial. However, in our implementation and performance study, we do not apply any compression so that we can explain the effects of certain parameters properly. The time complexity depends on the pruning rate for each iteration. Suppose the filtering rate for each iteration is β, then the total time required to obtain the final candidate set is $O(\delta(c+1)^{4^q/(\delta c)} + (|\mathcal{D}|/\omega)\frac{(1-\beta)(1-(1-\beta)^\delta)}{\beta})$ in the worst case. Note in practice, the algorithm is much more efficient since we hardly need to traverse the whole structure.

4.6 Experimental Studies

In this section, we evaluate the effectiveness, sensitivity, efficiency of DNA sequence similarity search based on the two-level index structure. First, we study the effectiveness of the two-level index by examining the effect of number of common q-grams, the effect of parameter c and q, and the effect of parameter ω and θ. Second, we use the theoretical sensitivity model proposed in Chapter 3 to investigate the sensitivity of similarity search based on the two-level index. Third, we use pier model proposed in Chapter 3 to extract the segments, construct the two-level index structure based on the extracted piers and perform the DNA sequence similarity search to investigate efficiency of DNA sequence similarity search based on the two-level index. Finally, we compare the DNA sequence similarity search based on two-level index to hash-based pier model proposed in Chapter 3 and we also compare it against the latest version of BLAST (NCBI BLAST 2). We use the default seed length 11 for BLAST (denoted as BLAST11).

4.6.1 Dataset and Experimental Settings

We use the same DNA sequence datasets as those listed in Table 3.4[2]. They are ecoli.nt (4.68MB), yeast.nt (12.3MB), human_est.fa (23.6MB), month.gss (286.2MB), patnt (702.1MB), other_genome (1.06GB) and other_genome+patnt (1.76GB). All the datasets are real datasets which are downloaded from NCBI website. The programs are implemented in C++ programming language, and are executed on a Linux server with 4 UltraSPARC-III+ CPU of 900MHz and 8GB memory, which is the same as the machine we used for experimental studies in Chapter 3.

4.6.2 The Effectiveness Analysis

We use the dataset *ecoli.nt* to study the effectiveness of the two-level index structure. We investigate the effect of number of common q-grams, the effect of parameter c and q, and the effect of parameter ω and θ on the performance of DNA sequence similarity search based on the two-level index structure.

Effect of Number of Common q-grams

Our proposed search method is based on q-grams, where the q-gram similarity is used as a filter for similarity sequence search in DNA sequences. Therefore we first explore the effect of the number of common q-grams on the proposed search method. For a given data segment with length ω and a certain similarity p to the query segment, we compute the probability of this segment being detected as a hit when we vary the number of common q-grams.

Figure 4.3 depicts the effect of the number of common q-grams on the proposed search method for ω=30, p=67% and $q = 2, 3, 4$. Figure 4.3 shows that the "hit" probability decreases as the number of common q-grams increases. It is because

[2]Table 3.4 lists the datasets used for experimental study in Chapter 3.

that the probability to get a hit will be lower when more common q-grams are needed. We also observe that smaller q led to higher hit probability when we fixed the number of q-grams, due to the fact that it is easier to get more common q-grams when q is smaller.

We also conduct the experiment to compute the hit probability for BLAST11. For ω=30, p=67%, the hit probability is 0.0338. In comparison with BLAST11, when ω=30, p=63% and q=4, our q-gram based method achieves higher hit probability than BLAST11 as long as the number of common q-grams smaller than 10.

Effect of Parameter c and q

We next investigate the effect of parameter c and q on the filter rate of the two-level index structure. The filter rate used in the experiment study is defined as below:

$$Filter\ Rate = 1\text{-}\ \frac{total\ number\ of\ hits\ found}{total\ number\ of\ segments\ in\ data\ sequence}.$$

We first conduct experiment to measure how the parameters c and q affect the effectiveness of filtering when we fix the segment length ω and error tolerance θ in the filter processing. Since we focus on investigating the effect of longer seeds on the performance of the search by using two-level index structure, we set ω=30 with error tolerance θ=3 [3]. The result in Figure 4.4 shows that as c increases, the filter rate deteriorates since the c-signature representing the segment becomes inaccurate. On the other hand, larger q results in better filter rate since the segment property can be captured more accurately by the c-signatures. We can achieve the very high filter rate, 99.9495%, for q=4 and c=3. It means that two-level index is very effective for filtering. We will use q=4 and c=3 as the default parameter values for the efficiency analysis in the following experiment.

[3] We will next investigate the effect of different value of ω and θ on the effectiveness of the search in the subsequent experiments.

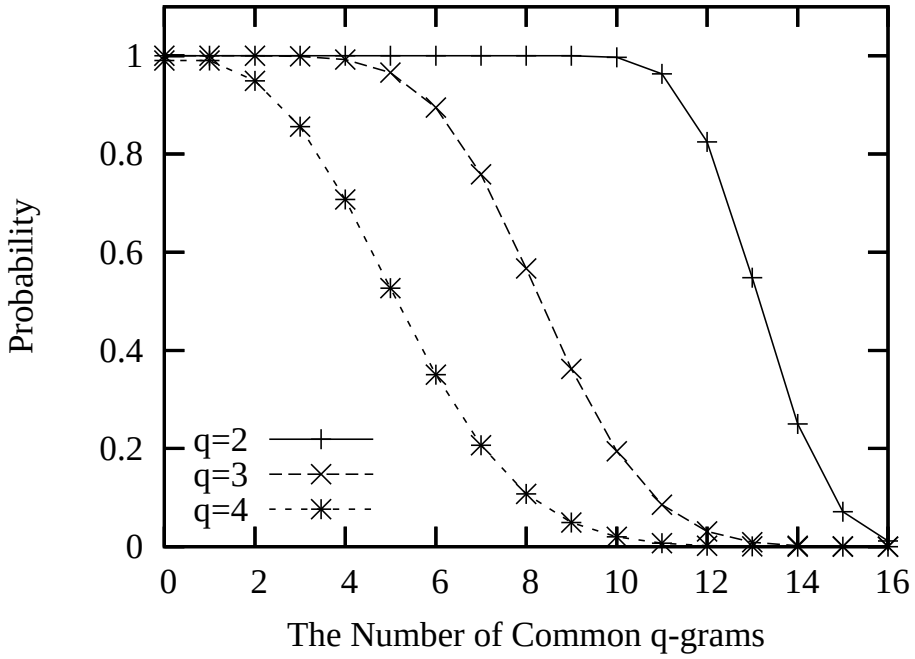

Figure 4.3: Effect of Number of Common q-grams: ω=30, p=67%

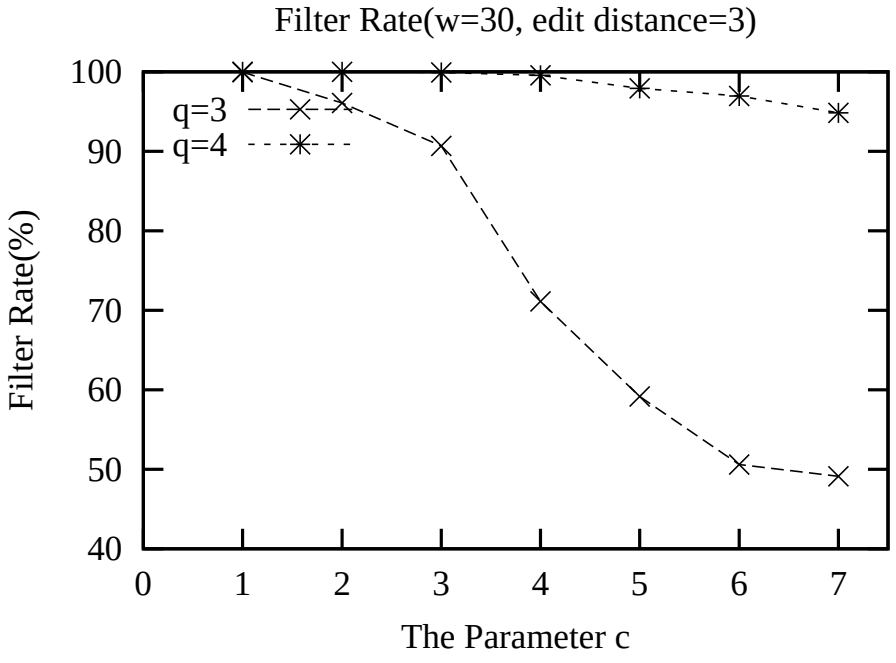

Figure 4.4: Filter Rate vs Parameter c

Effect of Parameter ω and θ

We next study the effect of segment length ω and error tolerance θ. The results are summarized in Figure 4.5. Figure 4.5 shows that the filter rate increases as ω increases and θ decreases. We note that even for ω=30 and θ=4, the filter rate is already as high as 95.895%. When ω=30 and θ=3, we can achieve much higher filter rate of 99.4%. The results confirm that the proposed index structure is very effective for the similarity search. We will use ω=30 and θ=3 as the default parameter values for the efficiency analysis in the subsequent experiment.

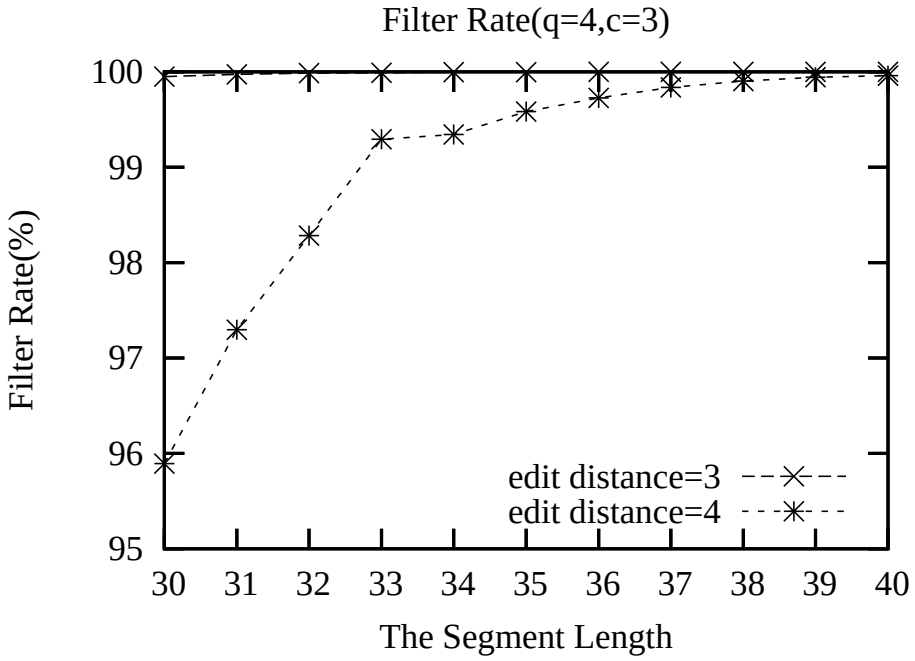

Figure 4.5: Filter Rate vs Segment Length ω

4.6.3 The Sensitivity Analysis

The two-level index structure can be used to organize very long seed with error tolerances. To study sensitivity of longer seed with error tolerances, we use the theoretical sensitivity model which was proposed in Chapter 3. There are three pa-

rameters in the theoretical model: the pier length ℓ_p, error tolerance θ, and number of seeds k; and the pier length ℓ_p corresponds to the segment length ω in the two-level index structure based similarity search. Since the two-level index structure is built on qClusters and c-signatures of data segments, its size is not affected by segment length. Naturally, we can index much longer seeds (segments) by using the two-index structure than using hash-based pier model. On the contrary, both BLASTn and hash-based pier model cannot handle very long seeds (segments) with error tolerances. BLASTn identifies seeds of length from 9 to 11 [22]. The hash-based pier model can not handle very long seeds (for example, seed with length 30) due to the big size of the hash structure and GPM.

Figure 4.6 presents the results on sensitivity where length of random region is set to 64. In the graph, we use (ω,k,θ) to denote the different seed for sensitivity analysis. The results show that smaller ω, bigger k and bigger θ yield higher sensitivity.

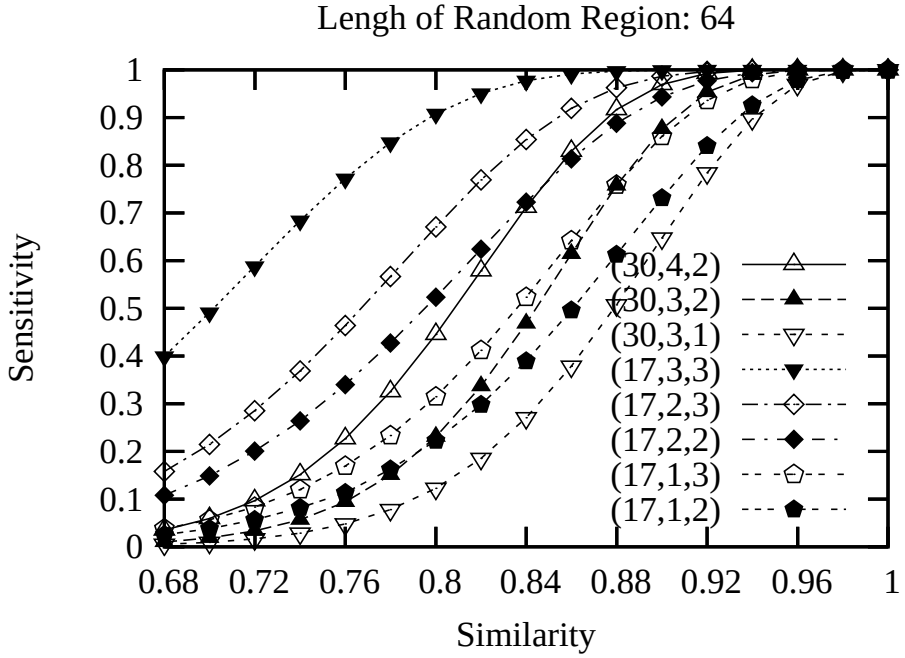

Figure 4.6: Similarity vs Sensitivity

4.6.4 The Efficiency Analysis

In this subsection, we evaluate the efficiency of sequence similarity search based on the two-level index structure. We build the two-level index on the disjoint [4] data segments by using the default parameter values selected in Section 4.6.2 to evaluate the efficiency of the search on the two-level index. We set q=4, c=3, ω=30 and θ=3, and we use *two-level*(30) to stand for the parameter setting. We conduct the experiments to investigate the effect of query length and dataset size on the performance of search based on two-level index for long seeds.

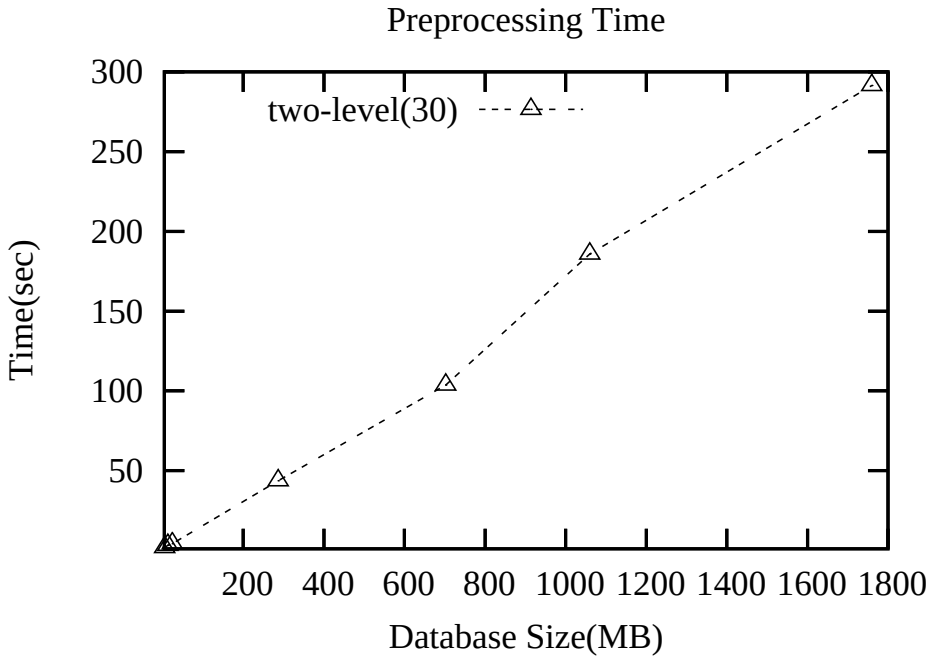

Figure 4.7: Efficiency of Preprocessing

Efficiency in Pre-processing

We would first like to investigate the efficiency of constructing the two-level index structure. Figure 4.7 presents the time of pre-processing the seven DNA sequence

[4]Note that disjoint segment corresponds to pier with span length ℓ_s=0 in Chapter 3.

datasets described in Table 3.4 before performing similarity search. The result shows that the time of pre-processing increases linearly with the size of dataset. We index all the disjoint segments instead of sliding segments in data sequences, and the response time of pre-precessing is scalable with respect to dataset size.

Varying Query Length

We next conduct an experiment on the two datasets *month.gss* and *patnt* to investigate the effect of query length on the performance of the search of the two-level index by setting the query length to 100, 300, 500, 800, 1000, 1500 and 2000. Figure 4.8 shows that the response time of the search increases linearly for both datasets as the query length increases. For dataset *patnt*, it takes about 12 seconds to return the alignments between the query sequence and data sequences when query length is set to 2000. The efficiency of query deteriorates linearly as query length increases because we use all the sliding windows of query sequence as the query patterns in similarity search based on the two-level index structure.

Varying Dataset Size

To study the effect of database size on the search efficiency of two-level index, we fix the query length to 1000, and conduct similarity search on the seven datasets of various sizes. The result in Figure 4.9 shows that the search method based on the two-level index is efficient in detecting the similar regions between the query sequence and data sequences by using longer seeds. Most of all, the results indicate that the efficiency deteriorates linearly with respect to dataset size.

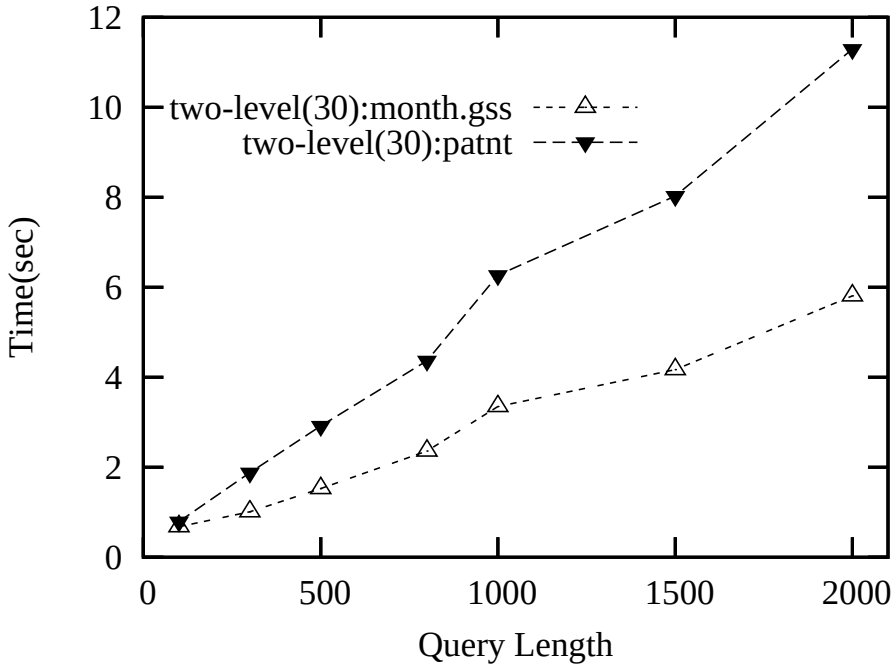

Figure 4.8: Query Time

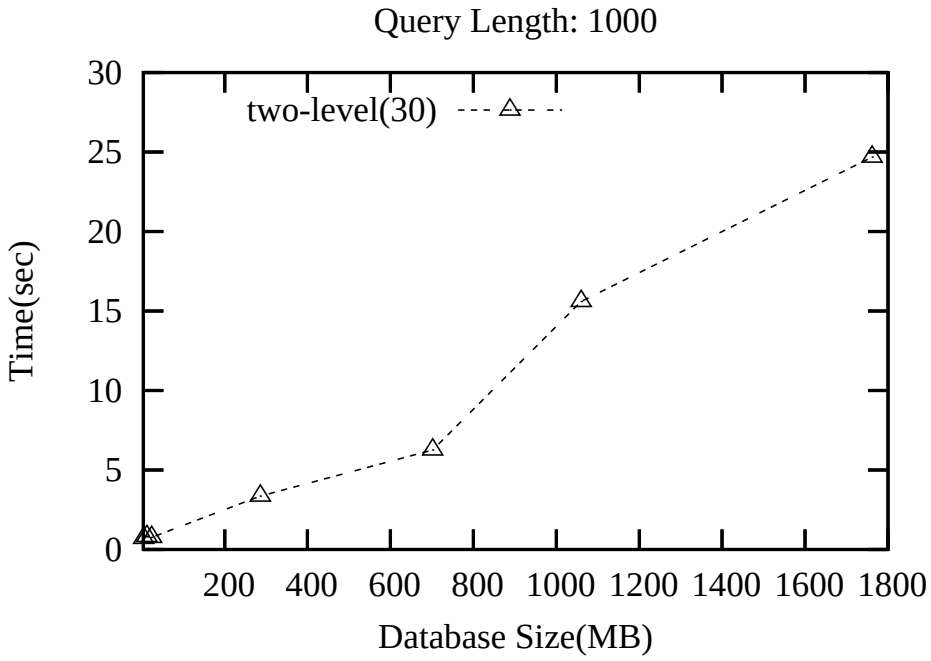

Figure 4.9: Query Time ($|Q|$=1000)

4.6.5 Comparison to Hash-based Pier model and BLAST11

In this subsection, we compare the performance of two-level index method with the hash-based pier model and BLAST11. First, we evaluate the efficiency of pre-processing of the three methods. Then we conduct experiments to investigate the effect of query length and the size of dataset on the performance of the two-level index method in comparison with both hash-based pier model and BLAST11. Finally, we analyze the results returned by the three methods in terms of precision and recall. In Chapter 3, we noted that when compared to BLAST11, the hash-based pier model can achieve higher recall and acceptable precision using parameter setting ℓ_p=17, ℓ_s=5 and θ=3. In the subsequent experiments, we therefore use the same parameter setting ℓ_p=17, ℓ_s=5 and θ=3 on the two-level index structure to compare against hash-based pier model and BLAST11. For two-level index method, we use $two-level(17)$ to denote the parameter setting; for hash-based pier model, we use $Pier(17)$ to denote the parameter setting.

Efficiency in Pre-processing

We first evaluate the efficiency in data sequence pre-processing before performing similarity search for the three search methods. Figure 4.10 shows that pre-processing of hash-based pier model is fastest, followed by the two-level index method and BLAST11. The hash-based pier model and two-level index method are fast to pre-process the sequence database since we only index the extracted "piers" instead of indexing all the sliding segments in data sequences as in BLAST11. Furthermore, the hash-based pier model directly hashes the extracted "piers" into hash table while the two-level index method needs to first transform the "piers" into c-signatures and qClusters in terms of q-grams. As a result, it takes more time for the two-level index structure to build the index structure than the hash-based

pier model. In all, the pre-processing of the hash-based pier model and two-level index is about 2-10 times faster and 2-6 times faster than BLAST11, respectively.

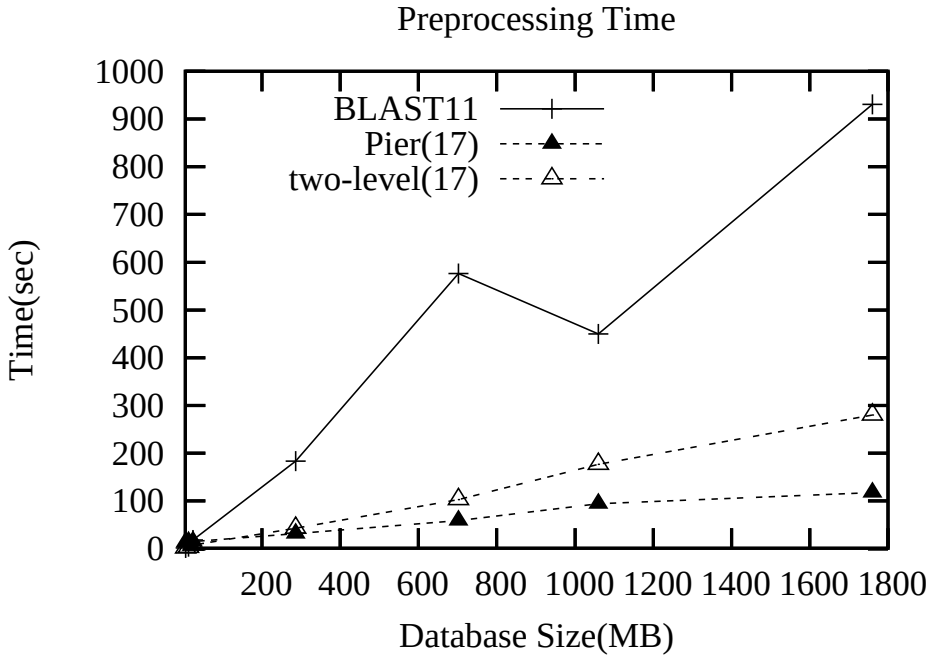

Figure 4.10: Efficiency of Preprocessing

Varying Query Length

An experiment is also conducted to investigate the effect of query length on the performance of two-level index method in comparison with hash-based pier model and BLAST11. In this experiment, we set the query length to 100, 300, 500, 800, 1000, 1500 and 2000. We perform the similarity search on dataset *patnt*. We found that both the hash-based pier model and two-level index method are faster than BLAST11 when we vary query length. Figure 4.11 shows that the hash-based pier model maintains a good margin in efficiency even as the query length increases compared to both two-level index method and BLAST11. This is because that the hash-based pier model enumerates the neighbors of query segments, and then retrieves the segments in hash structure in terms of the neighbors in time O(1). It

does not need to do any computation for candidate detection during the search. Figure 4.11 also shows that the two-level index method is faster than BLAST11 as we vary the query length from 100 to 2000, however, the gain of the two-level index method decreases slightly for large query length as it needs to transform all query sliding windows to their corresponding hash keys and c-signatures for the search.

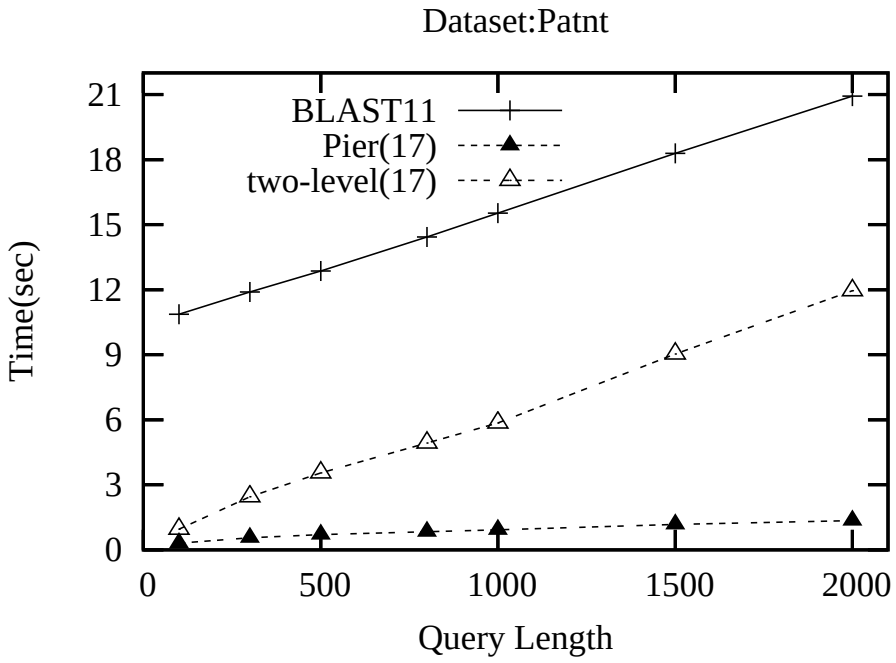

Figure 4.11: Query Time (Dataset:patnt)

Varying Dataset Size

Similarly, we perform a comparison of two-level index search method with hash-based pier model and BLAST11 on the seven datasets as well. To study the effect of database size on the three methods, we run them on seven datasets of various sizes when the query length is fixed to 1000. The results in Figure 4.12 show that both the hash-based pier model and the two-level index method perform well for large datasets and the margin over BLAST11 is widened as database size increases.

The reason is obvious: we use index structures to organize the extracted piers from the data sequences, while BLAST11 basically does the sequential scan on the data sequences. We note that other characteristics of the undivided dataset may have effects on the results, which are however not significant since the size dominates the cost in this test.

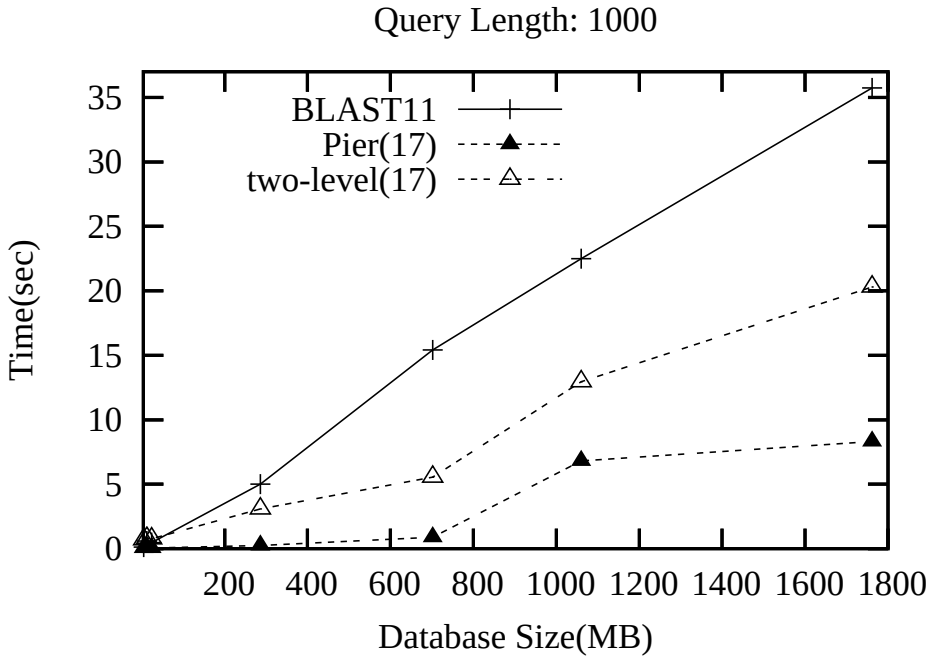

Figure 4.12: Query Time ($|Q|$=1000)

4.6.6 Search Accuracy Analysis

We now conduct the experiments to compare and analyze the accuracy of the results of two-level index method, hash-based pier model and BLAST11 in terms of two search accuracy metrics: precision and recall (The two accuracy metrics have been earlier defined in Section 3.7.5). To evaluate the precision and recall of the search methods, we use the same datasets and evaluation methods as those used in Section 3.7.5. We use 400,00 sequences in human EST dataset as data sequences;

we randomly select 20 sequences from mouse EST dataset as query sequences. We perform the search accuracy analysis on the results of the search methods: two-level index method, hash-based pier model and BLAST11. Table 4.2 shows the precision and recall of the three search methods. In Table 4.2, $T(30)$, $T(17)$ and $P(17)$ are the abbreviations of two-level(30), two-level(17) and Pier(17), respectively, which are used for efficiency analysis in Section 4.6.4 and Section 4.6.5.

QID	$T(30)$ **Precision**	$T(30)$ **Recall**	$T(17)$ **Precision**	$T(17)$ **Recall**	$P(17)$ **Precision**	$P(17)$ **Recall**	**BLAST11 Precision**	**BLAST11 Recall**
1	1	0	0.025057	0.275362	0.055556	0.014493	0.428571	0.012
2	0.156085	0.059476	0.027625	0.157258	0.26158	0.096774	0.895833	0.172
3	1	0	0.018717	0.007035	0.333333	0.003015	1	0.008
4	1	0	0.026607	0.077699	0.054264	0.007064	0.384615	0.005
5	0.08046	0.007028	0.025541	0.60241	0.144172	0.047189	0.1	0.001
6	1	0	0.010152	0.032193	0.034483	0.002012	0.358974	0.014
7	1	0	0.017414	0.125376	0.077519	0.01003	0.2	0.002
8	1	0	0.023193	0.043744	0.393939	0.013225	0.785714	0.011
9	1	0	0.020211	0.11269	0.147727	0.013198	0.363636	0.008
10	1	0	0.017698	0.269347	0.056034	0.013065	0.090909	0.001
11	1	0	0.01555	0.08871	0.057851	0.007056	0.375	0.003
12	0.707692	0.046092	0.226158	0.166333	0.649573	0.152305	1	0.202
13	1	0	0.020191	0.091919	0.070313	0.009091	0.478873	0.034
14	1	0	0.022796	0.015045	0.210526	0.004012	1	0.005
15	1	0	0.025784	0.060181	0.085714	0.006018	0.291667	0.007
16	0.06503	0.390244	0.10038	0.402439	0.150746	0.410569	0.484	0.121
17	1	0	0.032601	0.095286	0.289308	0.046138	0.815385	0.053
18	1	0	0.018945	0.22132	0.040541	0.012183	0.032258	0.001
19	0.6	0.003077	0.036889	0.217436	0.3125	0.030769	0.324324	0.012
20	1	0	0.014725	0.139279	0.059701	0.008016	0.142857	0.002

Table 4.2: Precision and Recall of the Results (Dataset:human_est.fa, 20 Queries Randomly Selected from mouse_est.fa)

The results in Table 4.2 show that two-level index method $T(30)$ can achieve higher precision followed by BLAST11, but low recall; and two-level index method $T(17)$ achieves higher recall followed by hash-based pier model $P(17)$, but low precision. Compared to two-level index method $T(30)$, the two-level index method $T(17)$ has lower precision and higher recall. The results of recall for different search methods are also consistent with the results of theoretical sensitivity analysis in Section 4.6.3, advocating that longer seeds lose distant homologies while small ones create too many random hits. In addition, we note that two-level index method $T(17)$ achieves higher recall and lower precision compared to hash-based pier model

$P(17)$. This is because that we use q-gram based filtering method in two-level index method, while we enumerate the neighbors of query segments in hash-based pier model. Some alignments with not very high similarity can be found by the q-gram based filtering method.

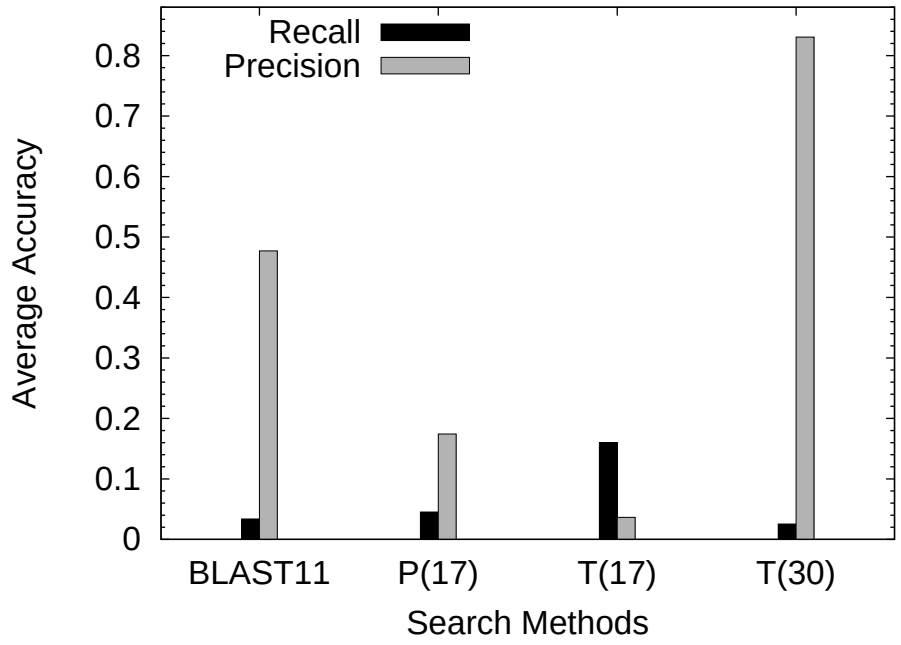

Figure 4.13: Average Accuracy (Dataset:human_est.fa, 20 Queries Randomly Selected from mouse_est.fa)

We also provide an overall evaluation on the average precision and recall for the search methods in Figure 4.13. The average precision and recall are computed based on the precisions and recalls (listed in Table 4.2) of the 20 randomly selected queries. The average precision of $T(30)$ is 0.8305, which is higher than BLAST11; the average recall of $T(30)$ is 0.0253, which is lower than that of BLAST11. Moreover, for the other two search methods $T(17)$ and $P(17)$, they both achieve lower precision but higher recall than BLAST11. In summary, the two-level index method $T(30)$ can achieve higher precision than the hash-based pier model and BLAST11, while the two-level index method $T(17)$ achieves highest recall compared to the hash-

based pier model and BLAST11.

4.7 Summary

We have devised a novel two-level index structure based on q-grams of the DNA sequences which can support efficient similarity search in DNA sequence database. The filtering principle with respect to the index structure is presented and it has been designed to achieve efficient sequence searching while keeping high sensitivity. We conducted experiments to evaluate the performance of our method in terms of sensitivity, effectiveness, efficiency and accuracy. We also conducted the experiments to compare the two-level index search method to our proposed hash-based pier model and BLAST11. The results show that the two-level index method can be used to effectively index very long seeds with error tolerance and efficiently detect the regions in DNA sequence database which are similar to the query sequence. However, for some seeds (length of seeds from 11 to 18) with error tolerances, the two-level index method is not as efficient as the hash-based pier model.

CHAPTER 5

Sequence Join Using Precedence Count Matrix

5.1 Introduction

Many applications manipulate sequence data, for example computational genomics, computational finance, and text and audio processing. One of the most frequently used and expensive operations is the *sequence join* that combines data from two datasets with *similar* sequence values on the join attribute. The similarity between two sequences is typically determined by the *edit distance*.

In this chapter, we study the problem of sequence join in the context of genomic applications, for example in *sequencing by hybridization* and *sequence assembly*, a sequence is assembled from a set of smaller and overlapping subsequences. In sequence assembly, the first step is to find how much a suffix of the first sequence matches a prefix of the second. Sequencing errors are a reality (even if they are only

in the 1-5% range) and suffix-prefix matching must allow for approximate matches [50]. In this context, two sequences are joinable if the prefix of one sequence is similar to the suffix of another with respect to the edit distance. For every ordered pair of sequenced sequences S_1 and S_2, we would compute the longest suffix of S_1 that approximately matches a prefix of S_2.

We propose an efficient filter-and-refine sequence join algorithm to find the longest suffix-prefix match which allows for approximate match for every pair of sequences. In the filtering phase, the proposed scheme can rapidly prune away sequences that are not joinable. In the refinement phase, a more comprehensive alignment scheme is used to filter out the false positives.

In the filtering phase, the key operation is to determine the similarity between two sequences. We propose to use the **precedence count matrix** (PCM) to estimate a lower bound for the edit distance between two sequences. Given the PCMs of two sequences, we derive an efficient algorithm for computing a lower bound for the edit distance between the two sequences. The complexity of this algorithm is $O(|\Sigma|^2 log|\Sigma|)$ where Σ is the alphabet set of the sequences. This means that our algorithm is effective for DNA sequences which have a small alphabet set of size 4. We conducted experiments to evaluate the proposed sequence join algorithm, and our results show that it outperforms existing techniques.

In the next section, we introduce the PCM and the algorithm for approximating the edit distance of two sequences using their PCMs. Section 3 presents the proposed sequence join algorithm. Results from a performance study will be reported in Section 4. In Section 5, we review some related work, and finally, we conclude in Section 6.

5.2 Approximating Edit Distance Using Precedence Count Matrix

The similarity (or distance) between two sequences is typically determined by the *edit distance*, which is computed by using the standard dynamic programming approach. As we have introduced in Chapter 2, it is very costly in terms of both time and space to compute the edit distance when dynamic programming is applied. In this section, we will introduce the precedence count matrix (PCM) and use it to approximate the edit distance between two sequences. The time complexity of our approximate algorithm based on PCM is $O(|\Sigma|^2 log|\Sigma|)$ where Σ is the alphabet set of the sequences. Throughout the discussion in this section, we will use the two sequences in Figure 5.1 and their PCMs as a running example.

Definition 5.2.1 Precedence Count Matrix
Let alphabet Σ be the set of characters $\{A,C,G,T\}$ and Q be a sequence formed from the characters in Σ. The precedence count matrix of Q, denoted as PCM_Q is a $|\Sigma| \times |\Sigma|$ matrix where each element, represented as $PCM_Q[a,b]$, $a \in \Sigma$, $b \in \Sigma$, is the number of unique occurrences of a preceding b (not necessary consecutive) in the sequence Q. □

Example 5.2.1 *Throughout the discussion in this section, we will use the two sequences in Figure 5.1 and their PCMs as a running example. Consider the sequence Q, and the element $PCM_Q[C,T]$. Since there are 2 occurrences of the character 'T' following the first four 'C', $PCM_Q[C,T]$ will be 8. The resultant PCM_Q is shown in Figure 5.1.* □

For the notation simplicity, we will refer $PCM_Q[a,b]$ both as a matrix element and as the precedence count of (a,b), it should be clear for the context.

	A	C	G	T
A	6	25	15	17
C	3	21	15	8
G	1	13	6	4
T	3	27	16	10

(a) PCM of Q

	A	C	G	T
A	21	25	6	15
C	10	10	3	6
G	15	12	3	13
T	20	19	2	10

(b) PCM of R

Q="AAATGTTCCCACTTCGGGCC"
$N_Q(A) = 4,\ N_Q(C) = 7,\ N_Q(G) = 4,\ N_Q(T) = 5$

R="AACGGTTGATATCTACAACC"
$N_R(A) = 7,\ N_R(C) = 5,\ N_R(G) = 3,\ N_R(T) = 5$

Figure 5.1: PCMs of Q and R

For ease of discussion, denote $Diag(PCM_Q) = \{PCM_Q[a,a] | a \in \Sigma\}$ as the **diagonal** of the matrix and other elements in the matrix which are not part of $Diag(PCM_Q)$ will be referred to as **non-diagonal elements**.

Before we describe the algorithm, we will first highlight the following two properties of the PCM.

Property 5.2.1 Occurrence Count Property

Let $N_Q(a)$ denote the number of occurrences of a character 'a' in a sequence Q. Then $PCM_Q[a,a] = f(N_Q(a))$ where $f(n) = \frac{n(n-1)}{2}$. Conversely, given $PCM_Q[a,a]$, we will have $N_Q(a) = f'(PCM_Q[a,a])$ where $f'(n)$ is the inverse of the function $f(n)$[1]*.* □

The occurrence count property simply states that the frequency of a character a in the sequence Q can be derived from its PCM and vice versa. The second property is as follow:

Property 5.2.2 Reverse Sum Property

Given a sequence Q and any two characters a and b, $a \neq b$, $PCM_Q[a,b] + PCM_Q[b,a] =$

[1] $PCM_Q[a,a] = 0$, the value $N_Q(a)$ can be determined with other information.

$N_Q(a) \times N_Q(b)$. □

Noticed that $PCM_Q[a,b]+PCM_Q[b,a]$ is the number of distinct unordered pairs of characters a and b, by the production rule of permutation, the above property can be obtained. The property will be used to reduce the number of cases we have to consider in our algorithm for approximating the edit distance between two instances.

Given the precedence matrix of two sequences Q and R, an algorithm for approximating the minimum edit distance between Q and R is sketched in Algorithm 8. We shall describe the three steps of the algorithm in more details.

Algorithm 8 Estimate Edit Distance
Input: PCM_Q and PCM_R.
Output: Lower bound of edit distance between Q and R.
Method:

1: Compute the minimum number of operations (insertion, deletion or replacement) $\mathcal{D}_1$ required to transform $Diag(PCM_Q)$ into $Diag(PCM_R)$.
An algorithm in [61] can be adopted for this step. Let this set of operations be denoted as $OPER$ and let the transformed precedence count matrix of Q be PCM'_Q.
2: Compute elements of PCM'_Q. Let the new precedence count matrix be PCM''_Q.
3: Finally, we need to compute the minimum number of operations $\mathcal{D}_2$ needed to adjust PCM''_Q such that its other non-diagonal elements are the same as PCM_R. This must be done **while keeping the diagonal unchanged**.

5.2.1 Adjusting Diagonal Elements

In the first step, we compute the minimum number of operations needed to adjust the diagonal elements of PCM_Q to be the same as the corresponding diagonal elements of PCM_R. The reason for doing so can be deduced from the occurrence count property we discussed earlier.

The diagonal elements directly correspond to the number of occurrences of each character in the sequence and to convert Q into R, we must make sure the frequency

	A	C	G	T
A	21	25	15	17
C	3	10	15	8
G	**1**	13	3	4
T	3	27	16	10

(a) PCM'_Q

	A	C	G	T
A	21	25	11	17
C	10	10	3	6
G	**10**	12	3	4
T	18	19	11	10

(b) PCM''_Q

Figure 5.2: Intermediate PCMs for Step 1 and 2.

of each character in Q and R is the same. We can just adopt an algorithm from [61]. Denote vector V as $V[a] = f'(PCM_R[a,a]) - f'(PCM_Q[a,a])$, $a \in \Sigma$. Then it is not difficult to deduce the following Lemma from the algorithm in [61].

Lemma 5.2.1 $\mathcal{D}_1 = (\sum_{a\in\Sigma} |V[a]| + ||R| - |Q||)/2$. □

Intuitively, to transform R into S, we need to delete or insert at least $||R|-|Q||$ characters, and then perform at least $(\sum_{a\in\Sigma} |V[a]| - ||R| - |Q||)/2$ replacement operations.

Example 5.2.2 *Consider the two sequences in Figure 5.1. Since sequence Q has only 4 occurrences of 'A' while sequence R has 7 occurrences of 'A', 3 'A' are necessary to be inserted. On the other hand, we need to delete 2 'C' and 1 'G' from Q so that the transformed PCM will have the same diagonal values as R.* $\mathcal{D}_1$ *has a value of 3 and* $V[A,C,G,T]$*=*$\{$*3,-2,-1,0*$\}$*. The resultant* PCM'_Q *is shown in Figure 5.2(a).* □

5.2.2 Computing Maximum Impact

In this phase, our aim is to assess how the various edit operations we compute in the earlier stage will impact the non-diagonal values of PCM'_Q. By adjusting PCM'_Q according to the maximum impact, we will derive PCM''_Q.

	$V[a] > 0$ Insert a	$V[a] < 0$ Delete a	$V[b] > 0$ Insert b	$V[b] < 0$ Delete b
Case (I) $PCM_R[a,b] \geq PCM'_Q[a,b]$	+	0	+	0
Case (II) $PCM_R[a,b] \leq PCM'_Q[a,b]$	0	-	0	-

Figure 5.3: Assessing Impact of Edit Operations on Non-Diagonal Element $PCM'_Q[a,b]$

SubCases	Conditions	Computing PCM''_Q
(1)	$V[a] > 0$ $V[b] > 0$	$PCM''_Q[a,b] = min\{PCM_R[a,b], PCM'_Q[a,b] + N_R[a]N_R[b] - N_Q[a]N_Q[b]\}$
(2)	$V[a] \leq 0$ $V[b] > 0$	$PCM''_Q[a,b] = min\{PCM_R[a,b], PCM'_Q[a,b] + V[b]N'_Q[a]\}$
(3)	$V[a] > 0$ $V[b] \leq 0$	$PCM''_Q[a,b] = min\{PCM_R[a,b], PCM'_Q[a,b] + V[a]N'_Q[b]\}$
(4)	$V[a] \leq 0$ $V[b] \leq 0$	$PCM''_Q[a,b] = PCM'_Q[a,b]$

Figure 5.4: Subcases for Case (I)

Note that an operation has an impact only if it brings the non-diagonal values of PCM'_Q closer to the non-diagonal values of PCM_R. We assess this impact individually for each non-diagonal value $PCM'_Q[a,b]$ where $a \neq b$. We show the two cases we have to handle in Figure 5.3 together with the edit operations involved and their potential impact. In Figure 5.3, '+' means that inserting a or b can affect Case (I), '$-$' means that deleting a or b can affect Case (II), and '0' means that there is no influence on both Case (I) and Case (II).

Note that $V[a] > 0$ corresponds to an insertion of at least one character a in sequence Q and $V[a] < 0$ corresponds to a deletion of at least one character a in Q.

Referring again to the table in Figure 5.3, we can see that only two types of operations will affect $PCM'_Q[a,b]$ in each case.

Give any one of the two cases, there are four additional subcases to be considered

when computing the maximum impact. We will illustrate this for case (I) here and leave out case (II) which is a "mirror" image of case (I). Figure 5.4 shows the four subcases and how PCM''_Q is to be computed for each of them. Our explanation for each of the four subcases is as follow:

- **Case (I) 1:** We have insertions for both characters a and b. Since $PCM'_Q[a,b] \leq PCM_R[a,b]$, these insertions will have an impact on $PCM'_Q[a,b]$. Assuming that all inserted a precede all b and all inserted b follow all a, the maximum increase in $PCM'_Q[a,b]$ can be computed as $N_R[a] * N_R[b] - N_Q[a] * N_Q[b]$. Since our aim is to bring $PCM'_Q[a,b]$ to be as close to $PCM_R[a,b]$ as possible, we compute $PCM''_Q[a,b]$ to be the minimum of $PCM_R[a,b]$ and $PCM_Q[a,b] + N_R[a] * N_R[b] - N_Q[a] * N_Q[b]$.

- **Case (I) 2:** For this case, there are only insertions of b. Inserting b will have an impact on $PCM'_Q[a,b]$ since it is smaller than $PCM_R[a,b]$. The maximum impact is achieved by assuming that all inserted b are behind all character a. This gives a maximum increase of $V[b] * N'_Q[a]$ to $PCM'_Q[a,b]$. Again, since we want to bring $PCM'_Q[a,b]$ closer to $PCM_R[a,b]$, we will take the minimum of the two values.

- **Case (I) 3:** There are only insertions of a in this subcase. The insertion of a will impact $PCM'_Q[a,b]$. The derivation of the formula is similar to Case (I)b and we leave out the explanation for brevity.

- **Case (I) 4:** In this case, there are no insertion of any characters of a and b. Thus, there is no impact on $PCM'_Q[a,b]$ for Case (I), and we will have $PCM''_Q[a,b] = PCM'_Q[a,b]$.

Example 5.2.3 *To illustrate how maximum impact is computed, let us consider the non-diagonal element,* $PCM'_Q[G,A] = 1$ *(highlighted in bold) in Figure 5.2(a)*

which is smaller than $PCM_R[G, A] = 15$ *in our running example. Since there are 1 deletion of 'G' and 3 insertions of 'A', this example falls into subcase (I)2. We thus assume that the 3 additional 'A' are inserted at the end of all 'G'. Since there are 3 'G' in sequence* Q' *(i.e.* $N'_Q(G) = 3$*), the 3 inserted 'A' will at most increase* $PCM'_Q[G, A]$ *by* $V[A] \times N'_Q[G] = 3 \times 3 = 9$*. This means that* $PCM''_Q[G, A] = min(PCM_R[G, A], PCM'_Q[G, A] + 9) = 10$*. We compute the other elements of* PCM''_Q *based on the cases we described above and the resultant* PCM''_Q *is shown in Figure 5.2(b).*

5.2.3 Adjusting Non-Diagonal Elements

Having computed PCM''_Q, this phase will proceed to calculate the minimum number of edit operations that are needed to transform PCM''_Q into PCM_R. We note that since $Diag(PCM''_Q) = Diag(PCM_R)$, we only need to adjust the non-diagonal values of PCM''_Q to be the same as those of PCM_R in the minimum number of operations. This must also be done **while ensuring that the diagonal values of** PCM''_Q **remain the same**, failing which we undo the effect from earlier edit operations. To do so, operations must be done in pairs and according to the following two cases:

- Case 1: $PCM''_Q[a, b] \geq PCM_R[a, b]$ & $PCM''_Q[b, a] \leq PCM_R[b, a]$. The following three pairs of operations will reduce the difference:
  - delete an a that precedes all b and insert an a after all b
  - delete a b behind all a and insert a b in front of all a
  - replace an a which is in front of all b with a b and replace a b which is behind all a with a
- Case 2: $PCM''_Q[a, b] \leq PCM_R[a, b]$ & $PCM''_Q[b, a] \geq PCM_R[b, a]$.

- delete an a behind all b and insert an a in front of all b
- delete an b in front of all a and insert a b behind all a
- replace an a which is behind all b with b and replace b which is in front of all a with a

Note that the third pair of operations in both cases are a combination of the first two pairs of operations using the edit operation replace. Since they perform more swapping at one go, they can reduce the difference between PCM''_Q and PCM_R with fewer edit operations. Thus, we will always perform the third pair of operations to reduce the difference between the two PCMs.

Theorem 5.2.1 *Let Q and R be two sequences. If $Diag(PCM_Q) = Diag(PCM_R)$, then no other cases can be true except the following:*

1. $PCM_R[a,b] \geq PCM_Q[a,b]$ &
   $PCM_R[b,a] \leq PCM_Q[b,a]$
2. $PCM_R[a,b] \leq PCM_Q[a,b]$ &
   $PCM_R[b,a] \geq PCM_Q[b,a]$

Proof: *Since the diagonal values for the two PCMs are equal, we can deduce from the reverse sum property shown in Property 5.2.2 that*

$$\begin{aligned} & PCM_Q[a,b] + PCM_Q[b,a] \\ = \; & N_Q(a) \times N_Q(b) \\ = \; & N_R(a) \times N_R(b) \\ = \; & PCM_R[a,b] + PCM_R[b,a] \end{aligned}$$

Thus the theorem holds. □

Theorem 5.2.1 simplifies the cases and assumptions we have to consider. Next, we will establish the formula to compute the maximum increase or decrease in $PCM''_Q[a,b]$ when swapping the characters 'a' and 'b' by the pair of replace operations.

Theorem 5.2.2 *Given that p pairs of replace operations are performed to reduce the difference between* $PCM''_R[a,b]$ *and* $PCM_R[a,b]$*, then the maximum reduction in the difference is* $p * (N_R(a) + N_R(b) - p)$.

Proof: *We assume that all a are before all b in* Q''*. In this case, every pair of replace operations will give maximum reduction in the difference if the first a is replaced by b and the last b is replaced by a. With i pairs of replace operations, there will be* $(N_R(a) - p)$ *a's in front of* $(N_R(b) - p)$ *b's. Thus, the maximum reduction in difference will be* $N_R(a) * N_R(b) - (N_R(a) - p) * (N_R(b) - p)$ *which gives the above formula.* □

With Theorem 5.2.2, we can now describe our algorithm for the last phase which is shown in Figure 9. In Step 1 of the algorithm, we invoke Theorem 5.2.2 to compute the minimum number of operations $MinOpr$ that are needed to remove the difference between all corresponding pairs of non-diagonal elements for PCM''_Q and PCM_R. In Step 2, the non-diagonal element with the highest $MinOpr$ is chosen and $MinOpr$ is added to $\mathcal{D}_2$ since we need at least this number of operations to transform PCM''_Q to PCM_R. Although only the cell $PCM''_Q[a,b]$ considered to be swapped in Step 2, there could be other cells that are in between them and Step 3 caters to this possibility. For any other letter x, these are the cases in which a non-diagonal elements involving x will be affected.

- **Case 1:** $PCM''_Q[a,b] < PCM_R[a,b]$;

- $PCM''_Q[a, x]$ affected
  if $PCM''_Q[a, x] < PCM_R[a, x]$;
- $PCM''_Q[x, b]$ affected
  if $PCM''_Q[x, b] < PCM_R[x, b]$;
- $PCM''_Q[x, a]$ affected
  if $PCM''_Q[x, a] > PCM_R[x, a]$;
- $PCM''_Q[b, x]$ affected
  if $PCM''_Q[b, x] > PCM_R[b, x]$;

Case 2: $PCM''_Q[a, b] > PCM_R[a, b]$;

- $PCM''_Q[a, x]$ affected
  if $PCM''_Q[a, x] > PCM_R[a, x]$;
- $PCM''_Q[x, b]$ affected
  if $PCM''_Q[x, b] > PCM_R[x, b]$;
- $PCM''_Q[x, a]$ affected
  if $PCM''_Q[x, a] < PCM_R[x, a]$;
- $PCM''_Q[b, x]$ affected
  if $PCM''_Q[b, x] < PCM_R[b, x]$;

To ensure the correctness of our algorithm in computing the lower bound, these affected non-diagonal elements are removed from future consideration in Step 3[2]. This process is repeated until PCM''_Q is transformed to PCM_R. The output of Algorithm 9, $\mathcal{D}_2$ is the minimum number of edit operations that are needed to transform PCM''_Q into PCM_R. By summing up $\mathcal{D}_1$ (computed by Lemma 5.2.1) and $\mathcal{D}_2$, we obtain a lower bound on the edit distance of Q and R.

[2]Otherwise, the process of selecting the non-diagonal elements in Step 2 will be more complicated since we need to analyze how these non-diagonal elements affect each other.

Algorithm 9 Phase 3: Adjust Non-diagonal Elements
Input: PCM''_Q, PCM_R.
Output: $\mathcal{D}_2$.
Method:

1: Find all different non-diagonal elements between PCM''_Q and PCM_R, and compute the minimum number of operations $MinOpr$ for each pair according to Theorem 5.2.2.
2: Find the non-diagonal element $[a, b]$ with maximum $MinOpr$; $\mathcal{D}_2 = \mathcal{D}_2 + MinOpr$.
3: Set $PCM''_Q[a, b]$ and other affected non-diagonal elements to be the same as the ones in PCM_R.
3: Go to 2 until all the non-diagonal elements in PCM''_Q are adjusted to the same as PCM_R.

We now illustrate this final phase with our running example.

Example 5.2.4 *For all the different non-diagonal elements between PCM''_Q and PCM_R in our example which are $[A, G]$, $[A, T]$, $[G, A]$, $[G, T]$, $[T, A]$, $[T, G]$, the corresponding $MinOpr$ are 2, 2, 2, 4, 2 and 4 according to Theorem 5.2.2. Here, $[G, T]$ and $[T, G]$ are the elements with the maximum $MinOpr = 4$ and the related elements $\{[G, A], [A, G], [G, T], [T, G]\}$ since swapping of 'G' and 'T' will affect these elements. Following Step 3 of the algorithm, we have $\mathcal{D}_2 = 4$ after the first loop. In the next loop, only $[A, T]$ and $[T, A]$ with maximum $MinOpr = 2$ is left for further processing. Using the same principle, we obtain $\mathcal{D}_2$=6.* □

Theorem 5.2.3 *The lower bound of edit distance between two DNA sequences Q and R based on PCM is $\mathcal{D}_1 + \mathcal{D}_2$. $\mathcal{D}_1 + \mathcal{D}_2$ can be computed in time complexity $O(|\Sigma|^2 log|\Sigma|)$.*

Proof: *$\mathcal{D}_1 + \mathcal{D}_2$ is computed by following 3 steps described in Algorithm 8. In Step 1, the minimum number of operations is obtained in $O(|\Sigma|)$ time. In Step 2, since the maximum impact is computed only once for each element based on V, PCM''_Q can be generated in $O(|\Sigma|^2)$ time. In Step 3, the elements must be sorted according*

to $MinOpr$ which takes $O(|\Sigma|^2 log|\Sigma|)$. An efficient implementation will take only one scan through the sorted elements to compute $\mathcal{D}_2$. Summing up the three steps, we will see that the complexity of Algorithm 8 is $O(|\Sigma|^2 log|\Sigma|)$.

It is proven in [61] that $\mathcal{D}_1$ is the lower bound of the edit distance between two sequences. For the second and third steps of Algorithm 9, it can be observed that we always overestimate the impact to reduce the differences of the two PCMs while underestimating the edit operations that are needed to do so. As such, $\mathcal{D}_1 + \mathcal{D}_2$ will be a lower bound of the edit distance between Q and R. □

5.3 Approximate DNA Sequence Join

In this section, we will describe how the PCM is useful in DNA sequence join. We assume there are two sets of sequences $PSet$ and $SSet$. We call them **prefix and suffix DNA sequence sets** respectively.

Given two sequences S_i and S_j, sequence approximate join is to find the longest suffix-prefix approximate match of S_i and S_j.

For any general sequence Q, we will use $Q[i:j]$ to denote the subsequence of Q that includes entry in position i through j. The i^{th} suffix of a sequence Q (i.e. $Q[i, |Q|-1]$) will be denoted as $suf(Q, i)$. Similarly, we will use $pre(Q, j)$ to denote the j^{th} prefix of the sequence Q(i.e. $Q[0, j]$).

Let P be a sequence in $PSet$ and S be a sequence in $SSet$. Our objective here is to find all pairs of P and S in which there exists i, j, $min((|S|-i), (j+1)) \geq Minlen$ such that $edit(suf(S, i), pre(P, j)) \leq e$. Here, $Minlen$ and e are user specified threshold and $edit(S, P)$ denotes the edit distance of the two sequences S and P. In order to do so efficiently, the new concept of PCM will be adopted as a fast filter. We will use $PCMEdit(S, P)$ to denote the estimated edit distance of S and

P that is computed based on PCM_S and PCM_P. The method can be described as follows:

1. **Transformation**

   In this step, for each DNA sequence, we generate two sets of PCMs. For each S in $SSet$, the first set of PCMs corresponds to the suffixes of S, i.e., each suffix of S results in a PCM. For each P in $PSet$, the first set of PCMs corresponds to the prefixes of P. For each sequence (either from $SSet$ or $PSet$), the second set of PCMs is generated in the same manner as follows. A set of w-tuples is obtained from a sequence by placing a sliding window of size w over the sequence. Each such w-tuple is transformed into a PCM.

2. **Filtering**

   Potential candidates of DNA sequence join are formed by using PCM as part of the filter. We will discuss the filtering mechanisms shortly. Candidate pairs for sequence join are of the form $\langle suf(S, i), pre(P, j)\rangle$;

3. **Verification**

   For two DNA sequences of length m and n, the edit distance computed by dynamic programming with time complexity $O(mn)$ can be used to process the candidates pairs generated to obtain the final results pairs of DNA sequence join.

Among the three, the transformation and verification steps can be easily understood. We will give details on the filtering step.

5.3.1 PCM-based Filtering of DNA Sequence Join

In this section, we present basic techniques for filtering approximate DNA sequence join based on PCM. The aim is to efficiently identify the candidate answers to our

problems by using the information in the PCMs. As DNA sequence join needs to be done efficiently without false dismissal, we propose the following three filtering techniques.

1. **Distance Filtering**

   In the Distance Filtering scheme, the candidates of sequence join are obtained by using the distance function based on the PCMs, which is the lower bound of edit distance. Algorithm 10 gives the algorithmic description to do so.

Algorithm 10 Distance Filtering Function
Input: $suf(S,i)$, $pre(P,j)$, w, e.
Output: $IsCandidate$.
Method:

```
1: IsCandidate ← true;
2: l ← ||suf(S,i)| − |pre(P,j)||;
3: if PCMEdit(suf(S,i), pre(P,j)) ≤ e then
4:    for each pair PCMs S* and P* of the corresponding disjoint w-tuples of
      suf(S,i) and pre(P,j), respectively do
5:       if (PCMEdit(S*, P*) > (e + l)) then
6:          IsCandidate ←false;
7:          break;
8:       end if
9:    end for
10: else
11:    IsCandidate ←false;
12: end if
13: return IsCandidate;
```

Firstly, for each prefix $pre(P,j)$ and suffix $suf(S,i)$ with length not shorter than $Minlen$, we compute the distance between the corresponding PCMs of the prefix and suffix. If the distance is greater than a given threshold e, this pair of prefix and suffix is not candidate. Otherwise, the PCMs of the disjoint sub-windows of the prefix and the suffix will be used as another layer of filtering. If all the distances between all the corresponding PCMs of the sub-windows are not greater than $e + ||pre(P,j)| - |suf(S,i)||$, $pre(P,j)$ and

$suf(S, i)$ will be accepted as a candidate, or else this pair will be filtered out.

2. **Length Filtering**

   The sequence length can be used as a filter for sequence join. If $||S|-|P|| > e$ then $edit(S, P) > e$.

   The function $LenFilter(S, e, P)$ returns the set of the prefixes of sequence P, $pre(P, j)$, which meets the $||S| - |pre(P, j)|| \leq e$.

3. **Heuristic Filtering**

   Assume $edit(suf(S, i), pre(P, j)) = k > e$ and $|suf(S, i)| = |pre(P, j)|$, for $suf(S, x_i)$ and $pre(P, y_j)$, where $i \leq x_i \leq n$, $0 \leq y_j \leq j$, we have $edit(suf(S, x_i), pre(P, y_j)) + (x_i - i) + (j - y_j) \geq k$ since k is edit distance for the best alignment of $suf(S, i)$ and $pre(P, j)$. Furthermore, according to the Length Filtering technique, the possible candidate pair $suf(S, x_i)$, $pre(P, y_j)$ should satisfy $x_i \geq i + k/2 - e$. Thus, if $x_i < i + k/2 - e$, we can eliminate the pair $\langle suf(S, x_i), pre(P, y_j)\rangle$ quickly.

We used a nested-loop approach to join two DNA sequence sets. Given e and $Minlen$, Algorithm 11 described can be used to generate the candidates of DNA sequence join for each sequence S in suffix set and each sequence P in prefix set.

5.4 Experimental Results

We implemented and evaluated the proposed PCM method with the three filtering schemes. As references, we also compared our scheme against the q-grams method (denoted qgram) and frequency vector (denoted FV) method.

Under qgram, an auxiliary file that stores the q-grams information will be created beforehand. We also used the Length Filtering and Heuristic Filtering, as well

Algorithm 11 Generate Candidates for DNA Sequence Join
Input: S, P, e, $Minlen$.
Output: $\langle suf(S,i), pre(P,j)\rangle$.
Method:

```
1: i ←0;
2: for suf(S,i) of sequence S do
3:    for each pre(P,j)∈LenFilter(suf(S,i),e,P) do
4:       if suf(S,i),pre(P,j) is regarded as candidate pair by following Algorithm 10 then
5:          Return ⟨suf(S,i),pre(P,j)⟩;
6:       else
7:          k ← PCMEdit(suf(S,i),pre(P,j));
8:       end if
9:    end for
10:   Use heuristic filtering to decide next suf(S,i);
11:   i ← i + max(k/2 − e,1);
12:   if |suf(S,i)| < MinLen then
13:      break;
14:   end if
15: end for
```

as the count filtering technique proposed in [48].

For the frequency vector (FV) method, we used the *Frequency distance* proposed in [61] as the distance filtering in our implementation. In addition, the Length Filtering and Heuristic Filtering are also deployed.

We also looked at two integrated strategies: PCM+qgram method and FV+qgram method. Both methods extend their base method (i.e., PCM and FV respectively) by using qgram method as a further filter for the candidates pairs generated by the respective base methods.

We randomly generated two datasets, prefix dataset and suffix dataset, from a complete sequence in *Ecoli* sequence database. Each sequence dataset consists of 1000 DNA sequences with the length varying between 200 and 300.

e	PCM	PCM+qgram	FV	FV+qgram	qgram
1	99.9985%	100%	99.81%	100%	100%
2	99.42%	99.998%	96.66%	99.998%	99.997%
3	92.47%	99.986%	84.98%	99.98%	99.98%
4	75.17%	99.918%	66.39%	99.91%	99.90%
5	54.42%	99.387%	47.25%	99.36%	99.28%

Figure 5.5: Filtering Rate for Minlen=40

$Minlen$	PCM	PCM+qgram	FV	FV+qgram	qgram
20	20.50%	32.09%	14.99%	27.91%	80.97%
30	39.17%	92.17%	32.27%	91.69%	90.73%
40	54.42%	99.39%	47.25%	99.36%	99.28%
50	64.74%	99.88%	58.36%	99.87%	99.87%
60	71.92%	99.96%	66.46%	99.96%	99.96%

Figure 5.6: Filtering Rate for e=5

5.4.1 Effect of Edit Distance e

In the experiment, we study the effect of e on the five join algorithms. We vary e from 1 to 5 for $Minlen = 40$ and $q = 3$. The size of q was set 3 for q-grams method since it always gives the best performance for our algorithm. We note that this is consistent with the observation given in [48]. The sliding window size w for PCM and FV is set as 40 in this experiment.

The filtering rate of the schemes is shown in Figure 5.5. The efficiency of the schemes is shown in Figure 5.7, Figure 5.8 and Figure 5.9 for the filtering, verifying and total processing of the sequence approximate join, respectively. The results show that qgram is very effective but least efficient. We also observe that PCM method is generally superior over FV method. However, though they provide fast filtering, the inability to prune away dissimilar sequences results in high refinement computation overhead. On the whole, they still outperform qgram. This is because the distance function based on PCM can cause better effect of filtering with low

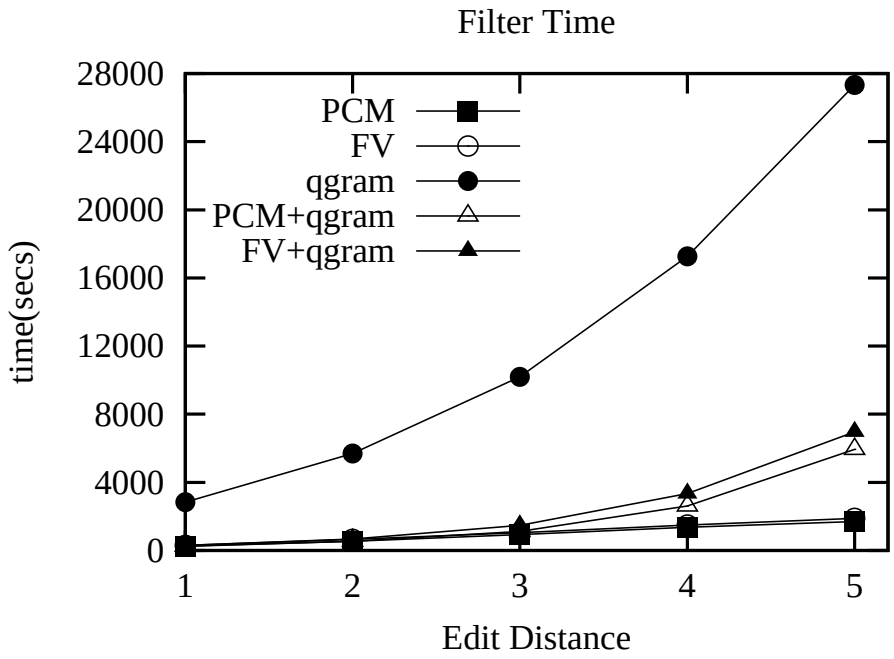

Figure 5.7: Filter Time vs Edit Distance (Dataset Size:1000, Minlen=40)

cost of computation. Finally, we note that the integrated methods, PCM+qgram and FV+qgram outperform qgram in terms of both effectiveness and efficiency, with PCM+qgram being slightly more superior.

Figure 5.9 plots the total time including filtering time and verifying time of DNA sequence join for all the five methods. With e increasing, the performance of PCM and FV deteriorates since the verifying time (shown in Figure 5.8) increases fast due to the bad filtering rate. Figure 5.9 shows the performance of both PCM and FV improves a lot with e increasing when combined with q-grams method. Furthermore, PCM+qgram method can outperform all the other methods as the edit distance e increases.

We also note that with increasing e values, the performance of all schemes degenerates. This is expected since more false positives are being retained in the filtering step.

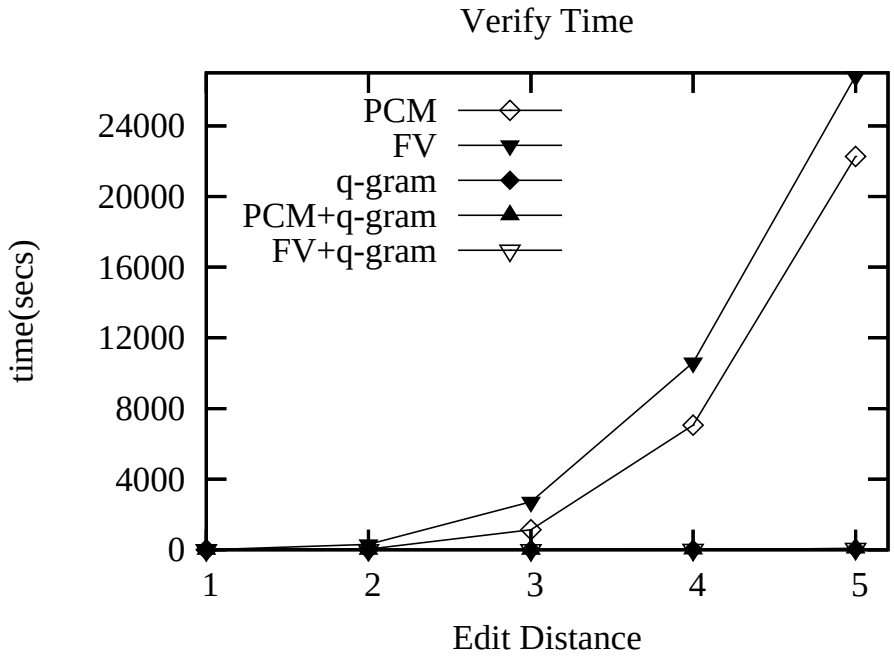

Figure 5.8: Verify Time vs Edit Distance (Dataset Size:1000, Minlen=40)

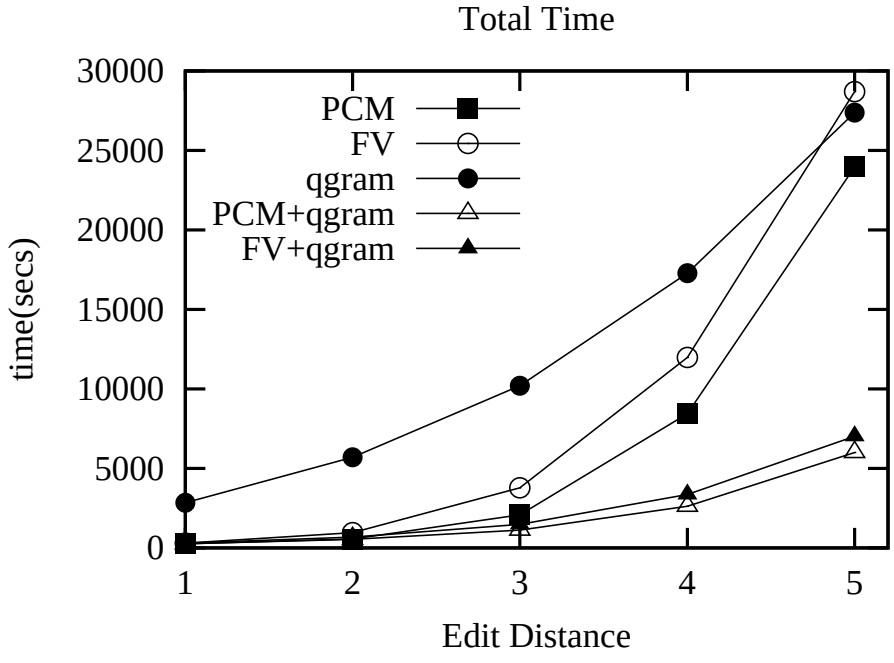

Figure 5.9: Total Time vs Edit Distance (Dataset Size:1000, Minlen=40)

5.4.2 Effect of Minlen

We also study the effects of Minlen, on the schemes on the same dataset. This experiment varies Minlen from 20 to 60 for $e = 5$. Figure 5.6 shows the filtering rate. Figure 5.10, Figure 5.11 and Figure 5.12 plot the filtering time, verifying time and the total time of the five methods for the DNA sequence approximate join processing.

As expected, the results show that all the schemes are less effective for small Minlen as the number of false candidates increases with smaller Minlen. The relative performance of the five methods is consistent with the results of the earlier experiments: PCM+qgram is the best in terms of both filtering rate and total running time, followed by FV+qgram, PCM, and qgram is superior over FV when Minlen decreases.

5.5 Summary

In this chapter, we have proposed a filter-and-refine sequence join algorithm for genomic applications. In the filtering phase, sequences that are not joinable are pruned away rapidly. The refinement phase employs an efficient algorithm to remove the remaining false alarms. The proposed scheme employs the precedence count matrix (PCM) to compute the edit distance between two DNA sequences efficiently. We have evaluated the proposed sequence join algorithm, and our experimental study shows that it outperforms known techniques.

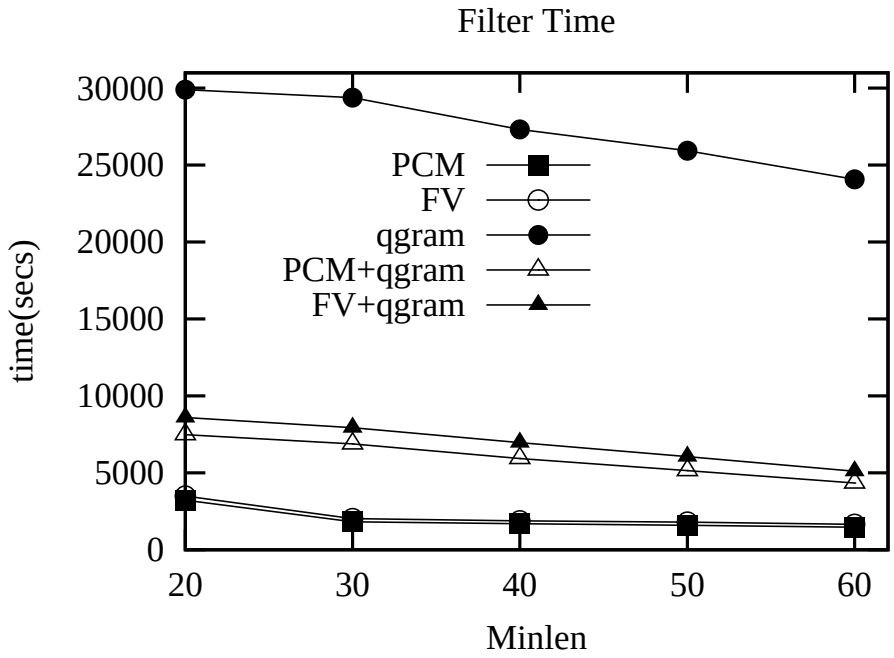

Figure 5.10: Filter Time vs Minlen (Dataset Size:1000, e=5)

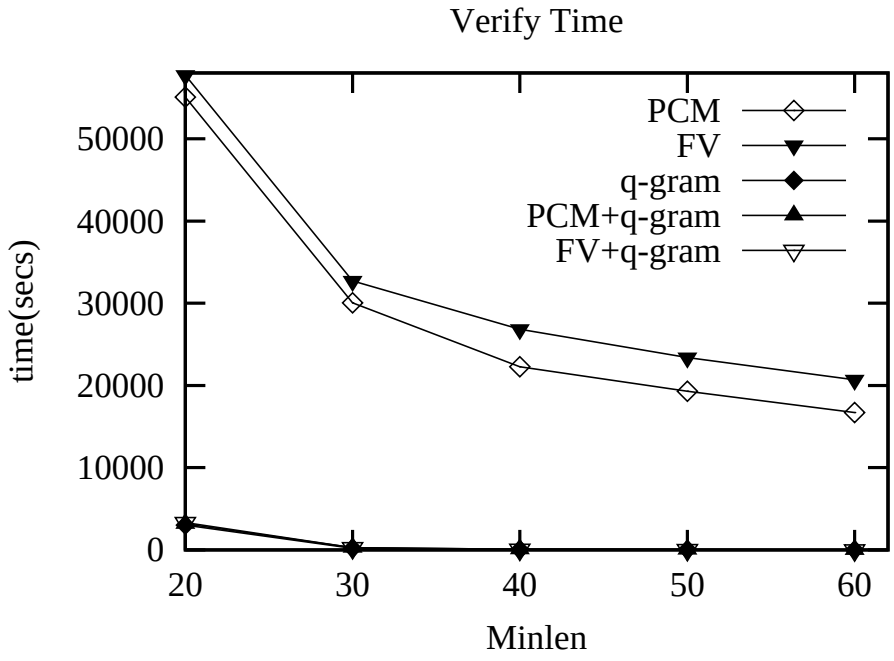

Figure 5.11: Verify Time vs Minlen (Dataset Size:1000, e=5)

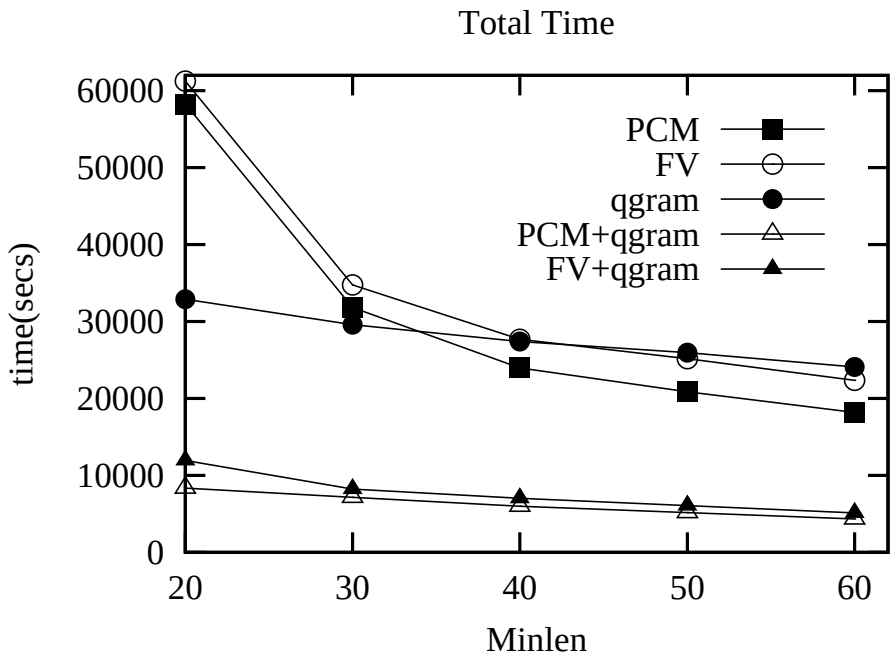

Figure 5.12: Total Time vs Minlen (Dataset Size:1000, e=5)

CHAPTER 6

The q-gram Based Protein Subcellular Localization Prediction

6.1 Introduction

With advancements in large-scale genome sequencing, biologists have produced a huge amount of nucleic acid sequences and amino acid sequences, which are available in public databases. There are more than 1,200 genome sequences stored in public databases. Since a protein's subcellular localization is closely related to biological functions and it is one of the key features of a protein, it is therefore very important to use automated annotation systems to identify or predict the subcellular localization of proteins.

Some efforts have been made in the prediction of protein subcellular localizations. Since 1991, a number of systems have been developed to support automated prediction of protein subcellular localization using different approaches. In these

systems, machine learning methods such as Artificial Neural Networks (ANN), the k-nearest neighbors method, and the Support Vector Machine (SVM) have been applied on different feature extraction from protein sequences.

For example, there are some programs, such as PSORT(PSORTI) for prokaryotic organisms [81], PSORTII [80], iPSORT [11] and TargetP [40] for eukaryotic organisms, and NNPSL [100] for both organism classes. The existing methods may be grouped into three categories. The first category of methods use the similarity search to assign functions including the subcellular localization of a protein. Subcellular localizations tend to be evolutionarily conserved; thus, homology to a protein of known localization can be a good indicator of a protein's actual localization site [79]. However, this method fails when the query sequence and the target protein sequence are not significantly similar. The second category of methods use the sequence motifs such as peptide signals, or nuclear localization signals, which are short subsequences with a length of 3-70 amino acids [40]. The problem with these methods is that sometimes it is very difficult to find universal motifs for a group of protein sequences. The third category of methods is based on amino acid composition, using some machine learning classifiers to implement the prediction. Biological experiments show that information needed to direct a protein to any localization site is mainly encoded in its amino acid sequence. For example, NNPSL [100] uses ANN, and SubLoc [55] uses SVM as the classifier based on amino acid composition. However, this method may not capture information on sequence order and inter-relationships between amino acids.

There is no single method of prediction which can achieve high prediction accuracy, precision or recall for all protein subcellular localizations. We proposed a new method based on the observation that two sequences may share a certain number of q-grams if the two protein sequences are in the same subcellular location.

Moreover, since there are 20 amino acids in the protein alphabet, we know that the number of all combinations of q-grams in a protein sequence is 20^q.

Our proposed method uses the q-grams of protein sequences to predict the subcellular localizations of these proteins. We proposed using the q-gram frequency vectors, q-gram wavelet vectors, q-gram similarity vectors, and q-gram TF.IDF vectors based on the q-grams of the protein sequence to capture information in protein sequences. These proposed q-grams based transformed vectors are then used as features for an SVM [20, 114] that searches for the hyperplane to separate the two classes of data. An SVM is a kernel learning algorithm, in which all the data are mapped into vectors in a k-dimensional feature space. In the proposed method, the k-dimensional feature space can be one of the four vector spaces: q-gram frequency vector space, q-gram wavelet vector space, q-gram similarity vector space, and q-gram TF.IDF vector space. Furthermore, the feature space can also be any combination of the above four proposed vector spaces.

The remaining of this chapter is organized as follows. In Section 2 we briefly review the related work, and describe the problem of protein subcellular localization prediction. In Section 3, several q-gram based feature extraction methods are proposed and our SVM-based method is described. Section 4 introduces the dataset used in our experiments, as well as the evaluation methods used in our method. Section 5 discusses the experimental results, and Section 6 concludes the chapter.

6.2 Problem Description

In this section, we formalize the problem of subcellular localization prediction in protein sequence datsets.

Protein sequences are chains of amino acids and there are 20 different types of

amino acids. Each amino acid is represented by an English alphabet letter from A to Z, excluding the letters B, J, O, U, X and Z. Given an unlabeled protein sequence S, and a known subcellular localization L, we want to determine whether or not sequence S locates in localization L. We assume there are two protein sequence datasets: a positive dataset and a negative dataset. For a localization L, positive sequences are the protein sequences that locate in localization L, and negative sequences are the protein sequence that do not locate in localization L. The problem can be defined formally as follows:

Problem 6.2.1 *Given the positive protein sequence dataset* $P_{set} = \{p_0, \ldots, p_m\}$ *and the negative protein sequence dataset* $N_{set} = \{n_0, \ldots, n_n\}$*, features in protein sequences are extracted, and a classifier is developed on these features to distinguish the sequence* p_i *in* P_{set} *from* n_j *in* N_{set}*.*

The features extracted from protein sequences can be amino acid composition, overall physico-chemical properties (hydrophobicity, hydrophilicity and polarity), dipeptide composition, fourier analysis tri-peptide frequency, frequent subsequences, etc.

6.3 q-gram Based Feature Extraction Method

This section introduces four sequence transformation methods based on q-grams. The features of proteins sequence can be represented as q-gram frequency vectors, q-gram wavelet vectors, q-gram similarity vectors, and q-gram TF.IDF vectors, respectively. A short description of SVM is also presented. We shall first briefly review the notion of q-grams, which we have already introduced in Chapter 4.

Definition 6.3.1 **q-gram of Sequence**

Given a sequence S*, its* q*-grams are obtained by sliding a window of length* q *over the*

letters of S, more specifically, the set of substrings: $\{S[i:i+q-1]|0 \leq i \leq |S|-q\}$. *For a sequence S, there are* $|S|-q+1$ *q-grams.*

6.3.1 q-gram Based Feature Extraction

In this chapter, q-grams are used to generate the features of protein sequences. For a protein sequence, there are a total of 8,000(=20^3) grams when q equals 3. Here, we present four kinds of feature extraction methods based on the q-grams of protein sequences.

The alphabet of a protein sequence comprises in all 20 letters: $\Sigma = \{C, S, T, P, A, G, N, D, E, Q, H, R, K, M, I, L, V, F, Y, W\}$. This means that there are in total $|\Sigma|^q = 20^q$ kinds of q-grams for a protein sequence, and we may arrange them according to lexicographic order, and use r_i to denote the ith q-gram in this order. The set of all the possible q-grams is denoted as: $\Re = \{r_0, r_1, \ldots, r_{20^q-1}\}$. In the following, we shall present four kinds of transformation methods for protein sequences based on q-grams. These methods are: *feature selection based on q-gram frequency*, *feature selection based on q-gram wavelet*, *feature selection based on q-gram similarity*, and *feature selection based on q-gram TF.IDF*.

Feature Selection Based on q-gram Frequency

The q-gram frequency based feature selection method is proposed for protein sequence by extending the concept of amino acid composition, dipeptide composition, and tri-peptide frequency encoding. The following definition describes the steps to transform the original sequence domain to frequency vector domain.

Definition 6.3.2 The q-gram Frequency Vector

Let $S=\{s_0, s_1, \ldots, s_{n-1}\}$ *be a protein sequence over the alphabet* Σ, *and the set of all possible q-grams be denoted as:* $\Re = \{r_0, r_1, \ldots, r_{|\Sigma|^q-1}\}$. *Then the q-gram frequency*

vector of S is defined as $FV^q(S)$=$[f_0, \ldots, f_{|\Sigma|^q-1}]$, where $f_i(\geq 0)$ corresponds to the occurrence frequency of the q-gram r_i in S, and $\sum_{i=0}^{|\Sigma|^q-1} f_i = |S| - q + 1 = n - q + 1$.

Feature Selection Based on q-gram Wavelet

A wavelet based method was proposed in [61] to transform sequences into a high-dimensional integer space. The number of dimensions is determined by alphabet size and number of wavelet coefficients. The wavelet transformation in [61] is based on the frequency of the letter itself. In contrast, wavelet transformation in our method is based on the frequency of q-grams. The definition of wavelet transformation based on q-grams is given below:

Definition 6.3.3 The q-gram Wavelet Vector

Let S=$\{s_0, s_1, \ldots, s_{n-1}\}$ be a protein sequence over the alphabet Σ, and the set of all the possible q-grams be denoted as: $\Re = \{r_0, r_1, \ldots, r_{|\Sigma|^q-1}\}$. Then the k-level wavelet transformation q-gram wavelet vector of S is defined as $W_k^q(S) = [v_{k,0}, \ldots, v_{k,\frac{n-q+1}{2^k}-1}]$, where $v_{k,i} = [A_{k,i}, B_{k,i}]$ for

$$A_{k,i} = \begin{cases} f(s_{i+q-1}) & k = 0 \\ A_{k-1,2i} + A_{k-1,2i+1} & 0 < k \leq log_2(n-q+1) \end{cases} \tag{6.1}$$

$$B_{k,i} = \begin{cases} 0 & k = 0 \\ A_{k-1,2i} - A_{k-1,2i+1} & 0 < k \leq log_2(n-q+1) \end{cases} \tag{6.2}$$

For sequence S, when $k = log_2(n-q+1)$, $A_{k,0}$ is the q-gram frequency vector $FV^q(S)$, and $B_{k,0}$=$FV^q(S[0 : \lceil \frac{n-q+1}{2} \rceil]) - FV^q(S[\lceil \frac{n-q+1}{2} \rceil : n-q+1])$. These two vectors represent the set of first and second wavelet coefficients. Note that the q-gram wavelet transformation is exactly the wavelet transformation presented in [61] when q equals 1.

Feature Selection Based on q-gram Similarity

Definition 6.3.4 The q-gram Similarity Vector

Let S=$\{s_0, s_1, \ldots, s_{n-1}\}$ be a protein sequence over the alphabet Σ, and the set of all the possible q-grams be denoted as: $\Re = \{r_0, r_1, \ldots, r_{|\Sigma|^q-1}\}$. Then the q-gram similarity vector of S can be denoted as $(w_0^S, w_1^S, \ldots, w_{|\Sigma|^q-1}^S)$, where the value w_i^S is calculated as the maximum similarity score between the q-gram r_i and the q-gram occurring in sequence S in terms of BLOSUM [1].

	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W
C	9	-1	-1	-3	0	-3	-3	-3	-4	-3	-3	-3	-3	-1	-1	-1	-1	-2	-2	-2
S	-1	4	1	-1	1	0	1	0	0	0	-1	-1	0	-1	-2	-2	-2	-2	-2	-3
T	-1	1	4	1	-1	1	0	1	0	0	0	-1	0	-1	-2	-2	-2	-2	-2	-3
P	-3	-1	1	7	-1	-2	-1	-1	-1	-1	-2	-2	-1	-2	-3	-3	-2	-4	-3	-4
A	0	1	-1	-1	4	0	-1	-2	-1	-1	-2	-1	-1	-1	-1	-1	-2	-2	-2	-3
G	-3	0	1	-2	0	6	-2	-1	-2	-2	-2	-2	-2	-3	-4	-4	0	-3	-3	-2
N	-3	1	0	-2	-2	0	6	1	0	0	-1	0	0	-2	-3	-3	-3	-3	-2	-4
D	-3	0	1	-1	-2	-1	1	6	2	0	-1	-2	-1	-3	-3	-4	-3	-3	-3	-4
E	-4	0	0	-1	-1	-2	0	2	5	2	0	0	1	-2	-3	-3	-3	-3	-2	-3
Q	-3	0	0	-1	-1	-2	0	0	2	5	0	1	1	0	-3	-2	-2	-3	-1	-2
H	-3	-1	0	-2	-2	-2	1	1	0	0	8	0	-1	-2	-3	-3	-2	-1	2	-2
R	-3	-1	-1	-2	-1	-2	0	-2	0	1	0	5	2	-1	-3	-2	-3	-3	-2	-3
K	-3	0	0	-1	-1	-2	0	-1	1	1	-1	2	5	-1	-3	-2	-3	-3	-2	-3
M	-1	-1	-1	-2	-1	-3	-2	-3	-2	0	-2	-1	-1	5	1	2	-2	0	-1	-1
I	-1	-2	-2	-3	-1	-4	-3	-3	-3	-3	-3	-3	-3	1	4	2	1	0	-1	-3
L	-1	-2	-2	-3	-1	-4	-3	-4	-3	-2	-3	-2	-2	2	2	4	3	0	-1	-2
V	-1	-2	-2	-2	0	-3	-3	-3	-2	-2	-3	-3	-2	1	3	1	4	-1	-1	-3
F	-2	-2	-2	-4	-2	-3	-3	-3	-3	-3	-1	-3	-3	0	0	0	-1	6	3	1
Y	-2	-2	-2	-3	-2	-3	-2	-3	-2	-1	2	-2	-2	-1	-1	-1	-1	3	7	2
W	-2	-3	-3	-4	-3	-2	-4	-4	-3	-2	-2	-3	-3	-1	-3	-2	-3	1	2	11

Table 6.1: BLOSUM62 Matrix

The BLOSUM matrix originates from a paper by Henikoff and Henikoff [51]. Different levels of the BLOSUM matrix can be created by differentially weighting the degree of similarity between sequences. For example, a BLOSUM62 matrix is calculated from protein blocks such that if two sequences are more than 62% identical, the contribution of these sequences is weighted to sum to one. In this way, the contributions of multiple entries of closely related sequences are reduced. The BLOSUM62 matrix is given in Table 6.1.

[1] BLOSUM stands for BLOcks SUbstitution Matrix.

For a protein sequence, the alphabet size for all the amino acids is 20. Hence, according to the definition of the q-gram similarity vector, each protein sequence is transformed into a $|\Sigma|^q (= 20^q)$-dimensional feature vector. The q-gram similarity vectors of the protein sequences can be obtained by using Algorithm 12.

Algorithm 12 q-gram Similarity Vector Extraction
Input: Protein sequences.
Output: q-gram Similarity Feature Vectors.

```
1: for each protein sequence S do
2:    for i = 0 to |Σ|^q − 1 do
3:       w_i^S ← −1000;
4:    end for
5:    for each q-gram r_i, 0 ≤ i < |Σ|^q − 1 do
6:       for each q-gram q_j^S in protein sequence S do
7:          w_i^S ← max(w_i^S, similarity(q_j^S, r_i));
8:       end for
9:       w_i^S ← w_i^S / similarity(r_i, r_i); //normalize the value w_i^S
10:   end for
11: end for
```

For sequences s and t of length m, the similarity function is defined as follows: $similarity(s,t)$=$\sum_{i=0}^{m-1}(blosumMatrix[s[i],t[i]])^2$. Note that for the mismatch operation, only replacement is considered in the proposed similarity function while insertion and deletion are disregarded.

Feature Selection Based on q-gram TF.IDF

We use a general text classification method to process a protein sequence for a feature vector. In text classification, the term frequency / inverse document frequency (TF.IDF) is commonly used to weight each word in the text document. The TF.IDF approach can capture relevancy among words, text documents and categories.

[2] $blosumMatrix[a,b]$ represents the similarity between a letter a and another letter b, which is shown in Table 6.1.

In our method, q-grams in protein sequences correspond to terms or words in text documents. Document frequency threshold is used to remove some q-grams that have little influence on the classification work. Finally, the feature values of the term are the term weights calculated by the simple TF.IDF method. Now we give the details of our algorithm in Algorithm 13.

Algorithm 13 q-gram TF.IDF Vector Extraction
Input: Protein sequences.
Output: q-gram TF.IDF Feature Vectors.

1: **for** each protein sequence S **do**
2: All the q-grams in S are generated;
3: **end for**
4: A hash-table is built on all the unique q-grams which occur in these protein sequences to maintain the q-gram inverted index $IIndex$;
5: Remove q-gram q_i from inverted index $IIndex$ if $df(q_i) < \epsilon$;
6: **for** each protein sequence S **do**
7: Compute the TF.IDF weight vector of all q-grams in inverted index $IIndex$ for S;
8: **end for**

For each protein sequence, its TF.IDF weight vector based on the q-grams inverted index $IIndex$ is computed by the following method:

$tfidf_{ij} = tf_{ij} \times \log \frac{N}{n_i}$, where

N: Number of protein sequences in the dataset;

n_i: Number of protein sequences containing the q-gram q_i;

tf_{ij}: the term weight of q-gram q_i in the protein sequence S_j.

In Algorithm 13, document frequency threshold ϵ is used to remove some features, (i.e., q-grams) that have little influence on the classification work. The number of documents that contain a certain q-gram q_i is denoted as $df(T_i)$, and the q-gram q_i is removed if $df(T_i) < \epsilon$. The threshold can be adjusted to meet the requirement of the features for the classifier.

6.3.2 Support Vector Machine

The SVM algorithm [20, 114] is a classification algorithm that gives better performance in a very large number of application domains including text classification, face detection, object recognition, speaker identification and handwriting recognition [89]. In recent years, the SVM learning algorithm has been extensively applied in the field of bioinformatics. Due to their ability to handle noise, high-dimensional data, and a large dataset [123], SVMs have become a new generation of machine learning algorithms that are popular in the analysis of biological problems such as gene and tissue classifications from microarray expression data, protein fold recognition, protein interaction prediction, protein secondary structure prediction, and protein localization prediction.

The key idea of SVM is that a classifier must not only work well on training samples, but also equally well on testing unknown samples. SVM is a learning algorithm [37] which learns a classifier from a set of positively and negatively labeled training dataset. The learned classifier can be used to classify new unlabeled test samples.

SVMs assume that all data are represented as feature vectors in feature space. Given a set of training data labeled with two classes, SVMs operate by finding a hyperplane in the space of the training dataset. This hyperplane separates positive samples from negative samples. The separation is done so that the distance from the hyperplane to the nearest of the positive and negative samples is the largest. Intuitively, this makes the classification correct for testing data that is near but not identical to the training data. In all, the goal of SVM is to find the optimal hyperplane with the maximum margin of separation. Figure 6.1 shows a very simple example of the SVM classifier.

We start with the simplest case: linear machines trained on separable data

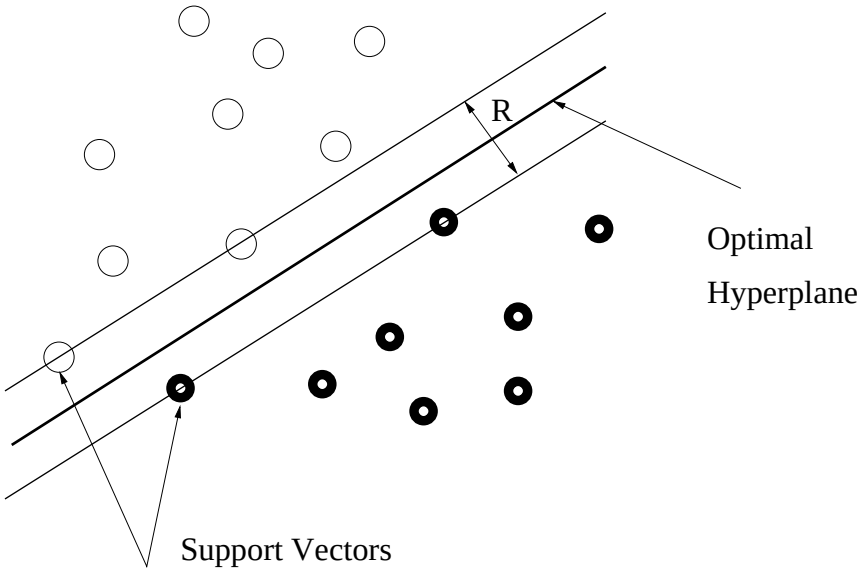

Figure 6.1: An Example of SVM Classifier

(as we shall see, the analysis for the general case – nonlinear machines trained on non-separable data – results in a very similar quadratic programming problem). Suppose we have a labeled training set, $\{X_i, y_i\}$, $i = 1, \ldots, N$, $y_i \in \{-1, 1\}$, $X_i \in R^k$. For each training sample X_i, we assign a target value or class label y_i with the value of -1 or 1. For the linearly separable case, the support vector algorithm simply looks for the separating hyperplane with the largest margin. This can be formulated as follows:

$$X_i \cdot w + b \geq +1 \; for \; y_i = +1 \tag{6.3}$$

$$X_i \cdot w + b \leq -1 \; for \; y_i = -1 \tag{6.4}$$

or

$$y_i * (X_i \cdot w + b) - 1 \geq 0, \; for \; i = 1, \ldots, N \tag{6.5}$$

In Equations (3),(4) and (5), w is the weight vector and b is the bias. The following formula is to find the optimal hyperplane in terms of w and b, where the

parameter C is the constant which controls the trade-off between maximizing the margin and minimizing error.

$$\begin{aligned} min \quad & (1/2w^T \cdot w + C \sum_{i=1}^{N} \alpha_i) \\ s.t. \quad & y_i * (X_i \cdot w + b) - 1 + \alpha_i \geq 0 \\ & \alpha_i \geq 0, \ for \ i = 1, \ldots, N \end{aligned} \tag{6.6}$$

In our study, we use SVMlight to predict the subcellular localization of proteins based on the proposed feature selection method. This software can be freely downloaded for academic use from http://www.cs.cornell.edu/People/tj/svm_light/. SVMlight is an implementation of Vapnik's SVM [113] for the problems of pattern recognition, regression, and learning a ranking function.

For a given dataset, SVMs are trained in the transformed feature space with different kernel functions and different values of parameter C. SVMs can handle very high-dimensional feature spaces efficiently and can effectively avoid overfitting by controlling the margin, and can automatically identify a small subset of support vectors.

6.4 Classifier Evaluation Method

The evaluation criteria for protein subcellular localization prediction are described and defined in this section.

6.4.1 The k-fold Cross Validation Method

Protein subcellular localization prediction is a multi-class classification problem, which can be handled by a series of binary classifiers. When the sample size is small, we cannot afford to use an independent test set; instead k-fold cross validation [76]

is the usual choice for assessing the performance of the classifier. In k-fold cross-validation, the data is divided into k subsets of (approximately) equal size. In this chapter, we use SVM for the prediction of subcellular localizations of Gram-negative proteins.

For each localization site, protein sequences with the class label are randomly divided into k groups. The SVMs are trained k times for each localization site, each time leaving out one of the subsets from training, but using only the omitted subset to compute whatever error criterion interests us. The basic flow chart of protein subcellular localization prediction is plotted in Figure 6.2. In the experimental study of this work, the value of k is set to 5.

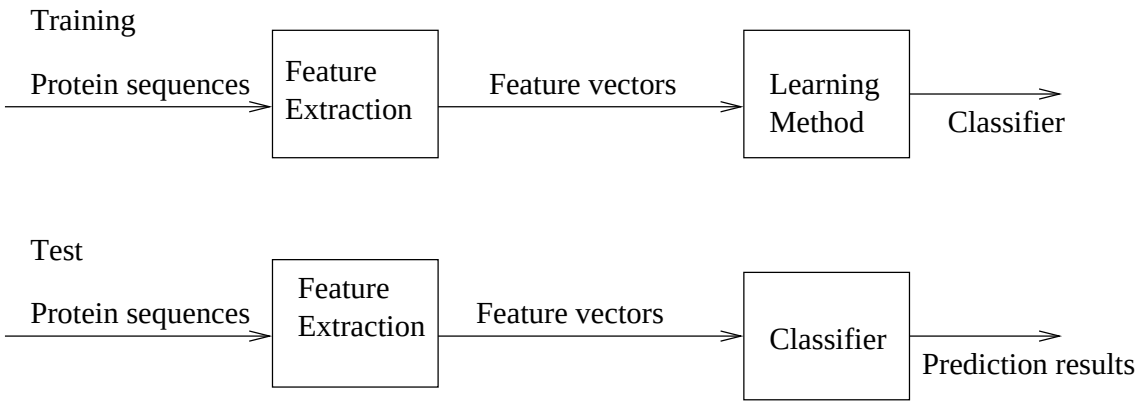

Figure 6.2: Flow Chart of Protein Subcellular Localization Prediction

6.4.2 Classifier Evaluation Measurement

The performance of a classifier is usually measured by average classification accuracy, precision, recall, specificity and Matthew's correlation coefficient (MCC) [73] based on the k-fold cross validation method for each localization site. The evaluation of the classifier for the subcellular localization site is based on the following quantities: number of true positives $TP(i)$, which is the number of correctly predicted protein sequences of the localization site i; number of true negatives $TN(i)$,

which is the number of correctly predicted proteins not of localization site i; number of false positives $FP(i)$, which is the number of incorrectly predicted proteins of the localization site i; and number of false negatives $FN(i)$ which is the number of incorrectly predicted proteins not of the localization site i:

$$precision(i) = \frac{TP(i)}{TP(i)+FP(i)},$$

$$recall(i) = \frac{TP(i)}{TP(i)+FN(i)},$$

$$specificity(i) = \frac{TN(i)}{TN(i)+FP(i)},$$

$$accuracy(i) = \frac{TP(i)+TN(i)}{TP(i)+FP(i)+TN(i)+FN(i)},$$

$$MCC(i) = \frac{TP(i)\times TN(i)-FP(i)\times FN(i)}{\sqrt{(TP(i)+FN(i))(TP(i)+FP(i))(TN(i)+FP(i))(TN(i)+FN(i))}}.$$

Here, N is the total number of protein sequences in all the localization sites, and m is the number of subcellular localization sites for the proteins in the experiments.

6.5 Dataset

The prediction performance is mainly related to two factors: number of protein sequences for training and number of target subcellular localizations covered by the dataset. While many bacteria have only three primary localization sites, Gram-negative bacteria have five primary localization sites: cytoplasm, cytoplasmic membrane, periplasm, outer membrane and extracellular space. In this chapter, we only consider the subcellular localization prediction of Gram-negative bacteria.

The dataset used to evaluate the effectiveness of the proposed prediction method is PSORTdb Dataset [46], available at http://www.psort.org/dataset/. PSORTdb is a new database of bacterial protein subcellular localizations that contains both

Positive	# of sequences	Negative	#of sequences
outer	391	Non-outer	1052
cyto	278	Non-cyto	1166
cm	309	Non-cm	1134
extra	190	Non-extra	1253
peri	276	Non-peri	1167

Table 6.2: Dataset

information determined through laboratory experimentation (ePSORTdb dataset) and computational predictions (cPSORTdb dataset). This dataset has been generated from the SWISS-PROT database with the extraction of all Gram-negative proteins with annotated subcellular locations. The final dataset used in our experiments consists of 1444 Gram-negative bacterial proteins of experimentally determined localizations, and all these proteins reside in a single localization site. For each localization, the relevant dataset consisting of proteins with specific subcellular localizations is denoted as a positive dataset, and the remaining proteins with other four subcellular localizations are regarded as the negative dataset. For example, when the localization site of outer membrane is considered, there are in total 391 proteins labeled with '+1' which reside at the outer membrane, and the remaining 1052 proteins which reside at cytoplasmic, cytoplasmic membrane, periplasmic and extracellular are labeled with '-1'. The datasets used in the experimental study are shown in Table 6.2.

6.6 Experimental Results and Discussion

We evaluate the proposed prediction methods on the datasets introduced in Section 6.5. SVMs are trained in different transformed feature spaces with different kernels, different values for the parameter C, and different values of q from 1 to 3 in our work.

In this section, we first transform outer membrane proteins into their corresponding feature vectors by using our proposed feature transformation method. Then SVMs are trained on the feature vectors of the outer membrane proteins to investigate how the parameters of the SVMs influence prediction performance. The five evaluation values: precision, recall, accuracy, specificity and MCC are computed by the average values over the five-fold cross validation. Note that the five folds for different proteins are generated only once and are used by all the proposed methods for the fairness.

According to the prediction results for outer membrane proteins, a group of parameter settings are selected for different transformation methods. Then we use these parameter settings on the other proteins with different subcellular locations.

6.6.1 Parameters Selection

In this section, we present a group of experiments on outer membrane proteins that investigate the impact of the parameter q and SVM parameters on prediction. The best parameters for various transformed methods are selected according to the prediction results for proteins in the location of outer membrane. The best results are selected to maximize the precision of prediction while keeping the corresponding recall and other evaluation measurements acceptable.

Table 6.3 reports the performance of the SVM classifier for outer membrane proteins, and the classifiers are built on the q-gram frequency feature space with different values of q, different kernel functions and the parameter C. When q=1, it is equal to the amino acid composition method proposed in [55]. When $q = 2$, it is equal to the amino acid pairs method in [92]. The results in Table 6.3 shows that we can achieve better prediction performance by using amino acid composition instead of amino acid pairs (or dipeptide composition [56]). Moreover, for q=2, the

q	Kernel	C	Precision	Recall	Accuracy	Specificity	MCC
1	Linear	default	76.77%	61.54%	84.51%	93.05%	0.588384
1	Linear	1	75.17%	63.59%	84.44%	92.19%	0.589732
1	Linear	10	74.65%	64.61%	84.44%	91.80%	0.591928
1	Polynomial	default	84.6%	72.82%	89.1%	95.14%	0.714005
1	Polynomial	1	79.82%	80.77%	89.24%	92.38%	0.728939
1	Polynomial	10	79.40%	81.03%	89.09%	92.09%	0.727372
1	RBF	default	89.64%	76.92%	91.39%	96.76%	0.774896
1	RBF	1	91.2%	79.74%	92.43%	97.14%	0.803515
1	RBF	10	88.22%	84.35%	92.70%	95.80%	0.813284
2	Linear	default	93.85%	23.08%	78.75%	99.43%	0.396009
2	Linear	1	75.51%	76.67%	86.94%	90.76%	0.671110
2	Linear	10	73.38%	6.92%	86.18%	89.62%	0.655921
2	Polynomial	default	86.78%	58.46%	86.39%	96.76%	0.634292
2	Polynomial	1	78.63%	82.82%	89.24%	91.62%	0.732671
2	Polynomial	10	78.63%	82.82%	89.24%	91.62%	0.732671
2	RBF	NaN					

Table 6.3: Results Based on q-gram Frequency Transformation for Outer Membrane Proteins

precision of the prediction nearly approaches zero when the RBF kernel is used, and we use 'NaN' to stand for the failure in this work. We do not further discover the prediction result for q=3 since the results show that when the dimension of the feature vector increases, performance will decrease a little. We also observe that the best results are achieved with RBF kernel, C=1 or 10, and q=1 in the case of the q-gram based frequency vector.

In Table 6.4, the results of SVM prediction performance based on the q-gram wavelet feature vector are described for the proteins residing in the outer membrane. We observe that for most of the cases with the same kernel function, the parameter C and $q = 1$, the q-gram wavelet feature vector can achieve better precision, accuracy, specificity and MCC but not recall. It outperforms the method based on the q-gram frequency feature vector when q=2 for all the evaluation values. It also cannot work in the case where the kernel function is RBF and q=2. The best results are achieved with parameter settings similar to those in the method

q	Kernel	C	Precision	Recall	Accuracy	Specificity	MCC
1	Linear	default	75.92%	62.31%	84.44%	92.67%	0.587525
1	Linear	1	76.24%	69.23%	85.76%	91.91%	0.631085
1	Linear	10	76.24%	69.23%	85.76%	91.91%	0.631085
1	Polynomial	default	85.72%	72.05%	89.17%	95.52%	0.716071
1	Polynomial	1	85.17%	85.9%	92.15%	94.48%	0.801536
1	Polynomial	10	85.17%	85.9%	92.15%	94.48%	0.801536
1	RBF	default	95.92%	65.9%	90%	98.95%	0.739688
1	RBF	1	95.85%	74.36%	92.16%	98.76%	0.797262
1	RBF	10	95.53%	77.44%	92.92%	98.67%	0.816776
2	Linear	default	95.89%	26.92%	79.86%	99.52%	0.438448
2	Linear	1	78.26%	80.77%	88.68%	91.62%	0.717044
2	Linear	10	78.26%	80.77%	88.68%	91.62%	0.717044
2	Polynomial	default	89.43%	61.8%	87.64%	97.24%	0.671987
2	Polynomial	1	80.65%	83.33%	90.07%	92.57%	0.751473
2	Polynomial	10	80.65%	83.33%	90.07%	92.57%	0.751473
2	RBF	NaN					

Table 6.4: Results Based on q-gram Wavelet Transformation for Outer Membrane Proteins

based on q-gram frequency feature vectors, where we use the RBF kernel function, C=1 or 10, and q=1: the precision is greater than 95%, with an acceptable recall (greater than 74%).

Table 6.5 shows the effects of the parameter settings on prediction performance when the q-gram TF.IDF transformation method is applied. The best result can be achieved when we use linear kernel function, C=1 or 10, and q=3. The experimental results also imply that the kernel function RBF does not work well in feature space of higher dimensions.

The prediction results based on q-gram similarity transformation for outer membrane proteins are shown in Table 6.6. Only the results for q=3 are reported since the q-gram similarity feature vector cannot represent well the information in the amino acid sequence when q is small (q=1 or 2). In other words, the dimension of the q-gram similarity vector is 20 and 400 respectively for q=1 and 2, and the length of protein sequences in the dataset is normally about hundreds of amino

q	Kernel	C	Precision	Recall	Accuracy	Specificity	MCC
1	Linear	default	75.92%	62.31%	84.44%	92.67%	0.587525
1	Linear	1	80.92%	38.97%	81.04%	96.67%	0.469620
1	Linear	10	76.10%	68.97%	85.69%	91.90%	0.628775
1	Polynomial	default	85.71%	72.05%	89.17%	95.52%	0.716071
1	Polynomial	1	85.17%	85.90%	92.15%	94.48%	0.801536
1	Polynomial	10	84.25%	54.62%	84.24%	95.24%	0.568523
1	RBF	default	87.75%	49.74%	83.96%	96.67%	0.533609
1	RBF	1	84.56%	63.85%	87.01%	95.62%	0.654435
1	RBF	10	86.70%	73.85%	89.93%	95.90%	0.735211
2	Linear	default	83.01%	32.82%	80%	97.52%	0.434879
2	Linear	1	77.76%	74.36%	87.36%	92.19%	0.67481
2	Linear	10	77.30%	73.08%	86.94%	92.10%	0.663395
2	Polynomial	default	86.68%	61.03%	86.88%	96.48%	0.649998
2	Polynomial	1	83.28%	76.41%	89.51%	94.38%	0.727726
2	Polynomial	10	83.28%	76.41%	89.51%	94.38%	0.727726
2	RBF	NaN					
3	Linear	default	94.06%	67.69%	90.07%	98.38%	0.741100
3	Linear	1	93.47%	75.64%	91.94%	98%	0.791047
3	Linear	10	93.47%	75.64%	91.94%	98.00%	0.791047
3	Polynomial	default	87.11%	5.64%	74.10%	99.52%	0.166650
3	Polynomial	1	87.36%	30.00%	79.86%	98.38%	0.432163
3	Polynomial	10	87.36%	30.00%	79.86%	98.38%	0.432163
3	RBF	NaN					

Table 6.5: Results Based on q-gram TF.IDF Transformation for Outer Membrane Proteins

acids, so most q-grams would have at least an occurrence in the protein sequence. According to the definition of q-gram similarity vector, every bit in the vector is set as 1 with high probability. So most of the protein sequences would be transformed into the same feature vector when q equals 1 or 2. We observe that the best result with high precision and recall based on q-gram similarity transformation is obtained when we use a polynomial kernel function, q=3, and C=1.

q	Kernel	C	Precision	Recall	Accuracy	Specificity	MCC
3	Linear	default	NaN				
3	Linear	1	90.05%	81.28%	92.5%	96.67%	0.805989
3	Polynomial	default	89.57%	52.05%	85.28%	97.62%	0.604546
3	Polynomial	1	91.37%	81.80%	92.99%	97.14%	0.818453

Table 6.6: Results Based on q-gram Similarity Transformation for Outer Membrane Proteins

6.6.2 Prediction Results for All Protein Subcellular Localizations

After the parameter settings are selected for different q-gram based transformation methods to achieve the best prediction performance according to the outer membrane proteins, we conduct the experimental study on all the proteins subcellular locations. The results depicted in Table 6.7 show that the prediction methods based on q-gram wavelet transformation, q-gram similarity transformation, and q-gram TF.IDF transformation all can achieve a precision higher than the prediction method based on q-gram frequency transformation for all the five datasets used in our experiments. However, the method based on q-gram frequency transformation can achieve higher recall. Therefore, we conduct the experiments to combine the feature vectors of different transformations to see if we can achieve both high prediction precision and recall.

6.6.3 Classification on Combined Feature Vectors

In Table 6.7, we observe that the method based on q-gram frequency transformation can achieve higher recall, while the methods based on the other three transformation methods achieve higher precision. Therefore in this section, we shall combine the features of q-gram frequency transformation with the features based on other transformation methods to see if we can achieve both higher precision and

localization	method	q	precision	recall	accuracy	specificity	MCC
outer	Frequency	1	88.22%	84.36%	92.71%	95.81%	0.813284
outer	DWT	1	95.53%	77.44%	92.92%	98.67%	0.816776
outer	Similarity	3	91.37%	81.79%	92.99%	97.14%	0.818453
outer	TF.IDF	3	93.47%	75.64%	91.95%	98%	0.791047
cyto	Frequency	1	81.10%	65.82%	90.50%	96.39%	0.675487
cyto	DWT	1	89.03%	56.36%	90.28%	98.28%	0.657381
cyto	Similarity	3	81.83%	74.91%	92.01%	96.05%	0.734141
cyto	TF.IDF	3	82.69%	56.36%	89.38%	97.17%	0.625097
cm	Frequency	1	98.09%	81.64%	95.76%	99.56%	0.870624
cm	DWT	1	100.00%	73.12%	94.31%	100.00%	0.825489
cm	Similarity	3	95.25%	85.57%	96.04%	98.85%	0.878768
cm	TF.IDF	3	100.00%	77.05%	95.14%	100.00%	0.851551
extra	Frequency	1	65.71%	67.37%	91.11%	94.72%	0.613877
extra	DWT	1	96.27%	56.32%	93.96%	99.68%	0.708682
extra	Similarity	3	86.77%	61.05%	93.61%	98.56%	0.694617
extra	TF.IDF	3	89.92%	51.05%	92.85%	99.20%	0.641817
peri	Frequency	1	74.17%	67.27%	88.96%	94.08%	0.637714
peri	DWT	1	89.37%	52.36%	89.58%	98.37%	0.631646
peri	Similarity	3	74.30%	62.91%	88.68%	94.76%	0.615731
peri	TF.IDF	3	85.82%	50.91%	89.03%	98.03%	0.606374

Table 6.7: Results for Different Transformation Based on q-grams for All Protein Subcellular Localizations

recall. Since features of q-gram wavelet transformation is a superset of features of q-gram frequency transformation, we will not consider the features of q-gram wavelet transformation here. In the following, we investigate the effect of the combined feature vector of q-gram frequency transformation and q-gram similarity transformation, the combined feature vector of q-gram frequency transformation and q-gram TF.IDF transformation, and also the combined feature vector of the three transformation methods.

SIM+: Combination of Frequency and Similarity

We first combine the feature vectors of q-gram frequency transformation and q-gram similarity transformation, and train the SVMs on the combined feature vectors to

investigate the effect on prediction performance. We use "SIM+" to denote the combined method. For q-gram frequency transformation, we set q=1 based on results obtained in Section 6.6.1. Similarly, for q-gram similarity transformation, q is set to 3 according to the parameter selection in Section 6.6.1. Moreover, although RBF kernel function is good to train the SVMs on q-gram frequency feature vectors, RBF kernel does not work on similarity feature vectors; we therefore use polynomial kernel function when the SVMs are trained on the combined feature vectors. We summarize the results of prediction based on "SIM+" in Table 6.8. By comparing the overall evaluation measurement of prediction MCC in Table 6.8 to MCC values in Table 6.7 for both frequency transformation and similarity transformation, we observe that the MCC value for the combined method is higher than MCCs for the two separate methods, Frequency and Similarity for protein localizations except for protein localization *cyto*. Furthermore, the result also shows that higher recall can be achieved by using the combined methods while keeping high precision of prediction for all the five protein localizations.

localization	precision	recall	accuracy	specificity	MCC
outer	91.25%	82.05%	92.99%	97.05%	0.819088
cyto	81.28%	72.36%	91.53%	96.05%	0.715748
cm	94.93%	85.90%	96.04%	98.77%	0.878851
extra	86.47%	61.58%	93.61%	98.48%	0.695968
peri	74.26%	66.18%	89.03%	94.42%	0.633788

Table 6.8: Results on Combined Method SIM+ for All Protein Subcellular Localizations

TF.IDF+: Combination of Frequency and TF.IDF

We also combine the feature vectors of q-gram frequency transformation and q-gram TF.IDF transformation, and train the SVMs on the combined feature vectors to investigate the effect on performance of prediction. In brief, we denote this combined method as "TF.IDF+". According to the analysis of parameter selection

in Section 6.6.1, we use linear kernel to train SVMs on the combine features; we set q=1 and 3 for frequency transformation and similarity transformation, respectively.

The prediction based on TF.IDF+ method is summarized in Table 6.9. We make the following two observations. Firstly, the result of prediction based on this combined method is mainly dominated by TF.IDF transformation instead of frequency transformation. Secondly, the precision of the combined method is similar to that of the method based on TF.IDF transformation, while a slightly higher recall is achieved by using the combined method than the method based on TF.IDF transformation.

localization	precision	recall	accuracy	specificity	MCC
outer	93.49%	75.90%	92.01%	98.00%	0.792867
cyto	82.90%	58.55%	89.72%	97.08%	0.639909
cm	100.00%	77.38%	95.21%	100.00%	0.853695
extra	89.99%	51.58%	92.92%	99.20%	0.645579
peri	85.55%	52.00%	89.10%	97.94%	0.612241

Table 6.9: Results on Combined Method TF.IDF+ for All Protein Subcellular Localizations

SIM+TF.IDF: Combination of Similarity and TF.IDF

Since the result based on the two combined methods SIM+ and TF.IDF+ are not dominated by Frequency transformation, we combine feature vectors of similarity transformation and TF.IDF transformation together and investigate the effect on protein localization prediction.

According to the analysis of parameter selection in Section 6.6.1, we again use linear kernel to train SVMs on the combine features; we set q=3 for both similarity transformation and TF.IDF transformation. We call the combined method based on similarity feature vector and TF.IDF feature vector as "SIM+TF.IDF".

The results based on SIM+TF.IDF combined method are shown in Table 6.10.

In Table 6.10, we notice that the recall of prediction based on SIM+TF.IDF is higher than that based on TF.IDF transformation vectors, and the precision of prediction is higher than that based on Similarity transformation vectors. The result indicates that combined feature vectors of similarity transformation and TF.IDF can represent the protein sequence better than the features vectors of single transformation method.

localization	precision	recall	accuracy	specificity	MCC
outer	94.51%	81.03%	93.54%	98.19%	0.833875
cyto	84.22%	65.09%	91.04%	97.17%	0.689362
cm	99.24%	80.66%	95.76%	99.82%	0.870857
extra	92.00%	55.79%	93.61%	99.36%	0.683450
peri	83.91%	58.91%	89.93%	97.25%	0.647476

Table 6.10: Results on Combined Method SIM+TF.IDF for All Protein Subcellular Localizations

SIM+TF.IDF+: Combination of Frequency, Similarity and TF.IDF

We also combine the feature vectors of Frequency transformation, similarity transformation and TF.IDF transformation and conduct the protein subcellular localization prediction based on the combined feature vectors. We called the combined method as "SIM+TF.IDF+". The result of protein subcellular prediction is shown in Table 6.11. Compared to the results (In Table 6.10) of prediction based on the combined method SIM+TF.IDF, the result based on SIM+TFIDF+ can achieve higher precision, recall, accuracy, specificity and MCC for all the protein localizations except proteins locating in "extra". Compared to all the other combined methods, SIM+TF.IDF+ can achieve much higher recall while keeping very high precision as well for protein subcellular localization prediction. Since more transformation methods are combined in method SIM+TF.IDF+, much more information of protein sequences is used on protein subcelluar localization prediction. It

therefore leads to higher accuracy of prediction on protein subcelluar localization prediction.

localization	precision	recall	accuracy	specificity	MCC
outer	94.53%	81.28%	93.61%	98.19%	0.835720
cyto	83.63%	65.82%	91.04%	97.00%	0.690526
cm	99.25%	80.98%	95.83%	99.82%	0.873019
extra	91.29%	55.79%	93.34%	99.28%	0.679932
peri	84.50%	59.27%	90.07%	97.34%	0.652985

Table 6.11: Results on Combined Method SIM+TF.IDF+ for All Protein Subcellular Localizations

6.7 Summary

In this chapter, q-gram frequency transformation, q-gram wavelet transformation, q-gram similarity transformation, and q-gram TF.IDF transformation have been proposed and described. These four transformation methods are then used to extract feature vectors from subcellular localization proteins. To build a system for protein subcellular localization prediction, transformed feature vectors are used to train SVMs. The experimental results show that q-gram based features can represent a protein sequence well, and they can be very useful in protein subcellular localization prediction. Since no single method of prediction can achieve high prediction accuracy, precision or recall for all protein subcellular localizations, we believe our proposed prediction method is a new investigation approach in research on protein subcellular localization prediction.

CHAPTER 7

Conclusion

In this chapter, we summarize the contributions of this thesis and discuss the possible directions of future work on sequence approximate matching in genomic sequence databases.

7.1 Summary of Contributions

In this thesis, we have studied three research problems – DNA sequence similarity search in sequence database, DNA sequence approximate join, and protein subcellular localization prediction, which are all related to sequence approximate matching in genomic databases. Our proposed search model and index structure are very effective in organizing a large genomic sequence database. Similarly, our proposed novel filtering algorithms are very efficient in processing approximate sequence matching, and the proposed q-gram based feature vectors extracted from protein sequence are helpful in predicting the subcellular localization of protein

sequences. We summarize our main contributions as follow.

7.1.1 DNA Sequence Similarity Search

Hash-based Pier Model

We have proposed a similarity search model, the hash-based pier model for DNA sequence similarity search. The pier model only extracts piers from a genomic database and eradicates any need to search the entire database. The sensitivity of the pier model is analyzed theoretically. The experimental results show that the size of the hash table in the pier model is relatively small compared to the original genomic database, and similarity search based on the pier model can be conducted very efficiently with high sensitivity. Compared to the most widely used DNA database search tool, BLASTn, the hash-based pier method is faster and requires less memory and disk space because our hash-based pier model simply extracts piers and hashes them instead of processing each segment in the sequence database as BLASTn and other methods do. Moreover, the GPM-based method has been presented to further improve search efficiency by reducing the computation cost of candidate verification. The hash-based pier model is very efficient and effective for similarity search in very large genomic databases owing to the idea of 'piers'. In addition, the proposed pier model is also applicable to other similarity search methods for selecting some segments as index terms.

Two-level Index Structure

We have devised an indexing structure, the two-level index comprising the hash table and c-trees, to support efficient similarity search in DNA sequence databases based on the q-grams of DNA sequences. At the first level, a hash table is built on the sequences in terms of qClusters, which are a group of clusters generated

on the q-grams. At the second level, a novel data structure called *c-trees* and constructed on a group of c-signatures of the DNA segments is used to support DNA sequence similarity search. The proposed filter principle based on qClusters and c-signatures guarantees high efficiency and high sensitivity in sequence search. We have carried out a series of experiments to evaluate the performance of the two-level index structure in terms of sensitivity, effectiveness and efficiency and accuracy. The experimental results are consistent with the theoretical analysis of the q-gram based filtering techniques and the sensitivity analysis of the proposed search model based on the two-level index structure.

7.1.2 DNA Sequence Approximate Join

We have studied the notion and operation of approximate sequence join in genomic databases. Following these, we have proposed a filter-and-refine sequence join algorithm for DNA sequence approximate join. In the filtering phase, sequences that are not joinable are pruned away rapidly. The refinement phase employs an efficient algorithm to remove the remaining false positives. The proposed scheme employs the precedence count matrix (PCM) to approximate the edit distance between two DNA sequences efficiently. We have evaluated the proposed sequence join algorithm, and our experimental study shows that our algorithm outperforms existing ones such as FV [61], q-gram and brute force by a wide margin in DNA sequence approximate join.

7.1.3 Protein Subcellular Localization Prediction

We have proposed using q-grams in protein sequences to predict the subcellular localization of proteins. The q-gram frequency vector, q-gram wavelet vector, q-gram similarity vector, and q-gram IDF.TF vector based on the q-grams of a protein

sequence are proposed and generated to capture information in protein sequences. These q-gram based vectors are then used as k-dimensional features for an SVM that searches for the hyperplane to separate the two classes. In the proposed method, the k-dimensional feature space can be one of four feature vector spaces: q-gram frequency vectors, q-gram wavelet vectors, q-gram similarity vectors, and q-gram IDF.TF vectors. We have implemented and compared the prediction systems based on the proposed feature vectors, and the results show that these feature vectors can represent a protein sequence well. Since there is no single method of prediction that can achieve high prediction accuracy, precision or recall for all the subcellular localizations for proteins, our proposed method is therefore substantial and useful in practice.

7.2 Future Work

While we have made a number of contributions in this thesis, we have also left a number of interesting questions unanswered, and they can be further explored in future work. Here, we suggest several interesting problems in the area of approximate sequence matching in genomic sequence databases.

First, for the similarity search problem, future work could consider the following:

- Our proposed search model and filtering methods are mainly about sequence similarity search in DNA sequence databases. For example, the novel two-level index structure is effective and efficient for DNA sequence similarity search, however it is limited to process sequences which are formed from a small alphabet size, such as DNA sequence data with an alphabet size of 4. For a larger alphabet size, such as a protein sequence with an alphabet size of 20, the two-level index structure on these q-grams would be much

larger than the original sequence database due to the large number of all the possible q-grams, so it is not applicable for indexing protein sequences. Since protein sequences allow more meaningful alignments with the use of scoring matrices (PAM or BLOSUM), we consider to propose some effective and efficient algorithms for searching protein sequences in protein sequence database in the future work.

Second, for the problem of DNA sequence approximate join, our work is confined to academic study; much work needs to be done before this approximate join method is deployed in real genomic applications in the area of computational biology, such as DNA sequence assembly and sequencing by hybridization.

Lastly, for the problem of prediction of subcellular localization of protein sequences, all our proposed prediction methods are based on q-grams in protein sequences. Though q-grams in protein sequences can represent the information in a protein sequence well, they are not enough for the prediction of protein subcellular localization. In the future, we can combine the feature vectors based on q-grams with other existing features extracted from the protein sequences. For example, we can combine our methods with homology analysis, identification of sorting signals and motifs, etc. into an expert system for the prediction of subcellular localization of protein sequences.

BIBLIOGRAPHY

[1] In *http://www.genome.gov*.

[2] In *http://www.ncbi.nlm.nih.gov/Genbank/genbankstats.html*.

[3] In *http://pir.georgetown.edu/*.

[4] In *http://www.ddbj.nig.ac.jp/*.

[5] In *ftp://ncbi.nlm.nih.gov/blast/db/README*.

[6] In *http://www.gnu.org/manual/gprof-2.9.1/gprof.html*.

[7] Wu-blast. In *http://blast.wustl.edu/*.

[8] 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.

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

[10] A. Bairoch and R. Apweiler. The swiss-prot protein sequence data bank and its new supplement trembl. *Nucleic Acids Research*, 24:21–25, 1996.

[11] H. Bannai, Y. Tamada, O. Maruyama, K. Nakai, and S. Miyano. Extensive feature detection of n-terminal protein sorting signals. *Bioinformatics*, 18:298–305, 2002.

[12] S. Bedathur and J. Haritsa. Engineering a fast online persistent suffix tree construction. In *Proc. 2004 Int. Conf. Data Engineering (ICDE'04)*, pages 720–731, Boston, USA, Mar. 2004.

[13] S. Bedathur and J. Haritsa. Search-optimized persistent suffix tree storage for biological applications. In *Technical Report TR-2004-04*, Database System Lab, Supercomputer Education and Research Center, Indian Institute of Sicence, Bangalore 560012, India, 2004.

[14] S. Begley and A. Rogers. It's all in the genes: in compuational biology, scientists track elusive dna stands through databases. *Newsweek*, page 64, 1994.

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

[16] 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.

[17] D.A. Benson, I. Karsch-mizrachi, D.J. Lipman, J. Ostell, and D.L. Wheeler. Genbank update. *Nucleic Acids Research*, 32:D23–CD26, 2004.

[18] M. Bhasin and G.P.S. Raghava. Eslpred: Svm-based method for subcellular localization of eukaryotic proteins using dipeptide composition and psi-blast. *Nucleic Acids Research*, 32:w414–w419, 2004.

[19] B. Boeckmann, A. Bairoch, R. Apweiler, M.C. Blatter, A. Estreicher, E. Gasteiger, M.J. Martin, K. Michoud, C. O'Donovan, I. Phan I, S. Pilbout, and M. Schneider. The swiss-prot protein knowledgebase and its supplement trembl in 2003. *Nucleic Acids Research*, 31:365–370, 2003.

[20] B.E. Boser, I.M. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *Annual Workshop on Computational Learning Theory archive, Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152, Pittsburgh, PA, July 1992.

[21] Brona Brejova, Daniel G. Brown, and Tomas Vinar. Vector seeds: an extension to spaced seeds allows substantial improvements in sensitivity and specificity. *Journal of Computer and System Sciences*, 70(3):364–380, 2005. Early version appeared in WABI 2003.

[22] D. Brown, M. Li, and B. Ma. A tutorial of recent developments in the seeding of local alignment. *Journal of Bioinformatics and Computational Biology*, 2(4):819–842, 2004.

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

[24] J. Buhler, U. Keich, and Y. Sun. Designing seeds for similarity search in genomic dna. In *Int. Conf. RECOMB*, pages 67–75, 2003.

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

[26] A. Califano and I. Rigoutsos. Flash: A fast look-up algorithm for string homology. In *Proc. of the Int. Conference on Intelligent Systems for Molecular Biology*, pages 56–64, Bethesda, MD, 1993.

[27] X. Cao, S.C. Li, B.C. Ooi, and A.K.H. Tung. Piers: An efficient model for similarity search in dna sequence databases. *ACM Sigmod Record*, 33, 2004.

[28] X. Cao, S.C. Li, and A.K.H. Tung. Indexing dna sequences using q-grams. In *Proc. of the 10th Int. Conf. on Database Systems for Advanced Applications (DASFAA'05)*, pages 4–16, China, 2005, Best Paper Award.

[29] X. Cao, B.C. Ooi, K.L. Tan, and A.K.H. Tung. The q-gram based protein subcellular localization prediction. In *Technical Report: School of Computing, National University of Singapore*, 2005.

[30] X. Cao, A.K.H. Tung, B.C. Ooi, K.L. Tan, and S.C. Li. String join using precedence count matrix. In *Proc. of the 16th Int. Conf. on Scientific and Statistical Database Management (SSDBM'04)*, pages 345–348, Greece, 2004.

[31] T. Chen and S. Skiena. Trie-based data structures for sequence assembly. In *Technical Report: Department of Computer Science*, Stony Brook N.Y, 1996.

[32] K.C. Chou and D.W. Elrod. Protein subcellular location prediction. *Protein Engineering*, 12:107–118, 1999.

[33] P. Clote and R. Backofen. *Computational Molecular Biology: An Introduction*. JOHN WILEY & SONS, LTD, 2000.

[34] W. Cohen. Integration of heterogeneous databases without common domains using queries based on textual similarity. In *Proc. of the 1998 ACM SIGMOD Conf. on Management of Data (SIGMOD'98)*, pages 201–212, 1998.

[35] F. Collins and D. Galas. A new five-year plan for the us human genome project. *Science*, 262:43–46, 1993.

[36] F.H.C. Crick. On protein synthesis. In *Symposium of the Society of Experimental Biology*, pages 12: 138–167, 1958.

[37] N. Cristianini and J. Shawe-Taylor. *An introduction to support vector machines and other kernel-based learning methods*. Cambridge University Press, Canbridge, UK, 2000.

[38] R.F. Doolittle, M.W. Hunkapiller, L.E. Hood, S.G. Devare, K.C. Robbins, S.A. Aaronson, and H.N. Antoniades. Simian sarcoma virus onc gene, v-sis, is derived from the gene (or genes) encoding a platelet-derived growth factor. *Science*, 221:275–277, 1983.

[39] F. Eisenhaber and P. Bork. Wanted: subcellular localization of proteins based on sequence. *Trans. Cell Biol.*, 8:169–170, 1998.

[40] O. Emanuelsson, H. Nielsen, S. Brunak, and G. von Heijne. Predicting subcellular localization of proteins based on their n-terminal amino acid sequences. *J. Mol. Biol.*, 300:1005–1016, 2000.

[41] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *Proc. 1994 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'94)*, pages 419–429, Minneapolis, Minnesota, May 1994.

[42] M. Farach, P. Ferragina, and S. Muthukrishnan. Overcoming the memory bottleneck in suffix tree construction. In *Proc. of IEEE Annual Symposium on Foundation of Computer Science*, 1998.

[43] P. Ferragina and G. Manzini. Opportunistic data structures with applications. In *Proc. of Symp. on Foundation of Computer Science*, pages 390–398, 2000.

[44] C. Fondrat and P. Dessen. A rapid access motif database (ramdb) with a search algorithm for the retrieval patterns in nucleic acids or protein databanks. *Computer Applications in the Biosciences*, 11(3):273–279, 1995.

[45] J.L. Gardy, M.R. Laird, F. Chen, S. Rey, C.J. Walsh, M. Ester, and Fiona S.L. Brinkman. Predicting of protein subcellular locations using fuzzy k-nn method. *Bioinformatics*, 21:617–623, 2005.

[46] J.L. Gardy, C. Spencer, K. Wang, M. Ester, G.E. Tusnady, I. Simon, S. Hua, K. deFays, C. Lambert, K. Nakai, and Fiona S.L. Brinkman. Psort-b: improving protein subcellular localization prediction for gram-negative bacteria. *Nucleic Acid Research*, 31:3613–3617, 2003.

[47] 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 *Int. Conf. RECOMB*, Japan, 2000.

[48] L. Gravano, P.G. Ipeirotis, H.V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. In *Proc. 2001 Int. Conf. Very Large Data Bases (VLDB'01)*, pages 491–500, Italy, Roma, Sept. 2001.

[49] R. Grossi and J.S. Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. In *Proc. 2000 ACM-SIAM Symp. Theory of Computing (STOC'00)*, Portland, Or, 2000.

[50] D. Gusfield. *Algorithms on Strings, Trees and Sequences, Computer Science and Computation Biology.* Cambridge University Press, New York, 1997.

[51] S. Henikoff and J.G. Henikoff. Amino acid substitution matrices from protein blocks. *Proc Natl Acad Sci*, 89:10915–10919, 1992.

[52] G.R. Hjaltason and H. Samet. Incremental distance join algorithms for spatial databases. In *Proc. of the 1998 ACM SIGMOD Conf. on Management of Data (SIGMOD'98)*, pages 237–248, 1998.

[53] W.K. Hon, T.W. Lam, W.K. Sung, W.L. Tse, C.K. Wong, and S.M. Yiu. Practical aspects of compressed suffix arrays and fm-index in searching dna sequences. In *Proc. of Workshops on Algorithm Engineering and Experiments*, 2004.

[54] http://www.ncbi.nlm.nih.gov/Class/MLACourse/Modules/MolBioReview. Molecular biology review.

[55] S. Hua and Z. Sun. Support vector machine approach for protein subcellular localization prediction. *Bioinformatics*, 17:721–728, 2001.

[56] Y. Huang and Y. Li. Predicting of protein subcellular locations using fuzzy k-nn method. *Bioinformatics*, 20:21–28, 2004.

[57] E. Hunt, M.P. Atkinson, and R.W. Irving. A database index to large biological sequences. In *Proc. 2001 Int. Conf. Very Large Data Bases (VLDB'01)*, pages 139–148, Roma, Italy, September 2001.

[58] R. Idury and M.S. Waterman. A new algorithm for dna sequence assembly. *Journal of Computational Biology*, 2:291–306, 1995.

[59] L. Jin, C. Li, and S. Mehrotra. Efficient similarity string joins in large data sets. In *Technical Report: Department of Information and Computer Science*, University of California, 2002.

[60] P. Jokinen and E. Ukkonen. Two algorithm for approximate string matching in static texts. In *Proc. of the 16th Symposium on Mathematical Foundataions of Computer Science*, pages 240–248, 1991.

[61] T. Kahveci and A. Singh. An efficient index structure for string databases. In *Proc. 2001 Int. Conf. Very Large Data Bases (VLDB'01)*, Roma, Italy, 2001.

[62] U. Keich, M. Li, B. Ma, and J. Tromp. On Spaced Seeds for Similarity Search. *Discrete Applied Mathematics*, 138(3):253–263, 2004.

[63] W.J. Kent. Blat - the blast-like alignment tool. *Genome Research*, 12:656–664, 2002.

[64] T.W. Lam, K. Sadakane, W.K. Sung, and S.M. Yiu. A space and time efficient algorithm for constructing compressed suffix arrays. In *Proc. of the 8th International Computing and Combinatorics Conference (COCOON'02)*, pages 401–410, 2002.

[65] Z. Lei and Y. Dai. A novel approach for prediction of protein subcellular localization from sequence using fourier analysis and support vector machine. In *Proc. of the 4th ACM SIGKDD Workshop on Data Mining in Bioinformatics (BIOKDD 2004)*, pages 265–274, Seattle, USA, 2004.

[66] V.I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 6:707–710, 1966.

[67] M. Li, B. Ma, D. Kisman, and J. Tromp. PatternHunter II: Highly Sensitive and Fast Homology Search. *Journal of Bioinformatics and Computational Biology*, 2(3):417–439, 2004. Early version in GIW 2003.

[68] C. Liebecq. *Biochemical nomenclature and related documents*. Portland Press, 2nd edition, 1992.

[69] Z. Lu, D. Szafron, R. Greiner, P. Lu, D.S. Wishart, B. Poulin, J. Anvik, C. Macdonell, and R. Eisner. Predicting subcellular localization of proteins using machine-learning classifiers. *Bioinformatics*, 20:547–556, 2004.

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

[71] U. Manber and G. Myers. Suffix arrays: a new method for on-line string searches. In *Proc. of the Fist Annual ACM-SIAM Symp. and Applied Mathematics*, pages 319–327, Philadelphia, USA, 1990.

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

[73] B.W. Matthews. Comparison of predicted and observed secondary structure of t4 phage lysozyme. *Biochim. Biophys. Acta*, 405:442–451, 1975.

[74] E.M. McCreight. A space-economical suffix tree construction algorithm. *Journal of the ACM*, 23:262–272, 1976.

[75] C. Meek, J.M. Patel, and S. Kasetty. Oasis: An online and accurate technique for local-alignment searches on biological sequences. In *Proc. 2003 Int. Conf. Very Large Data Bases (VLDB'03)*, pages 910–921, Berlin, Germany, Sept. 2003.

[76] T.M. Mitchell. *Machine learning*. McGraw-Hill, New York, 1997.

[77] David W. Mount. *Bioinformatics: Sequence and Genome Analysis*. Cold Spring Harbor Laboratory Press, 2001.

[78] S. Muthukrishnan and S.C. Sahinalp. Approximate nearest neighbors and sequence comparison with block operation. In *Proc. 2000 ACM-SIAM Symp. Theory of Computing (STOC'00)*, Portland, Or, 2000.

[79] R. Nair and B. Rost. Sequence conserved for subcellular localization. *Protein Science*, 11:2836–2847, 2002.

[80] K. Nakai and P. Horton. Psort: a program for detecting the sorting signals of proteins and predicting their subcellular localization. *Trends Biochem.*, 24:34–35, 1999.

[81] K. Nakai and M. Kanehisa. Expert system for predicting protein localization sites in gram-negative bacteria. *PROTEINS: Structure, Function, and Genetics*, 11:95–110, 1991.

[82] K. Nakai and M. Kanehisa. A knowledge base for predicting protein localization sites in eukaryotic cells. *Genomics*, 14:897–911, 1992.

[83] H. Nakashima and K. Nishikawa. Discrimination of intracellular and extracellular proteins using amino acid composition and residue-pair frequencies. *J Mol Biol.*, 238:54–61, 1994.

[84] G. Navarro. A guided tour to approximate string matching. *ACM Computing Surveys (CSUR)*, 33:31 – 88, 2001.

[85] S.B. Needleman and C.D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48:443–453, 1970.

[86] N. Neelapala, R. Mittal, and J. Haritsa. Spine: Putting backbone into string indexing. In *Proc. 2004 Int. Conf. Data Engineering (ICDE'04)*, pages 325–336, Boston, USA, Mar. 2004.

[87] H. Nielsen, J. Engelbrecht, S. Brunak, and G. von Heijne. A neural network method for identification of prokaryotic and eukaryotic signal peptides and prediction of their cleavage sites. *Int J Neural Syst.*, 8:581–599, 1997.

[88] Z. Ning, A.J. Cox, and J.C. Mullikin. Ssaha: A fast search method for large dna databases. *Genome Research*, 11:1725–1729, 2001.

[89] W.S. Noble. *Support vector machine applications in computational biology.* B. Schoelkopf, K. Tsuda and J.P. Vert, MIT Presse, 2004.

[90] C. O'Donovan, M.J. Martin, A. Gattiker, E. Gasteiger, A. Bairoch, and R. Apweiler. High-quality protein knowledge resource: Swiss-prot and trembl. *Brief Bioinform.*, 3(3):275–284, 2002 Sept.

[91] O. Ozturk and H. Ferhatosmanoglu. Effective indexing and filtering for similarity search in large biosequence datasbases. In *Third IEEE Symposium on BioInformatics and BioEngineering (BIBE'03)*, Bethesda, Maryland, 2003.

[92] K. Park and M. Kanehisa. Prediction of protein subcellular locations by support vector machines using compositions of amino acids and amino acid pairs. *Bioinformatics*, 19:1656–1663, 2003.

[93] W.R. Pearson. Searching protein sequences libraries: Comparison of the sensitivity and selectivity of the smith-waterman and fasta algorithm. *Genomics*, 11:635–650, 1991.

[94] W.R. Pearson. Protein sequence comparision and protein evolution. In *Tutorial T6 of Intelligent Systems in Mol. Biol.*, Cambridge, England, July 1995.

[95] W.R. Pearson. Effective protein sequence comparison. *Methods Enzymol*, 266:227–258, 1996.

[96] W.R. Pearson and D.J. Lipman. Improved tools for biological sequence comparison. *Proc. of the National Academy of Sciences*, 85:2444–2448, 1988.

[97] M. Peltola, H. Soderlund, J. Tarhio, and E. Ukkonen. Algorithms for some string matching problems arising in molecular genetics. In *Proc. of the 9th IFIP World Computer Congress*, pages 59–64, 1983.

[98] M. Peltola, H. Soderlund, and E. Ukkonen. Sequaid: a dna sequence assembly program based on a mathematical model. *Nucleic Acids Research*, 12:307–321, 1984.

[99] R. Ramakrishnan. *Database Management Systems.* McGraw-Hill, 1997.

[100] A. Reinhardt and T. Hubbard. Using neural networks for prediction of the subcellular location of proteins. *Nucleic Acids Research*, 26:2230–2236, 1998.

[101] C.M. Rice, R. Fuchs, D.G. Higgins, P.J. Stoehr, and G.N. Cameron. The embl data library. *Nucleic Acids Research*, 21:2967–2971, 1993.

[102] J.R. Riordan, J.M. Rommens, B. Kerem, N. Alon, R. Rozmahel, Z. Grzelczak, J. Zielenski, S. Lok, N. Plavsic, and J.L. Chou. Identification of the cystic

fibrosis gene: Cloning and characterization of complementary dna(in cystic fibrosis: Cloning and genetics). *Science*, 245:1066–1073, 1989.

[103] K. Sadakane. New text indexing functionalities of the compressed suffix arrays. *Journal of Algorithms*, 48:294–313, 2003. A preliminary version appears in ISAAC 2000.

[104] J.C. Setubal and J. Meidanis. *Introduction to computational molecular biology.* PWS Publishing Company, Boston, 1997.

[105] R. She, F. Chen, K. Wang, M. Ester, J.L. Gardy, and F.S. Brinkman. Frequent-subsequence-based prediction of outer membrane proteins. In *Proc. 9th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining (SIGKDD'03)*, Washington DC, USA, 2003.

[106] J. Sims, D. Capon, and D. Dressler. dnag(primase)-dependent origins of dna replication. *Journal of Biolical Chemistry*, 254:12615–12628, 1979.

[107] T.F. Smith and M.S. Waterman. Comparative biosequence metrics. *Journal of Molecular Evolution*, 18:38–46, 1981.

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

[109] B.S. Strauss. Book review: Dna repair and mutagenesis. *Science*, 270:1511–1513, 1995.

[110] Z. Tan, X. Cao, B.C. Ooi, and A.K.H. Tung. The ed-tree: an index for large dna sequence databases. In *Proc. 15th Int. Conf. on Scientific and Statistical Database Management*, pages 151–160, 2003.

[111] S. Tata, R.A. Hankins, and J.M. Patel. Practical suffix tree construction. In *Proc. 2004 Int. Conf. Very Large Data Bases (VLDB'04)*, Toronto, Canada, Sept. 2004.

[112] Y. Tian, S. Tata, R.A. Hankins, and J.M. Patel. Practical methods for constructing suffix trees. *The VLDB Journal*, 14:281–299, 2005.

[113] V.N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1995.

[114] V.N. Vapnik. *Statistical Learning Theory (Adaptive and Learning Systems for Signal Processing, Communications and Control)*. Wiley-Interscience, New York, 1998.

[115] T.K. Vintsyuk. Speech discrimination by dynamic programming. *Comput.*, 4:52–57, 1968.

[116] G. von Heijne, H. Nielsen, J. Engelbrecht, and S. Brunak. Identification of prokaryotic and eukaryotic signal peptides and prediction of their cleavage sites. *Protein Eng.*, 10:1–6, 1997.

[117] M.M. Waldrop. On-line achives let biologists interrogate teh genome. *Science*, 269:1356–1358, 1995.

[118] J.T.L. Wang, Q.H. Ma, D. Shasha, and C.H. Wu. Application of networks to biological data mining: a case study in protein sequence classification. In *Proc. 2000 Int. Conf. Knowledge Discovery and Data Mining (KDD'00)*, Boston, MA, 2000.

[119] M.D. Waterfield, G.T. Scrace, N. Whittle, P. Stroobant, A. Johnsson, A. Wasteson, B. Westermark, C.H. Heldin, J.S. Huang, and T.F. Deuel. Platelet-derived growth factor is structurally related to the putative transforming protein p28sis of simian sarcoma virus. *Nature*, 304:35–39, 1983.

[120] P. Weiner. Linear pattern matching algorithms. In *Proc. 14th IEEE Symp. On Switching and Automata Theory*, pages 1–11, 1973.

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

[122] R.W. Williams. The portable dictionary of the mouse genome: a personal database for gene mapping and molecular biology. *Mammalian Genome*, 5:372–375, 1994.

[123] N. Zavaljevski, F.J. Stevens, and J. Reifman. Support vector machines with selective kernel scaling for protein classification and identification of key amino acid positions. *Bioinformatics*, 18:689–696, 2002.