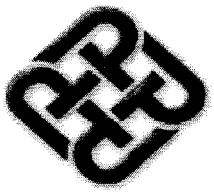


THE HONG KONG
POLYTECHNIC UNIVERSITY
香港理工大學

The Hong Kong Polytechnic University

Department of Computing

Genomic Sequence Search and Clustering Using Q-gram

Yuen, Man Chun

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Philosophy

March 2007

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written nor material which has been accepted for the ward of any other degree or diploma, except where due acknowledgement has been made in the text.

_____ (Signed)

Yuen Man Chun _____ (Name of Student)

ABSTRACT

With the advances in technologies, the amount of biological data such as DNA sequences and microarray data have been increased tremendously in the past decade. In order to obtain knowledge from the data, e.g., enhancing our understanding of the evolutionary changes and the causes of those severe diseases, one has to search for patterns from the databases of large size and high dimensionality. Information retrieval and data mining are powerful tools to extract information from the databases and/or information repositories. In the past several years, there have been attempts to apply these two branches of intelligent techniques to different bioinformatics applications. However, the performance of these existing techniques has not been optimized due to the characteristics of and requirements from biological data, e.g. extremely long genomic sequences with high dimensionality, and interpretable search/mining results.

In this thesis, we focus on how to improve the searching and the clustering performance in genomic sequence databases. A Q-gram based genomic search (QgramSearch) algorithm and a Q-gram based genomic sequence clustering (QgramClust) algorithm are proposed. Our QgramSearch can efficiently search the homologous database sequences to a query sequence. It makes use of two novel hashing techniques to enhance the efficiency of indexing and retrieval. These two hashing techniques can better capture the overlapping characteristics in the Q-gram based index. As demonstrated by the experimental results, they run faster than the existing data structures. Besides, we measure the similarity of sequences based on the significance of Q-gram instead of the expensive sequence alignment. Thus, our search algorithm can run faster than the famous Blast algorithm.

Following the idea of QgramSearch, a Q-gram based genomic sequence clustering (QgramClust) is proposed. In view of the challenge of expensive

pairwise sequence comparison for large database sequences faced by the existing clustering algorithms, QgramClust employs the inverted index of Q-gram in sequence comparison so that the clustering process can be made efficient. Our clustering algorithm is a hybrid of partitioning method and hierarchical method. It quickly clusters a group of nearest neighbors and finally merges the clusters. Our experimental results show that QgramClust runs faster than BlastClust.

ACKNOWLEDGEMENTS

It is a great pleasure for me to acknowledge those people who give tremendous efforts and contributions to me. Without their cooperation and advices, it would be much more difficult to complete this thesis.

Firstly, I would like to thank my supervisor, Dr. Korris Chung, for his patient guidance and supervision on my research during these years. Dr. Chung has given many good advices to me and assessed my research works. His innovative ideas trigger me to have new research insights. Every time when I get troubles in the research, Dr. Chung patiently teaches me. I have learned a lot under his supervision. Dr. Chung has contributed much to my research. This thesis cannot be finished without his supervision. I deeply express my grateful appreciation for his great efforts.

In addition, I would like to thank my co-supervisor, Dr. Robert Luk, who gave advices and suggestions, contributed much in my research. His creative idea of using Trie-Based hashing in genomic search is so important in my algorithms. Dr. Luk also helps me to evaluate the performance of different data structures. I really thank you for his great efforts on my research.

Lastly, I wish to express my gratitude to the staff in the Department of Computing. I thank those professors for teaching me and those staff for helping me during these years.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

1 Introduction

Nowadays, bioinformatics has become a hot research topic. It is a multi-discipline subject and involves the use of technologies from biology, chemistry, applied mathematics, statistics and computer science. The development of bioinformatics is growing tremendously and the developed methods have been widely used in many areas. Recently, researchers have completed the whole human genome -- a milestone of the nature of science. The discovery of genome can help molecular biologists to understand the genetic information of organisms and the evolution history. In addition, bioinformatics has also facilitated medical treatment such as cancer detection [54], rational drug design [60] and the identification of mutations that lead to genetic diseases [28].

In the context of computer science, bioinformatics applications often require computational techniques to process data and extract information. For example, homology search in sequence database requires the information retrieval techniques. Motif discovery involves pattern recognition techniques to identify motif [33]. Cancer detection applies classification methods to gene expression data [54],[36]. Despite of the fact that all these computational techniques have been well-developed or widely used in solving other applications, different problems have occurred when they are applied in the domain of bioinformatics. One typical example is genomic sequence search and clustering which are crucial in many

biological applications. In this thesis, we focus on this problem and try to solve it based on some famous computational techniques.

1.1 Problems and Motivations

In molecular biology, genomic sequences are important genetic information for which search and clustering are very useful in many practical applications. Genomic homology search can help biologists to obtain more information of a poorly characterized unit in sequences while genomic sequence clustering can provide the identification of possible protein functional groups [6]. In the past decade, researchers have proposed many different techniques and improved their efficiency and their effectiveness.

Genomic homology search is to find those biologically homologous sequences to a query sequence from a large set of data. Unlike the web documents composed of a set of words, each genomic sequence is a series of characters. Genomic sequences generally are much longer than text words found in many scientific and engineering applications. The high dimensionality of sequence makes it difficult to measure the similarity. In the earlier developments, the edit distance model [63] was proposed to measure the distance between pairs of sequences based on the number of edit operations. On the other hand, dynamic programming was used to recursively compute the edit distance of pairwise sequence and to generate the best alignment. However, finding the best alignment among a large set of genomic sequences is very expensive, thus dynamic programming is not

practical enough in genomic database. Heuristic search approaches can run faster than dynamic programming. For example, BLAST [31] and FASTA [62],[13] are two most famous heuristic search tools. Generally, these approaches firstly find the highly similar regions by scanning the database and then align the highest scoring regions together [3],[67]. The scores of the matched sequences were finally measured based on the pre-defined score matrix and gap penalty. One disadvantage of the heuristic search approaches is that a full database scanning is typically required. As new technologies are emerging in the past decade, the size of genomic database has been exponentially increasing. For example, GenBank, a repository of nucleic acid sequences, is doubling every 15 months and reaches 100 gigabases in 2005 [1]. The running time of the heuristic search approaches would then increase with database size.

Index-based search approaches are attractive alternatives. They avoid database scanning by using inverted index, which has been proved successful in web search engines. FLASH [2], RAMDB [9], MAP [55], PatternHunter[8] and CAFE [19],[21] are index-based search tools which use pre-built indices on subsequences to speed up the lookup process. CAFE claimed that it could run eight times faster than BLAST and 50 times faster than FASTA. However, these index-based search tools also face the common problem of expensive indexing and retrieval processes as well as the problem of neglecting the significance of subsequences. Firstly, as some genomic database may contain millions of sequences of billion bases, the

index creation time would be very long and the periodic updates of the new changes are frequently needed. Similarly, the query sequences are probably very long and some famous servers receive thousands of queries per day. Consequently, the servers suffer heavy loading for index retrieval [25]. Secondly, the index-based tools measure the similarities of sequences based on the occurrences of the common subsequences. But few of them consider the significance of subsequences [66]. In real world, a subsequence may not have as equal probability as another in genomic databases. Some subsequences might not be as significant as they were because they occur in "too many" sequences. This similarity measurement cannot reflect the real statistical significance of subsequences and search statistically significant homology sequences.

Genomic sequence clustering is another hot research topic. Similarly, the simple genomic clustering approaches use edit distance to group the clusters. However, the performance is not acceptable with respect to the huge amount of data in genomic databases. Some heuristic approaches have also been proposed. For example, ProtoMap [18] models the problem using weighted directed graph, CLICK [41] is a graph-theoretic method, ClusSTr [32] adopted the single linkage hierarchical clustering method and CLUSEQ [30] makes good use of a probabilistic suffix tree. Graph-based approaches require a scoring matrix and the construction of this matrix is pretty computationally expensive. Single linkage hierarchical clustering method is also ineffective due to the all-against-all initialization. All these methods

have a common disadvantage that the pairwise similarity measurement, involving all-against-all measurements, requires full database scanning.

1.2 Objectives and Scope

The objectives of this research are to devise efficient and effective search and clustering algorithms for genomic sequence databases. As these databases are exponentially growing in size, we aim at developing new search and clustering algorithms that are scalable to the increasing database size and query rate. They are expected to be faster than the existing tools for large database while the quality of the search and/or clustering results should be maintained.

In view of the fact that the main obstacles of the existing searching tools and clustering tools are the expensive computation time on full database scanning and pairwise sequence matching. In order to overcome these obstacles, we propose to develop the Q-gram based genomic search and clustering approaches. The objectives and the scope of this work can be described as follows:

- develop a technique to efficiently perform index creation and retrieval for a large genomic sequence collection
- define an effective similarity measure for genomic sequences and develop a fast sequence comparison algorithms
- derive an efficient genomic sequence clustering algorithm for again large genomic databases

1.3 Outline of the Thesis

The thesis consists of five chapters. In chapter 2, a brief overview of information retrieval and data mining techniques used in bioinformatics applications is given. In chapter 3, a Q-gram based genomic sequence search algorithm is proposed and its performance is evaluated. In chapter 4, a Q-gram based genomic sequence clustering algorithm is introduced and its performance compared with another algorithm is reported. The final chapter concludes the thesis and outlines out future works.

2 Overview of Bioinformatics

2.1 Introduction

In this chapter, we give a brief overview of bioinformatics and its major applications, focusing on particularly genomic homology search and clustering. Bioinformatics is commonly defined as applying informatics techniques, including applied mathematics, computer science and statistics, to better organize and understand the biological information on a large scale.

In the early beginning, molecular biologists were capable to process the biological data without the help of computational techniques. In the late 1960s, molecular biologists started to apply computing techniques to process the experimental data. With the invention of new technologies like DNA sequencing and microarray technologies, huge amount of data arises and creates opportunities and challenges. In order to process the tremendous amount of data, computers become crucial for the data storage, information retrieval, statistical calculation and genomic analysis [39].

According to Luscombe et al. [38], the aims of bioinformatics can be separated into three levels. The first one is to solve the fundamental problem — the flood of data. Without a well-organized storage system, the huge amount of biological data is useless. Thus, the primary purpose of using

computational techniques is to allow researchers access the existing biological information effectively and also input new discovered data in the system. Currently, there are many well-known public genomic databases. Other than the breakthroughs in the equipments, new computational techniques have also been developed to contribute to this aim. For example, Hunt et al. proposed a suffix tree data structure in excess of RAM size for indexing DNA and protein strings [17]. Williams proposed a compression technique for nucleotide databases and the data can be accessed from secondary storage [20]. Researchers are finding their ways to improve indexing and retrieval processes and integrate the heterogeneous data sources.

The second aim of bioinformatics is to develop more intelligent tools to retrieve relevant biological information. The intelligent sequence search tools FASTA and PSI-BLAST were designed to match homology sequences. They are not just a text-based search engine, but also consider the matches with biological significance, in which scoring matrices such as PAM [49] and BLOSUM [53] are used for computing the level of relatedness of a set of sequences.

With the fundamental data organization and intelligent tools, the third aim of bioinformatics is to analyze the data and interpret the results in a biologically meaningful manner. Biological studies are not limited to analyze the individual experimental result. They are also expected to

analyze all the available data and find the common principles that can apply across many systems and discover new features.

2.2 Biological data

Biological data is the key of knowledge discovery in biological databases. Different sources of biological data contain different information, including DNA sequences, protein sequences, macromolecular structures and the results of functional genomics experiments [28]. DNA (Deoxyribonucleic Acid) stores the instruction required by a cell to perform its function, which is a normally double stranded macromolecule. The two DNA strands form a helical spiral, each of which is formed from nucleotides. A nucleotide is composed of three parts: backbone of the DNA strand, deoxyribose sugar and nucleotide base. Nucleotide can be categorized into 4 bases: A for adenine, G for guanine, C for cytosine and T for thymine. Other than nucleotide base, there are some wildcard characters used for substitutions in a sequence. The deoxyribose sugar of the DNA backbone has 5 carbons and 3 oxygens. The carbon atoms are numbered C1, C2, C3, C4', and C5. The hydroxyl groups on the C5 and C3 link to the phosphate groups to form the DNA backbone. One end of the DNA backbone is called 5' and the other is 3'.

DNA sequencing is the process of determining the nucleotide order or a given DNA fragment. Raw DNA sequences are typically 1000 bases long while genomes are ranging from 1.6 million bases in Haemophilus influenzae to 3 billion in humans. The collection size of DNA sequence in GenBank has exceeded 100 gigabases in August 2005.

Proteins are the primary components of cellular structures and control the cell division process. Proteins are amino acid chains that fold into unique 3-dimensional structures. The amino acids are joined by a backbone: one end is N-terminus and the other is C-terminus. There are 20 different types of amino acids. The 3-dimensional structure is determined by the linear sequence of amino acids. The structures are called primary, secondary, tertiary and quaternary structures. Primary structure is the amino acid sequence; secondary structure is the highly patterned sub-structures (alpha helix and beta sheet); tertiary structure is the overall shape of a single protein molecule; and quaternary structure is the association of polypeptides. Proteins can be formed from tens to thousands of amino acids and the average length is 350. Amino acids are encoded by nucleotide triplets, called codons. Since there are 64 possible codons and only 20 amino acids, different codons can correspond to the same amino acids.

Macromolecular structural data represents a complex form of information. The 3D-structural information usually comes from three techniques: X-ray crystallography, NMR spectroscopy and cryo-electron microscopy. The Protein Data Bank, PDB, is one famous database for macromolecular structure data. A PDB file for a medium-size protein typically contains 2000 atoms.

Functional genomic experiment data is obtained from laboratory experiments. For example, microarray technologies enable scientists to simultaneously measure the transcription level of every gene within a cell.

Gene expression is based on the measured fluorescence intensity of red signal, the fluorescence intensity of green signal and the ratio of red to green signal. The gene expression level is assumed to be directly proportional to the abundance of mRNA for each gene [26]. The advantage of using microarray technologies is that researchers can measure many thousands of genes with only a few biological samples in a single experiment. Since the data is very high dimensional with little replication, it causes the problems of statistical analysis and normalization of multiple set of data.

There have been many different research areas in each biological data source. For DNA sequences, the research areas include homology search, separating coding and non-coding region, and identification of introns and exons. For protein sequences, it includes sequence comparisons, multiple sequence alignments, discovering functional protein families. For macromolecular structure, it includes secondary structure prediction. For functional experimental data, it is applied in cancer detection and drug prediction. In our research, we focus on genomic sequences.

2.3 Homology Genomic Search

In molecular biology, genomic sequences are very critical in gene prediction, sequence comparison, and identification of sequence repeats and functional regions. Homology among sequences represents their evolutionary distance of two species. Finding homologue between genomic sequences is very useful in many practical applications. For example, by finding the homologous sequences in the database that we have better understanding, biologists can obtain more information on a poorly characterized protein.

Genomic sequences are made of strings of characters. The trivial way of measuring the sequence similarity is string comparison. It has been applied in many areas such as web document search, speech recognition, library management and biological data. There are several famous models for string comparison such as edit distance model, maximal match model and Q-gram model.

2.3.1 Measuring Evolutionary Distance

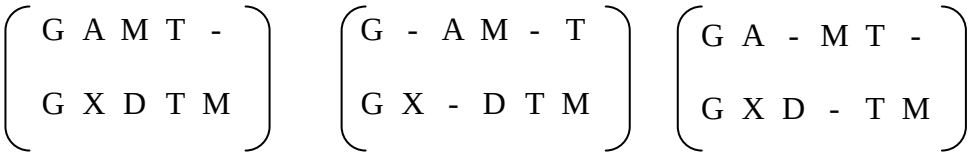
In the evolution process, the mutations of genomic sequences occur in the species. Measuring evolutionary distance is usually a measurement of the number of point mutations. It is regarded as the reconstruction of an evolution process that transforms one sequence into another one. Sequences are homologous if their measured distance is small. For example, Euclidian distance, hamming distance and block distance are used for measuring distance the strings of equal length. Edit distance model is one of the most

famous models for comparing strings in different length. It has been widely used in the construction of optimal alignment of genomic sequences [37].

In edit distance model, the distance of two strings is based on the number of edit operations. The main idea is that two strings are similar if the number of required edit operations converting one string to another string is only a few. An edit operation represents the transformation of a character in a source string into another one in target string. There are three types of edit operations: deletion of a character, insertion of a character and the replacement of a character to another character.

2.3.2 Sequence Alignment

A sequence alignment can be regarded as a sequence of edit operations converting a source sequence into a target sequence. Sequence alignment is used in natural language, financial data and genomic sequences. There have been much research in both pairwise alignment and multiple alignments. Sequence alignment approaches can be divided into two groups: global alignment and local alignment [5]. Global alignment conducts the alignment over the entire length of strings and it is suitable when two sequences are of similar length. Another one is local alignment. It identifies the regions that are highly similar and then involves the stretches of subsequences. It is suitable when a large set of sequences are of significantly different length. Figure 2-1shows three different alignments for two sequences u = GAMT and v = GXDTM.



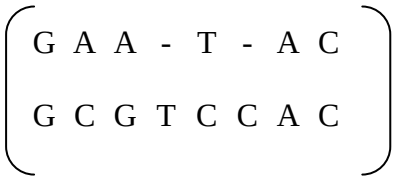
Alignment 1 Alignment 2 Alignment 3

Figure 2-1 Three alignments of sequences u and v

In 1970s, Needleman and Wunsch proposed a global alignment algorithm for two sequences [44]. The advantage of global alignment is that it can guarantee to find alignment with the maximum score. An alignment is scored based on the total of similarity scores of aligned characters. In Figure 2-2, for inserting an empty character, gap penalty g is given. The similarity score of alignment X = 3 + (-1) + (-2) + g + (-2) + g +8 + 5.

-	A	C	G	T
A	8	-1	-2	-3
C	-1	5	-5	-2
G	-2	-5	3	-1
T	-3	-2	-1	6

Similarity matrix



Alignment

Figure 2-2 A similarity scoring matrix and an alignment

For any two sequences, there could be many different alignments. Global alignment approaches apply the dynamic programming algorithm to the

optimal alignment. Dynamic programming for pairwise sequence alignment requires three steps: initialization, matrix fill and traceback.

It firstly initializes a two-dimensional score matrix C by placing the characters of sequence X into the first row and the characters of sequence Y into the first column. The value at $C_{i,j}$ stores the maximum score for aligning the prefix $a_1a_2…a_i$ of sequence X to the prefix $b_1b_2…$bj of sequence Y. Secondly, it uses a divide-and-conquer strategy to find the value at $C_{i,j}$. The matrix C is recursively calculated by the following scheme shown in Figure 2-3 and Figure 2-4. After the score matrix is filled, the traceback is performed to deduce the best alignment from the traceback matrix.

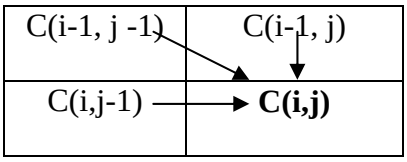


Figure 2-3 Relationship between a scoring matrix cell C(i,j) and its three adjacent cells.

$$C(i,j) = \max \begin{cases} C(i-1, j-1) + s(xi, yi) \text{ where } s(xi, yj) \text{ is the substitution score} \\ C(i-1, j) + g \\ C(i, j-1) + g \end{cases}$$

Figure 2-4 Divide-and-conquer method used in the computation of scoring matrix

Generally, local alignment approach is used rather than global alignment because these sequences may contain short coding regions separating by long non-coding regions. Nucleotide sequences are long and do not have notion of "end" and so generally do not have overall similarity. Based on the Needleman-Wunsch algorithm, Smith and Waterman proposed a local alignment algorithm [54], which runs faster than Needleman-Wunsch one. However, as the complexity is $O(n^2)$ in both time and space, the performance is still not acceptable for a large set of sequences.

Like pairwise sequence alignment, multiple sequence alignments are used to match the highly similar regions among a set of sequences. They have been used for characterizing protein families and inferring the biological characteristics of new sequences given known families of sequences. Feng and Doolittle proposed a progressive sequence alignment that has been adopted in most multiple alignments methods [12]. Clustal W is another famous multiple alignment methods. It uses individual weight to each sequence in a partial alignment and changes gap penalty scheme [29]. In general, the multiple alignments can be divided into three steps:

i) Firstly, all the pairs of sequences are aligned separately and the pairwise scores are stored in a distance matrix. The dynamic programming is adopted in the pairwise alignment.

ii) Secondly, a tree is constructed from distance matrix to guide the multiple alignment process using the Neighbor-Joining method. The tree is used to derive a weight for each sequence.

iii) Finally, the sequences are progressively aligned into a larger group according to the branch order of the guide tree. The process starts from the tips towards the root.

2.3.3 Heuristic Exhaustive Search

Both local and global alignment approaches are impractical for a large database collection. Local alignment approach takes a day of processing a query to match a large database collection. Therefore, heuristic exhaustive search tools were developed and they utilize heuristics in local alignment.

2.3.3.1 FASTA

Wilbur-Lipman algorithm is one of the pioneers of heuristic exhaustive search and it uses a global comparison of sequences based on fixed length subsequences [62]. Compared with the local alignment approach, it can reduce much time on the alignment process. Firstly, the algorithm built a hashing structure with all intervals of fixed length in the sequence as keys. An interval is overlapping such that there are l-n+1 intervals for a sequence of length l and interval length n. For example, for n=3, a query sequence ACGTGTA is processing with the intervals ACG, CGT, GTG, TGT and GTA. Then, all the intervals in each database sequence are processed with hashing. If the interval can be lookup in the pre-built hashing structure, there is a match with query sequence. The offsets are used in the matching for guarantying the alignments without gaps. Scores are accumulated for

each alignment without gaps. This method is faster than going through the query sequence for every interval in the database. There are many variations of Wilbur-Lipman algorithms. FASTA and BLAST are the popular ones and have been used in many public genomic databases.

FASTA consists of four steps. The first step of FASTA and BLAST is also using the variants of Wilbur-Lipman algorithms to preprocess the query such that the query sequence is divided into intervals of fixed length and the intervals in database sequence will be matched up using the hash table. In the second step, an accumulator is used to score the matched words between two sequences and localize the top ten regions. The third step is to join the highly-scoring regions using acyclic graph and only those sequences with a score higher than a threshold will go to the fourth step. The fourth step is the remaining high-scoring sequences are further processed by dynamic programming and the scoring of each alignment would be computed.

2.3.3.2 BLAST

BLAST has several variant versions [61]. BLAST 1 improves the performance of FASTA [15]. It locates ungapped similarity regions between sequences instead of comparing each word of the query with each word. The first step is to create a list of words of fixed length. The second step is to identify all exact matches with database sequences. In the third step, it extends the matched word in both directions until the obtaining score decreases to a certain level. The extended word has score higher than a

threshold, called High Scoring Segment Pair (HSP). The third and the fourth step are similar to FASTA. BLAST 1 uses a technique to speed up the alignment by disallowing the insertion and deletion operations on residues, but only allowing the substitution of one residue for another. The underlying assumption is that indels are a less significant factor of evolutionary event. This assumption has advantage in nucleotide comparison because single deletion and insertion events cause the meaning of codons to be completely lost. However, it has been shown that FASTA has higher sensitivity than BLAST at detecting distant homologous relationships.

BLAST 2 improves both speed and accuracy of BLAST 1. BLAST 2 permits the limit use of indels in forming alignment and adopts a two-hit HSP technique. The use of indels requires more computation to evaluate each local alignment. In order to solve this problem, BLAST 2 introduces a two-hit HSP technique for the selection of the diagonals. Only those diagonals containing at least two intervals will be considered. This selection process can reduce the number of sequence required for local alignment. Altschul claimed that BLAST 2 has better accuracy and performance than BLAST 1 [43].

2.4 Sequence Clustering

Genomic sequence clustering under a large database is another interesting and crucial research topic. Clustering genomic sequences is to get a biologically meaningful partitioning. According to SEQOPTICS, clustering on protein sequences has several advantages [65]. Proteins are grouped into families, which provide useful information of their general features and their evolutionary process; clustering also helps to predict the biological function of a new sequence by its similarity to some known function of a new sequence; besides, clustering can be used to facilitate protein 3-dimenional structure discovery. There have been many clustering algorithms developed for genomic sequences. They can be grouped into several categories: graph-based, hierarchal and partitioning model.

2.4.1 Graph-Based Approaches

Graph-based techniques have been proved successful in solving many complex computational problems. They have also been used in clustering problem. Jain and Dubes gave a brief introduction to graph-based clustering [4]:

- Compute a complete undirected graph G where vertices are identified with protein sequences and each edge represents a Smith-Waterman local alignment, weighted by a similarity score. The similarity score can be measured with Smith-Waterman score.

- Replace each undirected edge with two directed edges and the weighted is changed.
- Remove all edges with score less than threshold from graph G
- Compute all strongly connected components and list all the outputs. The strongly components are defined as maximal sets of vertices such that directed path exists from P to Q and from Q to P for all vertices.

The traditional graph-based clustering faces a problem of inappropriate transitive relations, especially in the multi-domain proteins [14]. This is called false transitivity. ProClust improved the graph-based clustering algorithm by introducing an asymmetric distance measurement [40]. The algorithm used the significance of alignment for the filtering step and Profile-HMM for the merging step. BAG utilized several graph properties of biconnectedness and articulation points to improve the problem [52].

2.4.2 Hierarchal Approaches

In hierarchical clustering, a distance matrix of all pair nodes is computed and then a cluster hierarchy is formed. Clusters can be obtained from the subtrees of the hierarchy. There are two ways of forming the hierarchy: agglomerative and divisive. Agglomerative hierarchical clustering starts with clusters containing single objects and then merges them until all objects in the same cluster. Divisive hierarchical clustering begins with only one big cluster and then iteratively divides it into small objects [57].

The similarities of sequences are commonly measured by local alignment or global alignment. Dynamic programming is time-consuming and it is not practical for large databases. The time complexity of the algorithm using dynamic programming is $O(n^2m^2 + n^2 \log n)$, where n is the number of data sequences and m is the average number of clusters.

Another alternative is using pattern-oriented agglomerative hierarchical clustering (POPC) [42]. The algorithm uses some well-known or frequent patterns discovered in the early pattern discovery process to measure the correlation of sequences. The similarities of sequences are measured by Jaccard coefficient. Although the scoring process of POPC is faster than dynamic programming, the time complexity of all-again-all strategy is still quadratic to the number of sequences. An additional pattern discovery process is also required.

2.4.3 Partitioning Approaches

In partitioning approaches, each data node is assigned to a cluster according to the distance of that node to the corresponding cluster. K-means and K-medoid are two common approaches in partitioning approaches. The main difference is that K-medoid uses a data node as a centroid of a cluster while K-means uses a feature vector of mean values among data nodes in that cluster as a cluster centroid. Guralnik proposed a feature-based K-means clustering, which projected each data-sequence into a new space whose dimensions are these features [59]. The features are selected when all

sequential patterns whose length is within specific range and satisfy a minimum support. The cosine similarity function is used to measure the similarity of sequences.

Previously, we describe three different clustering categories from the view of cluster formation. However, from another view of scoring method, the clustering methods could be generally divided in three groups: proximity-based methods, features-based methods and model-based methods [24].

Proximity-based methods include edit distance model and dynamic programming. Feature-based methods perform pattern projection into a new space. Model-based methods assume an analytical model for each cluster. CLUSEQ has adopted a probabilistic suffix tree to cluster a set of sequences if their similarity is less than a threshold. The algorithm uses the conditional probability distribution and assumes that sequences belonging to a cluster may subsume to the same probability distribution of symbols. The similarity of sequence s and cluster c is obtained by the probability of sequences divided by the probability of a random sequence. The algorithm determines whether a sequence should belong to a cluster by calculating the likelihood of (re)producing the sequence under the probability distribution. Besides, Hidden Markov Models (HMMs) is another model-based method for clustering sequential data [6]. First, HMMs assume that the observation at time t was generated whose state is hidden from the observer. Second, they

assume that the state satisfies the Markov property. Generally, HMMs are trained for clustering the sequences having similar behavior.

2.5 Summary

In this chapter, we have given a brief overview of bioinformatics and its practical applications. Bioinformatics is a multi-discipline subject and involves intelligent data analysis on the biological data. The biological data has different characteristics from other data like the web documents and has posed some challenges, e.g. high dimensionality, to the traditional information retrieval and data mining techniques. Genomic sequence search and clustering are also facing the problem of handling tremendous amount of data. We have reviewed the famous genomic sequence search and clustering tools. In the coming chapters, we will propose some novel algorithms to improve the performance of genomic sequence search and clustering.

3 Q-gram Based Genomic Sequence Search

3.1 Introduction

As the microbiology technology advances, more genomic sequences can be discovered in the laboratory experiments. There are many public genomic sequences databases that provide the search services to the biologists. Thousands of queries are processed by these search engines everyday and their loading will continue increasing as more new applications are developed. Most of these search tools use the exhaustive search to compare the sequences. The high growth rate of data will have a great impact on the performance of these search tools.

In this chapter, we introduce a novel QgramSearch (Q-gram based genomic sequence search) to enhance the efficiency and the effectiveness. QgramSearch has the advantage of using inverted index but greatly reduce the computational time of index creation and retrieval by using our proposed dictionary structures. Another enhancement is that we introduce a probabilistic scoring method instead of counting the occurrences of shared words. The statistical significance of words is considered in our proposed approach.

3.2 Q-gram Based Approach

Q-gram is a string of characters of fixed length q. A sequence can be broken into a set of Q-gram. There are several ways of formation of Q-gram: non-overlapping, consecutive overlapping and partially-overlapping. Suppose a sequence ACTAGACG and q equal to 4. In non-overlapping method, the sequence forms a set of Q-gram {ACTA, GACG}. In the consecutive overlapping method, the sequence forms a set of Q-gram {ACTA, CTAG, TAGA, GACG}. In the partially-overlapping method, consecutive Q-gram are overlapping on only a portion of characters. For example, if the overlapping characters are 2, the sequence forms a set of Q-gram {ACTA, TAGA, GACG}.

The time complexity of processing a sequence is proportional to the number of Q-gram in the sequence. Supposed a genomic sequence of length N is processed and Q-gram size is Q (Q<N). For non-consecutive overlapping scheme, the time complexity is O($\frac{N}{Q}$). For consecutive overlapping scheme, the time complexity is O(N-Q+1). For partially-overlapping scheme, the time complexity is O($\frac{N-Q}{Q-O}+1$), where O is the number of overlapping characters between two consecutive Q-gram and O is between 0 and Q.

Q-gram can be regarded as the features of sequences. The more features of sequences they share, the closer relationship they have. Q-gram

can be used for similarity measurement. Among all overlapping schemes, the consecutive overlapping one can have better recall rate than the other two methods because the larger overlapping regions would degrade the sensitivity of matching process. Our proposed tools would adopt consecutive overlapping scheme. Figure 3-1 shows the way of sequence comparison for two sequences A and B using Q-gram with size 4. Sequence A and sequence B share two same Q-gram.

There are some advantages of using Q-gram. Firstly, sequence comparison using Q-gram can run faster than the highly computational sequence alignment and edit distance model. Measuring the similarity of two sequences can be simply counting the number of Q-gram they share. The running time of Q-gram approach is linearly proportional to the length of sequences. It is scalable for the large amount of sequences. Secondly, Q-gram facilitates the use of indexing techniques. Indexing techniques have been successfully used in web content search. Since genomic sequences are strings of long sequences, it is not possible to directly apply indexing on the whole sequences. We propose to break sequences into a group of Q-gram that can be stored in the indexing file. By using inverted index on Q-gram, we can quickly search and cluster the genomic sequence database.

	1	2	3	4	5	6	7	8	9
Sequence A	A	C	C	T	A	C	T	G	A
Sequence B	A	C	C	T	T	C	T	G	A

Perform sequences comparison using Q-gram with 4

Sequence's A Q-gram		Sequence's B Q-gram
A C C T		A C C T
C C T A		C C T T
C T A C		C T T C
T A C T		T T C T
A C T G		T C T G
C T G A		C T G A

Figure 3-1 Sequence comparison for two sequence using Q-gram

3.3 Q-gram Scoring

Finding homologous sequences is generally done by calculating the pairwise similarity score of each database sequence and query sequence. Dynamic programming calculates the score by finding the best alignment. However, it is not practical for a large database, thus most of the existing search tools adopt heuristic approaches. One of the common heuristic approaches is to define the similarity of two sequences based on the occurrence of shared Q-gram. The more subsequences database sequence and query sequence they share, the higher similarity score is given. This theory has an assumption that all Q-gram occur in a random manner or have same statistical significance. It is not often in the real life. For example, the words like "THE", "AND"

have higher occurrence than the words “USA”, “DNA”, even they are in the same length.

We suggest that the statistical significance of Q-gram should be considered in our scoring scheme. If all Q-gram has equal weight, some frequent subsequences may cause bias. Therefore, in our methodology, weighted Q-gram is introduced to compute the similarity. The weight of Q-gram is inversely proportional to the rarity of subsequence.

Suppose the query sequence of length L_x and the database sequence L_y. For an alphabet of n symbols (generally, n=4 for DNA sequences and 20 for proteins), the possible number of Q-gram is n^q, where q is word length. A binary vector w^s represents the absence and the presence of each of the possible Q-gram in sequence s. $w_i^s = 1$ means the Q-gram i exists in sequence s while $w_i^s = 0$ means the Q-gram i does not exist in sequence s. A vector F represents the normalized frequency of each word. The sum of all F_i is equal to 1. The weighted score of two sequences is measured as following:

$$\text{Weighted Score} = \sum_{i}^{n^k} \frac{1}{F_i} \times W_i^x \times W_i^y .$$

However, the disadvantage of weighted score is that it tends to favor long sequences due to the higher number of Q-gram occurrence making larger similarity score. We normalize the weighted score by the lengths of sequences.

$$\text{Normalized Weighted Score} = \frac{1}{L_x \times L_y} \sum_{i}^{n^k} \frac{1}{F_i} \times W_i^x \times W_i^y .$$

3.4 Dictionary Structure Lookup

Heuristic exhaustive search approaches face the challenge of exponential growth in database collection size. The interval match of a large amount of database sequence to query sequence is quite expensive. Some tools are implemented by storing the collection in main memory. Such approaches will not be sustainable in the future. A pre-built dictionary structures is a common way to speed up the search process. It has been proved the dictionary structures can enhance the efficiency in many different applications. Database records are firstly pre-processed and stored the information in specially designed structures. When a query is performed, only relatively small amount of information is fetched via the dictionary structures. We will briefly describe some popular data structures, including binary search trees (BST), splay trees, hash tables and tries.

Binary search trees are composed by many nodes; each of them represents a character or a string and contains two pointers to its child nodes. For a balanced tree, the time for accessing a vocabulary is O(log n), where n is the number of nodes in the tree. However, longer access time is required for unbalanced tree which is typically bounded by O(n) [11].

Splay trees are a variant of self-adjusting binary search tree [22]. The special feature of splay trees is the self-adjusting mechanism to change the tree based on the access patterns. Some search operations can be speeded up but some requires more time to run. A splay tree requires more

space than the binary search tree because of the additional pointers to its parent.

Hash table is an array of records. Each record contains a key and its associated hash value. The hash value is calculated mathematically. During the past decades, researchers have developed many algorithms on hashing functions, which can be categorized into perfect or non-perfect ones. Perfect hashing algorithms can guarantee constant-time lookup even in the worst case. Non-perfect hashing causes more collisions than the perfect one. The most common hashing functions are briefly explained as follows.

Linear probing, a naive hashing algorithm, is used in many applications. Assume a hash function $h(x)$ associates with a key x. When inserting a key x, we check the bucket $h(x)$. If there is no item, we insert x. Otherwise, we check the next bucket h(x)+1. If that bucket is also occupied, we check h(x)+2 and so on [10]. The linear probing has a problem that collision may always occur in dense region.

Bitwise hashing function was proposed by Zobel. It takes use of a non-prime number as its seed. Typically, most hashing functions require many complex mathematical operations such as power and module. The advantage is that the bucket size is the power of 2 so that the shift bit operations can be used. The time of calculating a hash value becomes lesser than other complex hashing functions. But bitwise hashing function is still slow in indexing a large amount of sequences.

Perfect hashing function assigns each unique word to a unique hashing integer. For indexing genomic database sequence, the bucket size is the number of all possible Q-gram (independent of database size). The hashing integers can be assigned to the words based on the orders of its predefined alphabetical ID. The calculation of a hashing integer can be done by a formula. The parameters of the formula are the alphabet ID and the position of each character. We assume that the leftmost character in a string is the most significant while the rightmost character is the least significant. We can calculate the hash value by an ordered function. For a string "$a_{n-1}\, a_{n-2}$ ... $a_1\, a_0$",

Hash value of the string = $\sum p_k \times L^k + p_0$

where p_k is the alphabet ID of the k-th character in the string and L is the number of alphabet characters.

Trie is an abstract structure and Brandais was one of the early researchers of such data structure. This structure generates a list of nodes based on a set of words. There are three major representations of tries: array-tries, list tries and ternary search tries. The advantage of using Trie is the running time of processing a text is independent of the number of keywords [27]. It is efficient to search in a trie because it only requires one pointer traversal for each letter in the query string. Trie has been used in various applications like library bibliographic search, natural language dictionaries, database systems, compliers and trie image search. There are

many variants of tries proposed for searching dictionary words. Marshall and Alan focused on tackling the problem of pattern collision by using trie [35]. Pattern collision occurs when a perfect hashing function generates the same addresses of two different words. This algorithm has an advantage over other perfect hashing functions because it can produce an ordered minimal item lists. It adopts a two-dimensional array based trie: one dimension is the alphabets from ‘a’ to ‘z’ and the other dimension is the position in a word. Let us consider an example of searching the word “apple” in the array. First, look at the cell in the row ‘a’ and column 1. If the cell value is positive, we get the word and stop searching. If the cell value is negative, that means other words also have first character ‘a’ and we have to consider the second letter. Then, look at the cell in the row ‘p’ and column 2. If it is negative, repeat the above steps. The most recent one is burst tries proposed in 2002 [50]. It is composed of three main components: a set of records, a set of containers and an access trie. Using a set of containers, the burst trie requires much less string comparisons.

3.4.1 Comparison of Dictionary Lookup Structures

The Marshall and Alan algorithm is efficient but only practical for storing a small amount of words [35]. The maximum number of words in the index table is $L \times n$, where L is the number of alphabets and n is the maximum word length. It is not suitable for the large amount of genomic database sequences. As reported in [51], the bit-wise hashing can outperform the

binary search tree, splay tree and compact trie in accumulating the genomic Q-grams with size 9 and also requires the least memory. Binary search tree is the slowest one because it requires many string comparisons for large data. Compact trie is more efficient than splay tree but bit-wise hashing can run about 30% faster than compact trie. Burst trie is faster and requires less space than compact trie. However, the performance of burst trie is still not as good as that of bit-wise hashing. It was found that the burst trie has worst performance in genomic data because of its equal string length and its relatively flat probability distribution. Next, we will give a brief review on the genomic search tools using these dictionary structures.

The Search Search Tree (SST) was proposed to search a database of DNA sequences for near-exact matches, in time proportional to the logarithm of database size. It creates a tree-structured index of a fixed length tuples in vector space, with tree-structured vector quantization (TSVQ). It can achieve O(log n) for the search if the tree is balanced. It is claimed that SST is 27 times faster than BLAST for searching alone [48],[16].

The Suffix Trees based Exact Match (STEM) was proposed for indexing all suffix elements of the database sequence and the depth-first-traversal search is used for matching elements. There are several suffix tree representations including Patricia tree-based, linked list and hash table representations, suffix array and augmented suffix array, LC-trie, suffix

binary search tree and suffix AVL tree. STEM adopted suffix array as its representation [23]. In the next section, we will explain how our proposed data structures works on the genomic data and discuss the advantages.

3.5 Proposed Dictionary Structures for Genomic Sequences

In our Q-gram based sequence search, we propose a novel dictionary data structure to handle the genomic sequences. The existing dictionary data structures often take time proportional to database size. Even the perfect hash function has better performance than BST and splay trees, it still takes more time to compute the perfect hash value if a word is in longer length. A genomic search system often performs index creation periodically and deals with thousands of queries per day, an efficient dictionary data structure is important. Thus, we propose two different dictionary lookup approaches, which can reduce the number of mathematical operations than the existing ones.

3.5.1 Refined Perfect Hashing

The overlapping word indexing technique is adopted in most successful search tools such as BLAST, FASTA and CAFE because this can achieve better retrieval effectiveness. We discovered that this overlapping interval property could make hashing functions more efficient. Since any two consecutive overlapping words contain nearly the same characters, it can take less time to obtain the hash value of the followed word. The idea is to calculate the hash value of a followed word based on the hash value of the previous word. The number of operations required in the refined version can be much reduced. For a string "$a_{n-1}a_{n-2}... a_1a_0$",

$$\text{Hash value of the string} = (H - p_{n-1} \times L^{n-1}) << 2 + p_0$$

where H is the hash value of previous word, p_k is the alphabet ID of the k-th character in the string and L is the number of alphabet characters.

For example, a new sequence ACTAAAAAAGT is indexed. The sequence is broken down into three Q-grams with size 9: ACTAAAAAA, CTAAAAAAG and TAAAAAAGT. The hash value of the first word is 24576, which is calculated by the perfect hashing function. For the remaining words, the refined version is applied. Compared with the first word, the second word misses the first character 'A' and adds the character 'G' to the end. The steps of the refined version to calculate the new hash value are shown in Figure 3-2

1. Use the hash value of previous word 2. Minus the value of the missing character 3. Operate a left shift operation 4. Add the value of the last character	1. Hash value of CTAAAAAAG = (24576 – 0) <<2 +3 = 98307 2. Hash value of TAAAAAAGT = (98307–4^8) <<2 +2 = 131086

Figure 3-2 The calculation steps of Refined-Perfect Hashing

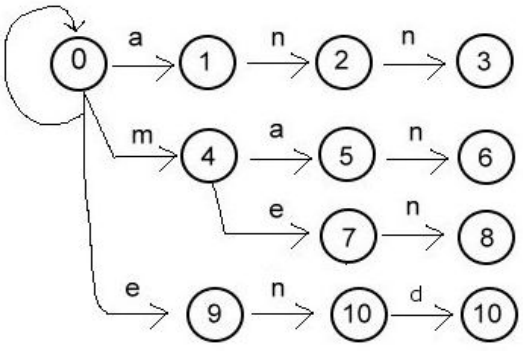
3.5.2 Loop-Back Trie-Based Hashing

The loop-back trie is based on the matching machine in Trie proposed by Alfred [58]. By applying a matching machine in Trie, it can identify a set of keywords in a string efficiently. The algorithm consists two parts: the first part constructs a finite state matching machine based on the input set of

keywords while the second part applies the matching machine to lookup the query keyword and give the matching signals [34],[53].

3.5.2.1 Matching Machine

The matching machine consists of a number of states, which are represented by consecutive integers. When the characters are read from the text one by one, the machine will move from one state to the other. That is called state transitions. The state transitions and the matching signals are monitored by three functions: *Goto* function, *Failure* function and *Output* function. The Goto function is executed when the machine successfully matches the processing character to the states and finally moves to a new state. The failure function is operated when the machine fails to match the processing character. The machine is directed to the state associated with the longest suffix of the string. The output function gives the signal that a keyword is found in the text string. These functions with the keywords {ann, man, men, end} are shown in Figure 3-3.



(a)Goto function

State	Output
3	ann
6	man
8	men
12	end

(b)Output function

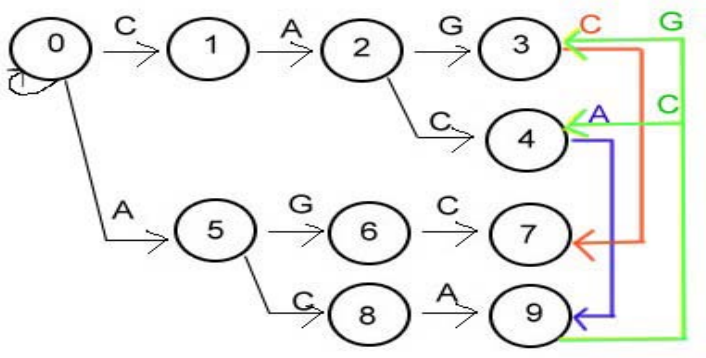
State	Failure
6	2
8	10

(c) Failure function

Figure 3-3 Pattern Matching Machine

3.5.2.2 Loop-Back Path

A machine state is built by pre-processing the database sequences. The pre-calculated states are stored in the three functions. Since the consecutive words are overlapping, we can add a guide path to the state for a next word, which is called loop back path. Loop back path can help to reduce the number of state transitions for going through a series of consecutive words. For example, given a set of indexed keywords {CAG, CAC, AGC, ACA}, the functions will be shown as followings:



(a) Goto function

State	Output
3	CAG
4	CAC
7	AGC
9	ACA

(b) Output function

State	Failure State if Next Character is
3	7 if N.C. is C
4	9 if N.C. is A
9	3 if N.C. is G or 4 if N.C. is C

(c) Failure function

Figure 3-4 Pattern Matching Machine used with Loop Back Path

In Figure 3-4, the modified failure function provides a failure state based on the next character. By using the loop back path, the hash value of the

consecutive words can be obtained quickly. Only one state transition is required in the hashing. However, if without using the loop back path, the machine state must restart to state 0 for the hashing of every word. It spends much time to go through the duplicated characters. The loop back paths of Trie can skip the process of duplicated characters. For example, an input sequence is ACAGC. The machine state starts from 0 and processes ACA. It then goes through states 5, 8 and 9. Since the next character is 'G', it follows the loop back path to state 3 and then directs to the state 7 for the next character 'C'. Trie hashing function can be very fast since the complex calculation process can be avoided. When the machine state processes a genomic sequence, it jumps one state to another according to the loop back paths. Since the failure function table can be stored in the primary memory, the state transition operations can be run very fast.

3.6 Effectiveness of Loop-Back Pointers

In this section, we focus on the effectiveness of loop-back pointers in the Trie. Loop-back pointer is used to quickly find the hash value of consecutive Q-gram. They are stored in a pre-built data structure. The effectiveness of loop-back pointers can be affected by the characteristics of genomic sequence databases. The characteristics include several factors including the number of sequences and the ratio of the unique Q-gram in database to the possible Q-grams. In order to enhance the effectiveness of lookup process, we have devised two different approaches to manage the loop-back pointers. Under the different characteristics of database, different approach can be chosen.

3.6.1 Minimal Trie

Minimal Trie is suitable for the datasets, in which the number of unique words in database is expected to be much smaller than the number of all possible n-grams. The construction of the failure function is done by dynamically allocating the memory for the unique words in database. Since those words that do not appear in the database would not be stored, the table size could be smaller. Table 3-1 shows Minimal Trie for nucleotide characters, in which each row contains 4 possible loop back pointers in the failure function.

The goto and output functions are implemented by an array-tree. This structure is suitable to build the machine state because the time of insertion,

deletion and lookup of items is constant and independent of the tree size. The failure function uses a two-dimensional array to store all loop back pointers. For the index construction, the algorithm is to go through all genomic sequences in the database. The hash value of each consecutive word is obtained from the failure function. If it does not exist, the output function will be checked whether it has been added to the tree. If not, add hash values for this new word in the tree. Finally, insert the corresponding value into the loop back state of the failure function.

Table 3-1 Pattern Matching Machine for Minimal Trie

Hash Value	Output
1	ACA
2	AGC
3	CAC
4	CAG

(a) Output Function

Hash Value	A	C	T	G
1	-	3	-	4
2	-	-	-	-
3	1	-	-	-
4	-	2	-	-

(b) Failure Function

3.6.2 Perfect Trie

Perfect Trie is easier to deploy but requires more space for the array than the first one as shown in Table 3-2. It statically allocates memory for the hash values of all Q-grams and the table size is L^q, where L is the number of alphabets and q is word size. In the failure function, only one loop back pointer for the character 'A' is stored since the pointers of other characters can be calculated. For example, the hash value of GCA is 52 and so the hash value of GCG = 52+3 =55.

Table 3-2 Pattern Matching Machine for Perfect Trie

Hash Value	Output
0	AAA
3	AGC
17	CAC
63	GGG

(a) Output Function

Hash Value	A
0	0
13	52
17	4
63	60

(b) Failure Function

Like the perfect hashing, the hash values are continuous and they can be alphabetically generated. Thus, the output function would use the perfect hashing formula and hence no extra data structure is required for the array-tree. The failure function uses a one-dimensional array to store one loop back value for each word. Since the hash values are contiguous, the failure function values can be simply calculated. As it does not require any processing genomic sequences, the construction time of the failure function table is less than that of the first one. We believe that the construction time of the failure table only causes little overhead to the whole indexing process. The number of insertions in the failure table depends on the number of unique words. Since this number is relatively small compared with the words in a genomic database, the pre-calculation of the failure values can enhance the whole indexing process.

3.7 Simulation Results

In this section, we measured the performance of the QgramSearch. We evaluated the performance of the proposed algorithm in several aspects: space requirement, computational time and scalability on the database size. In these experiments, our data came from the genomic sequences in a public famous database GenBank, in which Expressed Sequence Tag (EST) was chosen. The program run under the Unix environment and the machine was SunFire48000 with 4 US-III 900 MHz plus 4 GB Ram. QDBM, a widely used library for inverted index applications, was used in our program. The followings are the experimental results for measuring several aspects:

3.7.1 Effect of Word Size on Dictionary Structure Efficiency

The first experiment evaluated the efficiency of several dictionary structures on the genomic sequences. Figure 3-5 shows the running time of BitWise, Perfect, Refined Perfect and Trie-based hashing functions to generate the hash values of words in the genomic sequence file. Since we expect the running time of the two trie-based implementations are nearly the same, only the first one is shown in the experiment. Our program first broke down the sequences into words and computed a hash value for each word. The experiment focused on the effect of word size on the speed of hashing

functions, thus word sizes 3, 7 and 10 were used. The sequence file was 28.6 MB and the lengths of sequences ranged from 100 to 1000.

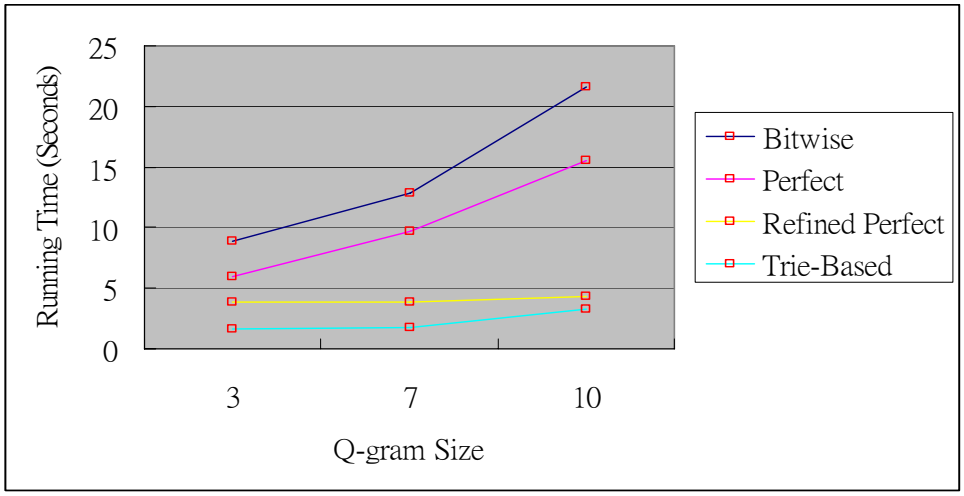


Figure 3-5 Plot the running time of BitWise, Perfect, Refined Perfect and Trie non-minimal hashing indexing.

Our results showed that the trie-based hashing function was the most efficient one in generating hash values for genomic Q-grams. The trie-based hashing function was about 5 times faster than the bitwise hashing function and about 4 times faster than the perfect hashing function. The refined perfect hashing function was also efficient but not as good as the trie-based. The running time of our proposed hashing functions kept good performance even when the word size increased. The result also showed that the bitwise and the perfect hash functions were relatively slow due to their expensive calculations. The performances of the bitwise and perfect hashing functions decreased with the word size. Our proposed refined perfect hashing and trie-based hashing functions outperform than the other two hashing functions.

3.7.2 Effect of Utilization on the Trie-based Implementation

The space requirement is one of the main considerations of the feasibility of an indexing scheme. The bitwise, perfect and refined perfect hashing functions do not require any extra space to calculate the hash values of genomic words. The trie-based hashing stores the pre-calculated loop back values in order to enhance the speed. This section serves to evaluate the space requirement of the Minimal Trie and Perfect Trie.

For the Perfect Trie, the size of failure table is the all possible n-grams for the genomic sequence. The size of a Perfect Trie table is $L^n \times$ (size of a pointer), where L is the number of alphabet characters. But, in order to achieve higher speed, L may be set larger so that it is a power of 2. Like Bitwise, we prefer to use the inexpensive shift operations rather the power or the module operations. For example, the number of alphabet character in amino acid is 20 and then we should set L to 32. It would enhance the efficiency but requires more space.

The Minimal Trie is more suitable for sparse data because only those words appearing in the database are stored in the failure function table. We need to use an indicator to evaluate the dataset whether they are sparse enough or not. The utilization is the number of unique words in a genomic database to

that of a perfect table. It can help to determine which implementation should be used.

$$\text{Utilization} = \frac{\text{No. of unique index entry of a genomic database}}{\text{No. of all possible ngram}}$$

In this experiment, our aim was to discover the utilization of a genomic database. We also evaluated the effects of different size of data and word length on the utilization. For the nucleotide sequences, *L* set to be 4 so that the discovered utilization could directly reflect the distribution of genomic sequences in gbest.seq.

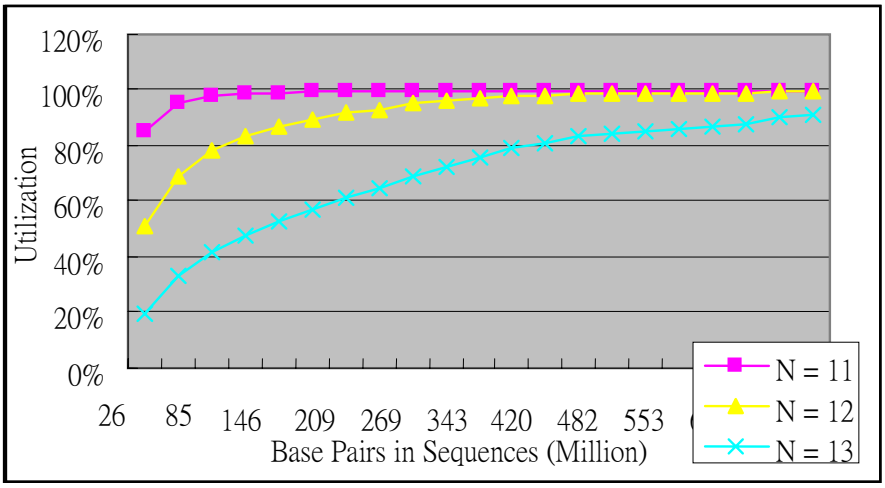


Figure 3-6 The utilization of a genomic sequence file for different n=11, 12 and 13

As shown in Figure 3-6, the utilization of the genomic database steadily increased with the sequence bases. It was expected that the utilization for *N*=13 will also reach 100% when more sequences are inserted. This experiment showed that the failure function table of the trie-based could be fully utilized for indexing this genomic database provided that *L* was not enlarged. Thus, the utilization of the index table was affected by the number of alphabets, the size of n-gram, the genomic database size, and the distribution of sequences. Perfect Trie should be used if it is expected the utilization is high and vice versa. Therefore, we can decide to select the suitable search methods according to the size of dataset and the running environment. If the memory space is large enough to hold the failure function table, the trie-based hashing could be used to achieve higher

efficiency. If there is not enough memory space for the table, the refined perfect hashing should be chosen. Both of these two approaches are better than the existing ones. We believe that more genomic databases will choose to apply indexed homology search in the near future.

3.7.3 Effect the number of sequences on Size of Inverted Index

QgramSearch utilizes the inverted index to speed up the search process. In this experiment, we evaluated the inverted index size under different number of sequences in the database. The program read an EST file and broke each sequence into Q-grams. Then all the Q-grams were inserted into the inverted index using QDBM. As shown in Figure 3-7, the inverted file size increased linearly with the number of genomic sequences.

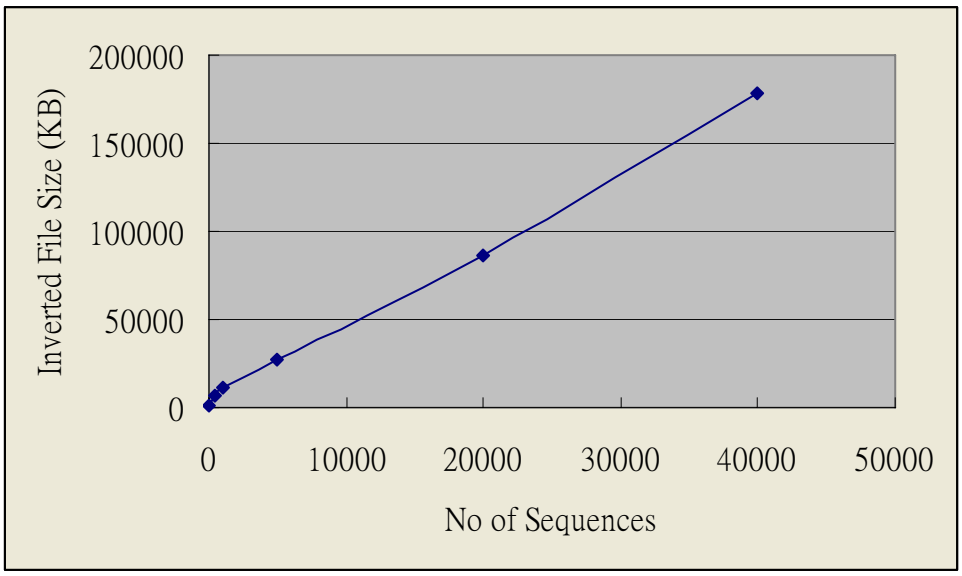


Figure 3-7 The inverted index size under different number of genomic sequences.

3.7.4 Effect of Number of Sequences on Indexing Time

In this experiment, we evaluated the computational time of index creation for different number of sequences in the database. In this experiment, we measured the running time of the whole index creation process. The computation time included reading the EST file, Q-gram generation, dictionary structure computation and storing data in the inverted file. As shown in Figure 3-8, the index creation time grew exponentially with the number of sequences in database.

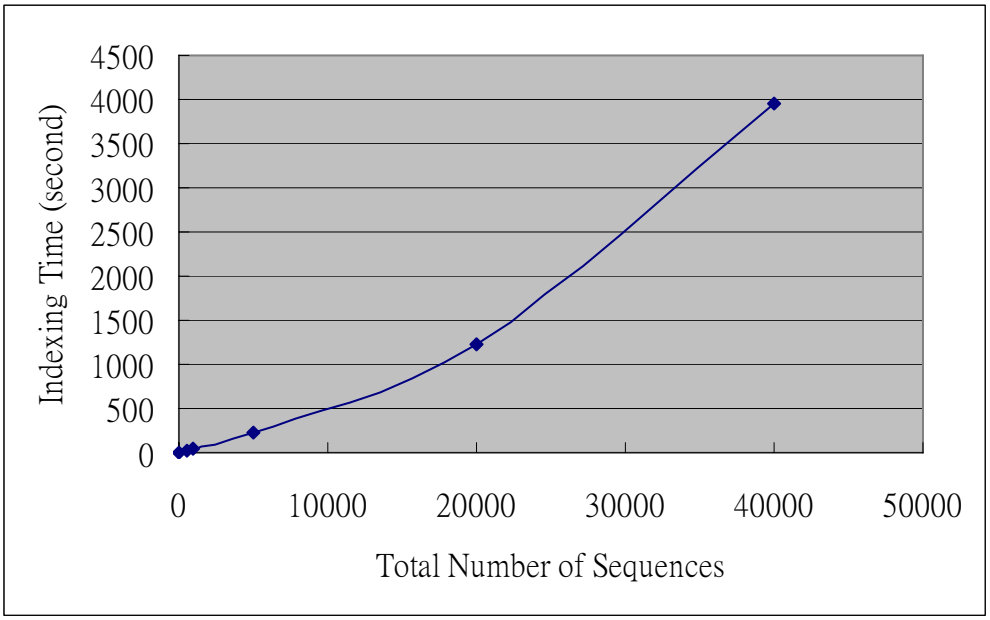


Figure 3-8 The inverted index time under different number of genomic sequences.

Index maintenance strategies on invested files are mainly employed an in-place or merge-based update scheme. For in-place scheme, the in-memory

posting lists are appended to the existing ones on-disk lists. It requires the relocation of the existing on-disk list. Since relocating existing on-disk list is time-consuming, some adopted overallocation strategy that leaves some amount of free space after every on-disk posting list.

For merge-based strategies, a new on-disk inverted file is created by combining an existing one with the in-memory data. In general, merge-based strategies are better at dealing with short posting lists, while in-place strategies are better at handing long posting lists.

It is claimed that the overall performance of the proposed hybrid strategy could be improved so that the indexing time is linear in the size of the text collection. However, the main shortcoming is their slightly reduced query processing performance due to internal fragmentation in the on-disk posting lists [47].

Effect of Number of Sequences on Query Time

In this experiment, we evaluated the computational time of query search using different number of sequences in the database and using different lengths of query sequence. We measured the searching time for a query sequence. The experiment was to evaluate the time of a query search under different collection sizes in the database. The number of sequences in database increased from 50 to 40000. When the sequence collection increased in a large proportion, the query time only rose in a relatively small proportion. It is because the inverted index can allow access a small portion of database sequences instead of the whole database. The inverted index makes the search scalable for large genomic database.

Besides, we used two different lengths of query sequences: one query sequence was of length 50 and the other was of length 200. As shown in Figure 3-9, the query time increased with the length of query sequence. Generally, the query length ranges from tens to hundreds bases. The longer the query sequence is, the more number of computational operations are performed.

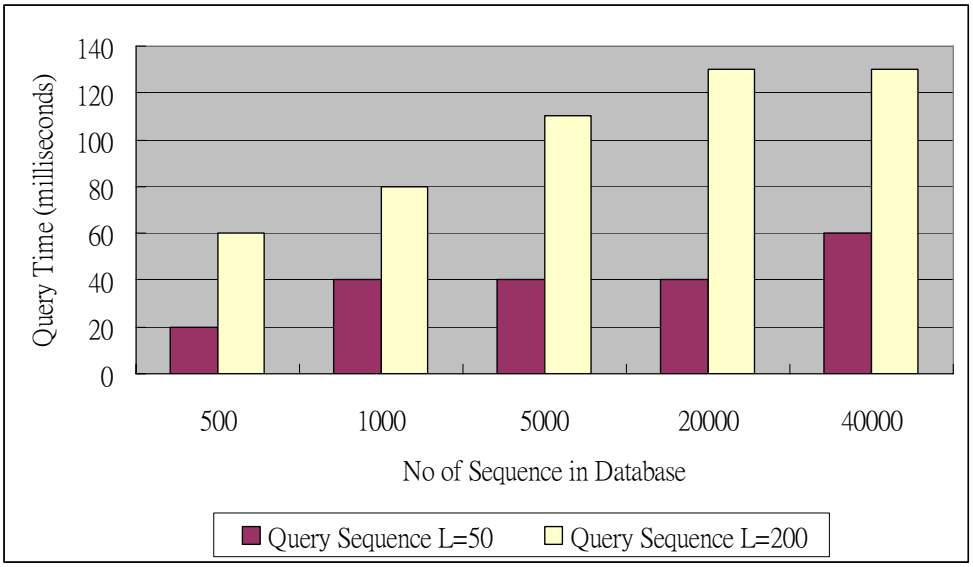


Figure 3-9 The query time is taken under different number of sequences and query length.

Furthermore, we compared QgramSearch with other exhaustive searches. For a small database collection, we believe our proposed search has similar performance as the programs like BLAST. In our experiment, both of the programs could run within 1 second to reply a query. In some cases, the query time remained unchanged even the number of database sequences increased for same sequence length. It is because the system processes a series of Q-gram in query and counts the number of matching Q-grams in the database. The query time mainly depends on the query length and slightly depends on the number of matching Q-grams since the latter operation could run faster. Besides, since the number of database sequences in the above experiment is a relatively small portion to the number of possible Q-gram, the number of matching Q-grams could be more or less the same under some cases. Thus, the query time could be the same under some cases.

However, according to CAFÉ's experiment, as the data size increased, the search times of BLAST and FASTA grew rapidly. For example, BLAST takes about 15 seconds per query under 500 Mb data size, but the search time rises to about 180 seconds for 700Mb data size. QgramSearch adopts the trie-based index and thus search time is proportional to the query and the posting lists. The running time of our proposed rises slowly compared with the collection size.

3.8 Summary

In this chapter, we have proposed an efficient QgramSearch (Q-gram Based Genomic Search). It has an advantage over the exhaustive search tools on the process of sequence matching. We use the inverted index on Q-gram to retrieve the homologous sequences from database instead of the expensive full database scanning. Besides, we introduce two novel dictionary data structures for the retrieval and the index creation processes. The Refined Perfect Hashing and the Trie-Based Hashing utilize the overlapping characteristics of genomic sequences in Q-gram. The experiments show that our two proposed dictionary data structures are faster than other data structures. Compared with the Bitwise hashing, our proposed hashing functions run about 5 times faster. Index-based genomic search generally face two problems: the problem of heavy workload on the servers caused by thousands of queries per day and the problem of long index creation. Our proposed structures can efficiently generate the hash value for the consecutive Q-gram.

QgramSearch uses a normalized weighted score to measure the similarity of sequences. The normalized weighted score uses the probability of the occurrence of Q-gram. This weighted measurement is to take the consideration of the statistical significance of Q-gram. It can help to identify the significant features in the sequences.

The experiments showed the query time of QgramSearch was almost linearly proportional to the number of database sequences. Our proposed index-based approach is more scalable than the existing search tools like BLAST.

4 Q-gram Based Genomic Sequence Clustering

4.1 Introduction

Genomic sequence clustering has been used in the identification of protein families; the prediction of the families for a newly discovered sequences and medical treatment. Clustering is to partition a set of data such that intra-cluster distance is small and inter-cluster distance is large.

The traditional clustering algorithms perform the sequence comparison for each sequence pair in database. The number of possible sequence pairs is usually very huge. This exhaustive pairwise sequence comparison is expensive for a large genomic sequence database containing billions of sequence.

In this chapter, we introduce a novel (QgramClust) Q-gram based genomic sequence clustering to improve the speed of clustering process. It has several advantages over the existing clustering approaches. Firstly, most existing sequence clustering approaches require exhaustive database scans for sequence comparison. We reduce the number of database scanning by using inverted index. Secondly, many clustering approaches adopt dynamic

programming to find sequence alignment, but they are so slow and take $O(n^2)$ time. Q-gram based similarity measurement is used instead of the expensive sequence alignment. Lastly, most of the existing hierarchical clustering methods regard every sequence as a single data point. Since the hierarchical clustering requires n^2 comparisons and space, the performance is not good due to the large number of database sequences. A hybrid way of partitioning and hierarchical clustering is adopted in our algorithm such that the scalability and the efficiency can be balanced.

4.2 P-Similar Neighbors

The concept of neighbor we use is derived from KNN (K-Nearest-Neighbor). KNN has been applied in search and classification. It is an efficient strategy to solve a computationally heavy task by finding the k nearest neighbor. In KNN classification, a data is classified as the label based on the labels of its neighbors [56]. KNN is efficient because the number of data being processed is a small proportion of the whole database collection. It would be good if we utilize the concept of neighbor to reduce the number of sequence comparison. Therefore, we introduce nearest-neighbor in our clustering algorithm

Two sequences are said to P-similar if the number of Q-gram they share is greater than threshold P. A set of data points is the neighbors of a specific point M if they are P-similar to that point M. We called the point M is the

centroid of these neighbors. The idea of QgramClust is to find the neighbors of some regions and then merge the closest regions together.

4.3 Cluster Assignment

As mentioned in previous chapter, the hierarchical clustering takes $O(n^2)$ complexity in time and space. It is not scalable for a large collection. Therefore, we apply a hybrid of the partitioning clustering and the hierarchical clustering. By this way, the number of data points is reduced in the hierarchical clustering process. In the first step, we perform the partitioning clustering --- the close neighbors are grouped into a number of clusters. Figure 4-1 shows the procedure of partitioning clustering. Firstly, the algorithm randomly selects a sequence as a centroid and finds all P-similar neighbors by using our inverted index. Then the centroid and its neighbors are clustered into a primary cluster. The algorithm repeats to select centroids and formulate clusters until all database sequences are grouped into clusters. If some sequences do not have P-similar neighbor sequences, they are formed into primary clusters with only single sequence. A list of primary clusters of P-similar neighbors are returned and passed to the second step.

In the second step, we perform a hierarchical clustering on the primary clusters. A two-dimensional scoring matrix is created and filled with the distances between clusters. The distance of two primary clusters is based on

the distance of their cluster centroids. Agglomerative hierarchical clustering is applied for merging clusters and single linkage is used for distance measurement for clusters.

Partitioning Clustering

Given a set of database sequence S and a threshold T
Define an array of clustered status and an array of score for S

Step 1: Mark all sequences in S with status = "N"
Step 2: Set all sequences in S with score= 0
Step 3: Randomly select a sequence S_i with status = "N" in S
Step 4: Create a cluster C_x and set S_i with status = "Y"
Step 5: Search the posting list for S_i using inverted index
Step 6: Count the number of shared Q-gram as score for each sequence in the set S
Step 7: Mark the score for sequences with status = "N"
Step 8: Overwrite the score for sequences with status = "N" if new score is higher than old score
Step 8: Assign those non-clustered sequences whose score >= T to C_x and set them with status = "Y"
Step 9: Re-assign clustered sequence if new score is higher than old score
Step 10: Repeat and goto Step3 until no sequence in S with status = "N"

Figure 4-1 The algorithm of partitioning clustering approach.

4.3 Efficiency and Effectiveness

In our algorithm, we use Q-gram based approach to measure the similarity of database sequences. Comparing with those clustering algorithms requiring pairwise alignment of each sequence pair in the database, the similarity measurement using Q-gram is more efficient. For clustering a large sequence database, those algorithms using the pairwise alignment is not acceptable. Some heuristic clustering algorithms have better performance than the alignment-based algorithms. For example, BlastClust is one of the famous heuristic clustering algorithms, which is based on the pairwise matches found using the BLAST algorithm and cluster sequences using the single-linkage method. In BlastClust, two sequences are considered to neighbor if the coverage and the Blast score are above some certain thresholds. However, BlastClust requires to measure Blast score of each sequence pairs and it takes $O(n^2)$ complexity in running time. In our proposed algorithm, the similarity of between a centroid sequence and database sequences can be obtained using the inverted index. The running time depends on the number of selected centroids or the number of clusters. In normal case, the number of clusters should be comparatively small to the number of sequences.

For effectiveness, the sensitivity is affected by the selected word size or Q-gram. The smaller word size means that the smaller regions are also considered in the similarity measurement. If a large word size is used, some shared regions are missed. BlastClust uses an input parameter to control the

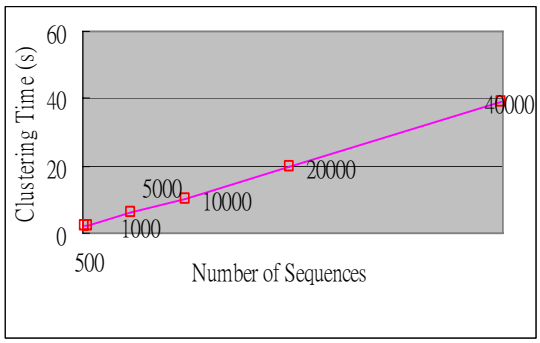
word size for initial matches. The performance of BlastClust drops greatly if a smaller word size is used. Under same word size, the experiment showed our algorithm could run faster than BlastClust.

4.4 Simulation Results

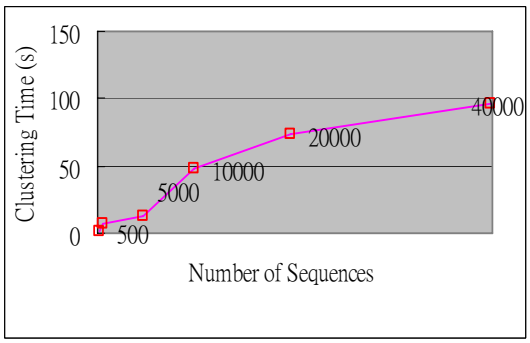
We conducted some experiments to evaluate the performance of QgramClust. Since the partitioning clustering was the core component of our algorithm, the experiments focused on the performance of the partitioning clustering process. We also compared the performance of QgramClust and BlastClust with different collection size. The following experiments were conducted in the same platform as that in the Chapter 3. The same Genbank EST dataset was used and the size of Q-gram was 9.

4.4.1 Effect of Number of Sequences on Clustering Time

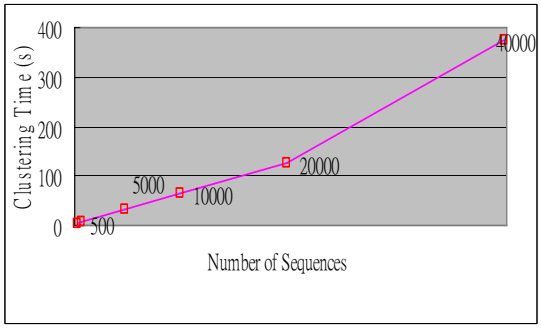
In this experiment, we evaluated the partitioning clustering time of QgramClust using different number of sequences in the database and different similarity threshold. We used several datasets ranging from 500 to 40000 sequences. Different similarity threshold values were also used to control the similarity neighbors in the clusters.



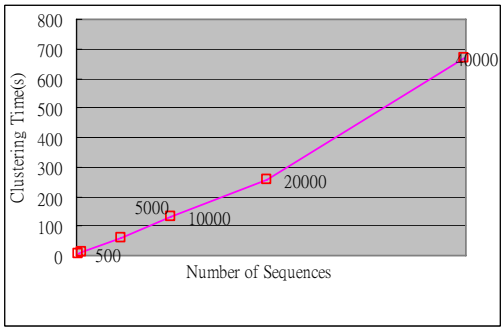
Threshold =3



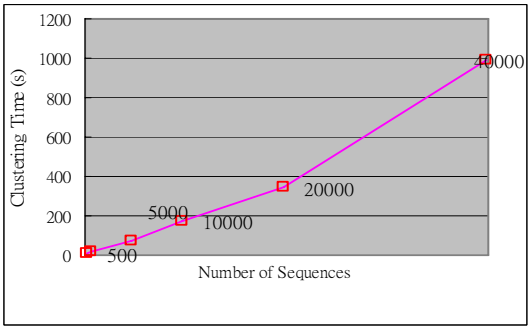
Threshold =6



Threshold =9



Threshold =12



Threshold =15

Figure 4-2. Plot the clustering time and the number of sequence under different threshold

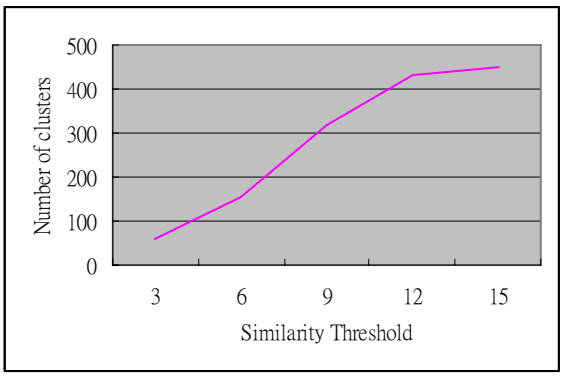
Table 4-1 The result data of clustering time and the number of sequence under different threshold

Number of Sequences/ Threshold	500	1000	5000	10000	20000	40000
T = 3	2	2	6	10	20	39
T = 6	2	7	13	48	73	96
T = 9	4	8	32	64	127	373
T = 12	6	10	57	131	254	667
T = 15	7	15	69	169	346	989

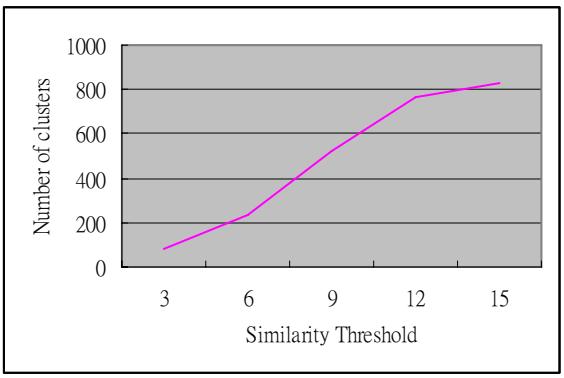
Figure 4.2 showed the clustering time of QgramClust with different collection size. Under the same similarity threshold values, we found that the clustering time was almost linearly proportional to the number of sequences in the dataset. Our proposed algorithm is scalable and the running time would not exponentially increases with the collection size. Another factor on the algorithm is the similarity threshold. From the above results, we discovered that the clustering time increased with the similarity threshold. It is because the higher similarity threshold causes fewer neighbors assigning to each cluster and more clusters to be generated. Some of clusters were formed with only one single data point when using large threshold values. The following experiments showed the relationships of the number of clusters generated and the similarity threshold.

4.4.2 Effect of Similarity Threshold on Number of Clusters

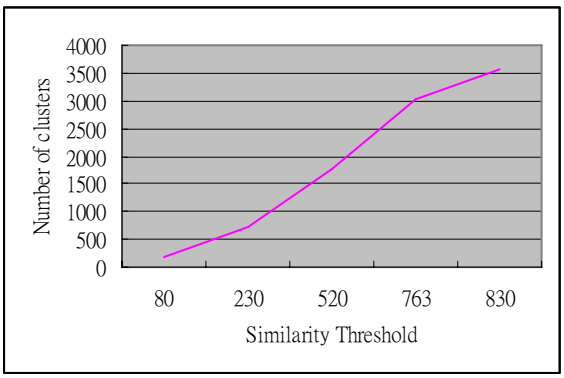
This experiment was to evaluate the numbers of clusters is formed under different similarity threshold. We evaluated six sets of data and counted the total number of clusters. The total number of clusters formed must not be more than the number of data in the dataset.



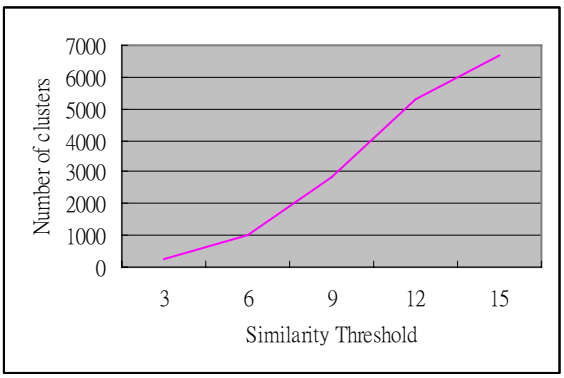
Dataset Size is 500



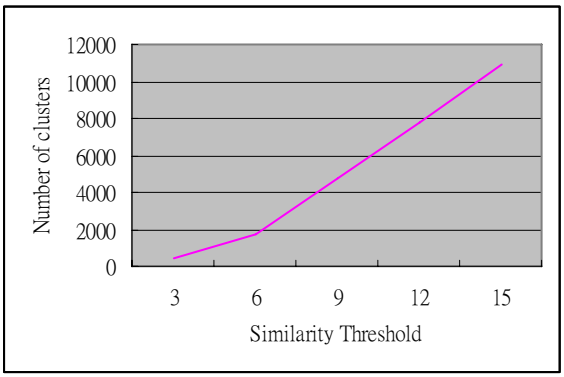
Dataset Size is 1000



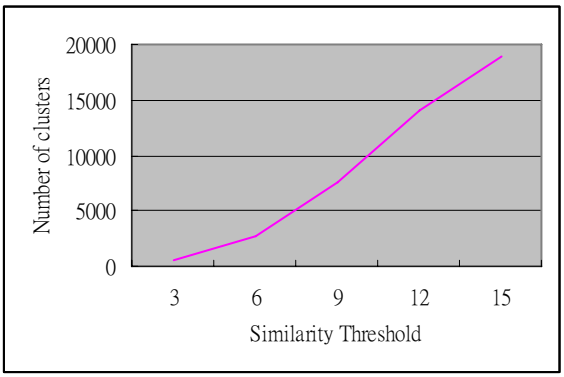
Dataset Size is 5000



Dataset Size is 10000



Dataset Size is 20000



Dataset Size is 40000

Figure 4-3. Plot the relationship of the number of clusters and the similarity threshold under different dataset size

In this experiment, we found the number of clusters formed by the algorithm grew with similarity threshold values. In Figure 4.3, the growth rate of the clusters formed slowed down when the number of clusters reached the dataset size.

4.4.3 Comparison of BlastClust and our proposed algorithm

We also evaluated the performance of BlastClust. We downloaded the executable file of BlastClust and ran it under the same platform as the previous experiments. Figure 4-4 showed the clustering time of our proposed ran faster than BlastClust under 500 to 40000 sequences. The clustering time and the number of clusters formed using BlastClust under its default parameters (word size = 28) were shown in Table 4-2.

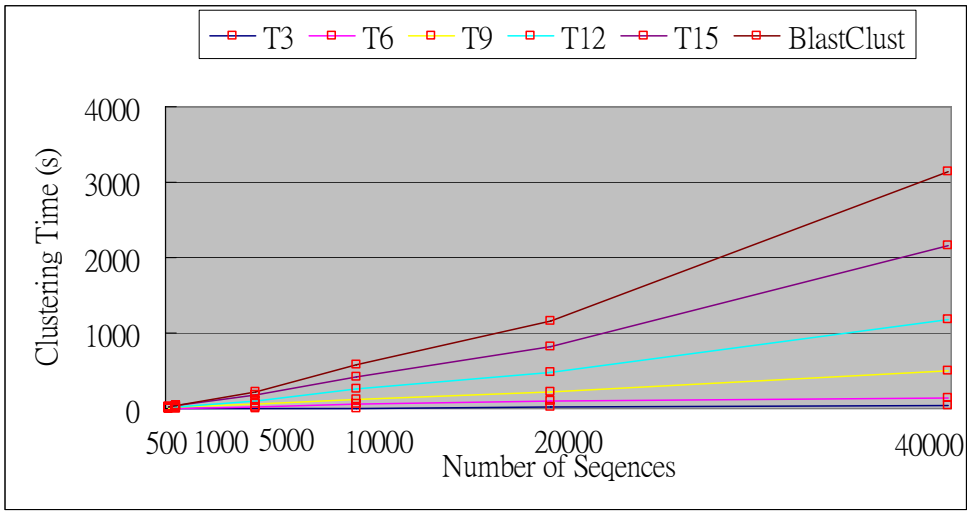


Figure 4-4. Plot the relationship of the clustering time and the number of sequences using BlastClust and our algorithm with several threshold values.

Table 4-2 The result data of clustering time and the number of sequences using BlastClust

No of Sequences	Running Time (s)	No of clusters generated
50	2.7	50
500	3.7	495
1000	6.2	983
5000	42.1	4849
10000	130.6	9617
20000	343	19057
40000	975.9	38016

4.5 Summary

In this chapter, we propose a novel QgramClust, which is a hybrid way of partitioning clustering and hierarchical clustering. A large public genomic database consists of millions to billions of sequences. Since a hierarchical approach takes O (n^2) in time and space, it is not feasible for such huge amount of data. Therefore, we introduce the partitioning clustering as the first step to generate the primary clusters and reduce the number of data passing to the hierarchical clustering.

The traditional clustering algorithms perform the sequence comparisons for each pair. We introduce inverted index to speed up the process sequence comparisons. Besides, we introduce the concept of nearest neighbors in the clustering algorithm and define a P-similar measurement to find the members of clusters. By doing this way, the inter-clusters are dissimilar and the intra clusters are similar.

The experiment showed our algorithm was generally faster than a famous genomic clustering, BlastClust. Besides, it is found that our algorithm is able to discover more distant sequences that cannot be shown in BlastClust. Our clustering algorithm can also facilitate the outliner discovery by finding the clusters with a single sequence. It is an efficient and effective genomic clustering algorithm.

5 Conclusions

In this research, our aim is to devise a new effective and efficient way to mine the information from the genomic sequences. Genomic homologous sequence search and genomic sequence clustering are two important applications to extract knowledge from huge amount of biological data. Since the collection size of the public genomic sequence database is growing exponentially, the traditional genomic search and clustering techniques face the scalability problem. To overcome these demanding challenges, it is necessary to devise new algorithms that can run efficiently and effectively under the increasing amount of data. Therefore, we propose an efficient and effective QgramSearch and QgramClust. The contributions of this research are follows:

1. This research introduces a QgramSearch. We utilize the indexing technique on Q-gram to speed up the search process. Dictionary lookup is a fundamental function in both index creation and retrieval processes. We propose two novel dictionary data structures: Refined Perfect Hashing and Trie-Based Hashing. In the experiment, our proposed data structures run much faster than other data structures. Compared with Bitwise hashing --one of the fastest hashing functions, our proposed hashing function can run 5 times faster.

2. We introduce a normalized weighted score that uses the probability of the occurrence of Q-gram. This takes the significance of Q-gram into consideration. It can efficiently and effectively measure the similarity of two sequences. We also add the factor of the length of sequences into the score. This can reduce the dominance of long sequences in the results.
3. QgramSearch can quickly find the similar sequences. It cannot only be used as a stand-alone search tool but also act as a selection step for other search tools. Since our search process is light, it is probably working a filter to find the high-scoring matches for further processing. For example, dynamic programming can be the final processing to rank the results.
4. QgramClust is a novel way to cluster sequences based on the nearest neighbors of centroid. There has been some research on KNN classification but few researchers conduct research on nearest neighbors clustering method. We find that the nearest neighbor clustering is well-suited for the sequence clustering problem. Comparing with traditional approaches requiring pairwise sequence comparison, our proposed approach can efficiently partition similar sequences.
5. We introduce a novel idea to use indexing technique in sequence clustering. With the inverted index, the nearest neighbors of a centroid can be quickly found and form primary clusters. Our proposed algorithms is more efficient than the traditional sequence clustering

algorithms because it does not require expensive pairwise comparsions. Our experiment showed our clustering algorithm could run faster than BlastClust.

6. Our proposed clustering algorithm is a hybrid approach of partitioning clustering and hierarchical clustering. Since the partitioning clustering step can help to reduce the number of records passing to the hierarchical clustering step, it is scalable to handle a huge amount of data. The similarity threshold can be used to control the size of clusters and the number of primary clusters.

We believe several issues and works that can be done in the future.

1. Semi-supervised learning can be applied in our algorithms. The biological data usually contains some label data and lots of unlabel data. For example, biologists have identified some well-known protein families. The new research is to consider how to utilize the label data to improve the quality of data. Firstly, the future work can include an investigation on the effect of cluster quality after using the label data as the cluster centroids. Secondly, it can include designing a new weighted score. If sequences of same label frequently share a set of Q-gram, the weight can be added. Thirdly, some long and frequent pattern in the sequence of same label can be extracted and they can be used as features in similarity measurement.
2. In our current clustering approach, a sequence can be re-assigned to a newly created cluster since each sequence can only belong to one cluster.

Therefore, a sequence could only be assigned to the closest cluster. In the real biological applications, there might be inter-relationship between clusters. It is worth to doing further research on membership of multiple clusters. Each sequence can be associated with more than one cluster.

3. Compression techniques can be used to reduce the size of inverted index. The inverted index increases with the number of database sequence. The future work can develop a technique to compress the inverted index in order to reduce the storage size.

References

[1] Genbank overview.2006, http://www.ncbi.nlm.nih.gov/Genbank/

[2] A. Califano and I. Rigoutsos. Flash: A fast look-up algorithm for string homology. Computer Vision and Pattern Recognition, Proc. CVPR 93, IEEE Computational Science & Engineering,NY, USA, pp. 353-359, 1993.

[3] A. Chattaraj and H. E.Willams. Variable-length Intervals in Homology Search. Proc. Second Asia-Pacific Bioinformatics Conference (APBC2004), Dunedin, New Zealand, vol. 59, pp. 85-91, 2004.

[4] A. K. Jain and R.C. Dubes. Algorithms for Clustering Data. Prentice-Hall, Englewood Cliffs, NJ, 1998.

[5] A. L. Delcher, S. Kasif, R. D. Fleischmann and J. Peterson al et. Alignment of whole genomes, Nucleic Acids Research, vol 27(11), pp.2369-2376, 1999.

[6] A. Panuccio, M.Bicego and V.Murino. A Hidden Markov Model-Based Approach to Sequential Data Clustering, Springer Berlin, vol 2396, pp.734, 2002

[7] B. Bergeron. Bioinformatics computing, Prentice Hall PTR, 2002

[8] B. Ma, J. Tromp and M. Li. PatternHunter: faster and more sensitive homology search. Bioinformatics, Oxford University Press, vol.18(3), pp. 440-445, 2002.

[9] 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 Biosciences, vol 11(3), pp. 273-279, 1995.

[10] C. Silverstein. A practical perfect hashing algorithm. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 59, pp. 23-47, 2002.

[11] D. D. Sleator and R.E. Tarjan. Self-adjusting binary search trees, Journal of the ACM, vol. 32, pp. 652-686, 1985.

[12] D. F. Feng, and R.F. Doolittle. Progressive sequence alignment as a prerequisite to correct phylogenetic trees, Journal of Molecular Evolution. vol 25(4), pp. 351-360, 1987.

[13] D. J. Lipman and W.R. Pearson, Rapid and sensitive protein similarity searches, Science, vol 227, pp. 1435-1441,1985

[14] E. Bolten, A. Schliep, S. Schneckener, D. Schomburg and R. Schrader. Clustering protein sequences-structure prediction by transitive homology. Bioinformatics, vol. 10, pp. 935–942, 2001.

[15] E. G. Shpaer, M. Robinson, D. Yee, J.D. Candlin and T. Hunkapiller. Sensitivity and selectivity in protein similarity searches: a comparison of Smith-Waterman in hardware to BLAST and FASTA, Genomics. vol. 38(2), pp. 179-191, 1996

[16] E. Giladi, M.G. Walker, J.Z. Wang and W.Volkmuth. SST: an algorithm for finding near-exact sequence matches in time proportional to the logarithm of the database size, Bioinformatics, vol 18(6), 873-879, 2002.

[17] E. Hunt, M.P. Atkinson, R.W. Irving. Database indexing for large DNA and protein sequence collections, The VLDB Journal, The International Journal on Very Large Data Bases, vol 11(3), pp. 256-271, 2002.

[18] G. Yona, N. Linial. and M. Linial. ProtoMap: automatic classification of protein sequences and hierarchy of protein families. Nucleic Acids Research, vol 28(10), pp. 49-55, 2000

[19] H. E. Williams and J. Zobel. Indexing and retrieval for genomic databases. IEEE Trans. on Knowledge and Data Engineering, vol 14(1), pp. 63-78, 2002.

[20] H. E. Williams and J. Zobel. Compression of nucleotide databases for fast searching, Bioinformatics, vol 13(5), pp. 549-554, 1997.

[21] H. E. Williams and J. Zobel. Indexing nucleotide databases for fast query evaluation. Proc. of the 5th International Conference on Extending Database Technology: Advances in Database Technology (EDBT), pp.275-288, 1996.

[22] H. E. Williams, J.Zobel and S.Heinz. Self-adjusting trees in practice for large text collections, Software Practice and Experience, vol. 31(10), pp.925-939, 2001.

[23] H. Mihail, S. Nematollaah and T. Anand. Exact match search in sequence data using suffix trees, Proc. of the 14th ACM international conference on Information and knowledge management, Bremen, Germany, pp. 123-130, 2005.

[24] H. Wang, W. Wang, J. Yang and P.S. Yu. Clustering by pattern similarity in large data sets, International Conference on Management of Data, Proc of the 2002 ACM SIGMOD international conference on Management of data, Madison, Wisconsin, pp. 394-405, 2002.

[25] H. Williams. Genomic information retrieval. Proc. of the 14th Australasian Database Conference, Australia, pp. 27-35, 2003

[26] I. S. Kohane, A. Kho and A.J. Butte. Microarrays for an Integrative Genomics, The MIT Press, 2005.

[27] J. Aoe, K. Morimoto, M.Shishibori and K.H.Park. A Trie compaction algorithm for a large set of keys, IEEE Trans on Knowledge and Data Engineering, vol 8(3), pp. 476-491, 1996

[28] J.C. Setubal and J.Meidnis. Introduction to computational molecular biology, PWS publishing company, 1997.

[29] J. D. Thompson, D.G. Higgins and T.J. Gibson. Clustal W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. Nucleic Acids Res., vol 22, pp. 4673–4680, 1994.

[30] J. Yang and W. Wang. CLUSEQ: efficient and effective sequence clustering. Proceedings of the 19th IEEE International Conference on Data Engineering (ICDE), pp. 101-112, 2003.

[31] K. Ian, Y. Mark and B. Joseph. BLAST. O'Reilly and Associates, Dunedin, 2003.

[32] Kriventseva, E. V., Fleischmann, W., Zdobnov, E. M., Apweiler, R. CluSTr: a database of clusters of SWISS-PROT+TrEMBL proteins. Nucleic Acids Res, vold 29, pp. 33-36, 2001

[33] L. J. Wong and J. Li. Identifying good diagnostic gene groups from gene expression profiles using the concept of emerging patterns, Bioinformatics, vol 18: pp. 725-734, May 2002.

[34] L. Shang and T.H. Merretal. Trie for approximate for string matching, IEEE Trans. on Knowledge and data engineering, vol 8(4), pp. 540-547, 1996.

[35] M. Brain and A. Tharp. Using Tries to eliminate pattern collisions in perfect hashing, IEEE Trans on Knowledge and Data Engineering, vol 6(2), pp. 239-247, 1994

[36] M. B. Eisen, P.T. Speman and P.O. Brown al et. Cluster analysis and display of genome-wide expression patterns, Proc. of Natl. Acad. Sci, USA, vol 95(02), pp. 14863-14868, 1998

[37] N. C. Jones and P. A. Pevzner. An introduction to bioinformatics algorithms, The MIT Press, 2004

[38] N. M. Luscombe, D.Greenbaum and M.Gerstein. What is Bioinformatics? A proposed definition and overview of the field, Method Inform Med. , 40(4), 346-358, Apr 2001.

[39] P. Baldi and S. Brunk. Bioinformatics, The Machine Learning Approach, The MIT Press, 2001

[40] P. Pipenbacher, A. Schliep, S. Schneckener, A. Schonhuth al etc. ProClust: improved clustering of protein sequences with an extended graph-based approach, Bioinformatics, vol. 18, pp. 182-191, 2002.

[41] R. Sharan and R. Shamir. CLICK: a clustering algorithm with applications to gene expression analysis. Proc. International Conference on Intelligence Systems for Molecular, vol. 8, pp. 307-316, 2000.

[42] R. Xu and D. Wunsch. Survey of clustering algorithms. Proc. of Neural Networks, IEEE Trans, vol 16(3), pp. 645-678, May 2005

[43] S. Altschul, T. Madden, A. Zhang and J. Zhang al et. Grapped BLAST and PSI-BLAST: a new generation of protein database search programs. Oxford University Press, Nucleic Acids Research, vol. 25(17), pp. 3389-3402, 1997

[44] 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, vol. 48, pp. 443-453, 1970.

[45] S. Basu, A.Banerjee and R. J. Mooney. Semi-Supervised Clustering By Seeding. Proc. of 19th International Conference on Machine Learning(ICML), Australia, pp. 19-22, 2002.

[46] S. Basu, M. Bilenko and R. J. Mooney. A probabilistic framework for semi-supervised clustering, Proc. of the 10th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 59-68, 2004.

[47] S. Buttcher, C.L.A Clarke and B. Lushman. Hybrid index maintenance for growing text collections. Proceedings of the 29th annual international ACM Conference on Research and Development in Information Retrieval (SIGIR 2006), Seattle, pp. 356-363, August 2006.

[48] S. F. Althschul, W. Gish, W. Miller, E.W. Meyers, and D.J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology,* vol 215, pp. 403-410, 1990.

[49] S. Heinikoff and J.G Heinikoff. Amino acid substitution matrices from protein blocks, Proc. of the National Academy of Sciences of the United States of America vol. 89, pp. 10915-10919, 1992.

[50] S. Heinz, J. Zobel and H.E. Williams. Burst tries: a fast, efficient data structure for string keys, ACM Trans. on Information Systems, vol. 20 (2), pp. 192-223, Apr 2002.

[51] S. Heinz and J. Zobel. Performance of data structures for small sets of strings, Proc. of the 25th Australasian Conference on Computer Science, Australia, vol 4, 87-94 , 2002.

[52] S. Kim and A. Gopu. BAG: A Graph Theoretic Sequence Clustering Algorithm, International Journal of Data Mining and Bioinformatics, 1(2), pp. 178-200, 2006.

[53] S. Ranjan and Z. Justin. Efficient Trie-based sorting of large sets of strings, Australasian computer science conference, Adelaide, Australia, pp. 11-18, 2003

[54] T. H. Larry, F.L. Chung and C.F. Stephen and Simon M.C. Yuen. Using emerging pattern based projected clustering and gene expression data for cancer detection, Proc. of the second conference on Asia-Pacific, pp. 75-84,2004

[55] T. Kahveci and K. Tamper. Map: searching large genome databases, The Pacific Symposium on Biocomputing Conference, Hawaii, pp. 11-18, 2003

[56] T. Liu, A.W. Moore and A.Gray. Efficient exact k-NN and nonparametric classification in high dimensions, Proc. of Neural Information Processing Systems, vol. 15, 2003

[57] T. Morzy, M. Wojciechowski and M. Zakrzewicz. Scalable Hierarchical Clustering Method for Sequences of Categorical Values, Proc. of the 5th Pacific-Asia Conference on Knowledge Discovery and Data Mining, pp. 282-293, 2001.

[58] V. Alfred and J. Margaret. Efficient string matching an aid to bibliographic search, Bell Laboratories, vol. 18(6), pp. 333-340, 1975

[59] V. Guralnik. and G. Karypis. A scalable algorithm for clustering sequential data, Proc. of the 2001 IEEE International Conference on Data Mining, IEEE Computer Society, pp. 179-186, 2001.

[60] V. Lertnatte and T. Theeramunkong. Multidimensional text classification for drug information, IEEE Trans. on Information technology in biomedicine, vol 8 (3), Sept 2004.

[61] W. J. Kent. BLAT: The BLAST-like alignment tool. *Genome Research*, 12 (4), pp. 656-664, 2002.

[62] W. J. Wilbur and D.J. Lipman. Rapid similarity searches of nucleic acid and protein data banks. Proceedings of the National Academy of Sciences of the USA 80, pp. 726-730, 1983.

[63] W. R. Pearson and D.J. Lipman. Improved tools for biological sequence comparison. Proc Natl Acad Sci USA, 85, pp. 2444-2448, 1988

[64] W. R. Pearson and W. Miller. Dynamic programming algorithms for biological sequence comparison, Method in Enzymology, vol 210, pp. 575-601,1992.

[65] Y. Chen, K.D. Reilly, A. P. Sprague and Z. Guan. SEQOPTICS: a protein sequence clustering system, Bioinformatics, vol. 7(4), suppl 4, 2006.

[66] Y. K. Yu. Statistical Significance of Probabilistic Sequence Alignment and Related Local Hidden Markov Models, Journal of Computational Biology, vol. 8(3), pp. 249-282, 2001.

[67] Z. Ning, A.J. Cox and J.C. Mullikin. SSAHA: a fast search method for large DNA databases. *Genome Research*, vol 11, pp. 1725-1729, 2001.