December 2012

Improvements on Seeding Based Protein Sequence Similarity Search

Weiming Li
The University of Western Ontario

Supervisor
Dr. Bin Ma
The University of Western Ontario

Joint Supervisor
Dr. Kaizhong Zhang
The University of Western Ontario

Graduate Program in Computer Science

A thesis submitted in partial fulfillment of the requirements for the degree in Doctor of Philosophy

© Weiming Li 2012

Improvements on Seeding Based Protein Sequence Similarity Search

(Spine title: Improvements on Seeding Based Protein Sequence Similarity Search)

(Thesis format: Monograph)

by

Weiming Li

**Graduate Program
in
Computer Science**

**A thesis submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy**

**The School of Graduate and Postdoctoral Studies
The University of Western Ontario
London, Ontario, Canada**

Certificate of Examination

THE UNIVERSITY OF WESTERN ONTARIO
THE SCHOOL OF GRADUATE AND POSTDOCTORAL STUDIES
CERTIFICATE OF EXAMINATION

The thesis by
Weiming Li
entitled:
Improvements on Seeding Based Protein Sequence Similarity Search
is accepted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Date: ____________

Chair of the Thesis Examination Board

Abstract

The primary goal of bioinformatics is to increase an understanding in the biology of organisms. Computational, statistical, and mathematical theories and techniques have been developed on formal and practical problems that assist to achieve this primary goal. For the past three decades, the primary application of bioinformatics has been biological data analysis. The DNA or protein sequence similarity search is perhaps the most common, yet vitally important task for analyzing biological data.

The sequence similarity search is a process of finding optimal sequence alignments. On the theoretical level, the problem of sequence similarity search is complex. On the applicational level, the sequences similarity search onto a biological database has been one of the most basic tasks today. Using traditional quadratic time complexity solutions becomes a challenge due to the size of the database. Seeding (or filtration) based approaches, which trade sensitivity for speed, are a popular choice among those available. Two main phases usually exist in a seeding based approach. The first phase is referred to as the hit generation, and the second phase is referred to as the hit extension.

In this thesis, two improvements on the seeding based protein sequence similarity search are presented. First, for the hit generation, a new seeding idea, namely spaced k-mer neighbors, is presented. We present our effective algorithms to find a good set of spaced k-mer neighbors. Secondly, for the hit generation, a new method, namely HexFilter, is proposed to reduce the number of hit extensions while achieving better selectivity. We show our HexFilters with optimized configurations.

Keywords: spaced seeds, homology search, similarity search, HexFilter, neighbors.

To My Family

Acknowledgments

First of all, I am extremely proud to be a student supervised by my co-supervisors Dr. Bin Ma and Dr. Kaizhong Zhang. To me, they are two of the most inspirational researchers in my life. They unconditionally and persistently supported my research over the past years. I am extremely overwhelmed with delight when considering them as my mentors and lifetime friends.

I have always felt very fortunate that I have spent my post-secondary studies at Western in London, Ontario, Canada. It was definitely a remarkable 10 years of my life which I will never forget.

I would like to thank all my dear colleagues in room MC222, the great friends, the brightest faculty, and the best staff members in the Department of Computer Science at Western.

I would like to thank the examiners of my thesis proposal, Dr. Lila Kari and Dr. Sheng Yu, for their suggestions to improve this work at the earlier stage. And, I would like to thank my thesis examiners, (in alphabetical order) Dr. Lucian Ilie, Dr. Lila Kari, Dr. Jason Wang, and Dr. Xingfu Zou, for their helpful suggestions concerning this work.

Last but not least, I am extremely thankful to my family for their love and support.

Table of Contents

List of Tables

List of Figures

Chapter 1
Introduction to Sequence Similarity Search

The primary goal of bioinformatics is to increase an understanding in the biology of organisms. Computational, statistical, and mathematical theories and techniques have been developed on formal and practical problems that assist to achieve this primary goal. For the past three decades, the primary application of bioinformatics has been biological data analysis. An earlier focus had concentrated on the area of genomics involving large-scale DNA data while in recent years more attention has been on the area of proteomics concerning large-scale proteins data. The production of such biological data is becoming much easier and cheaper. For example, the growth rate of the popular database, GenBank [1], in NCBI is exponential since its first release in 1982, a rate faster than Moore's Law. It took Human Genome Project over 10 years and $3 billion U.S. dollars to completely sequence the first human genome by many international researchers from multiple disciplines. Today, the same task can be done in less than two weeks for about $1500 U.S. dollars by many companies. Enormous volumes of biological data have been generated as a result of the rapid growth in biological data. However, having a collection of such an amount of biological sequences alone does not increase our knowledge on the biology of organisms. Comparing new sequences to similar sequences with known functions can help in understanding the new sequence.

The sequence similarity search is perhaps the most common, yet vitally important task for analyzing biological data. Many applications, for example, the gene and protein predications, phylogeny and evolutionary analysis, and sequences assembly and annotation are based on the sequence similarity search. On the theoretical level the problem of sequence similarity search is complex. It is not a single problem, but a collection of different problems sharing a common base of the sequence alignment. Traditional approaches of sequence alignment include the famous Needleman-Wusch and Smith-Waterman algorithms. These algorithms are guaranteed to find the best alignment between two sequences. On the applicational level, database sequence similarity search is one of the most basic tasks today while efficiency is continually emphasized. On a daily basis there are over 100,000 queries on the NCBI's BLAST server for sequence similarity searches. This increases at a rate of 10% each month. Since the Smith-Waterman algorithms run in quadratic time of the total length of the sequences, they become impractical when invoking large-scale sequences comparisons. So, a target of heuristics is to maintain reasonable high sensitivity and make few as possible calls to the Smith-Waterman algorithm. Seeding (or filtration) algorithm based tools, which trade sensitivity for speed, is a popular choice among other approaches. The seeding based approach runs faster than the Smith-Waterman algorithm, but misses some true homologies. Many clever ideas on the seeding algorithm have been developed which helps to bring a more efficient application to life. The FASTA program was first released in 1985; the BLAST program was first released in 1990; and the PatternHunter program was first released in 2002. These exemplified the evolution of seeding based approaches over the past three decades. The BLAST perhaps is the most widely used homology search tool today. BLAST first finds a match of three consecutive letters as a hit between two sequences. Next, the identified hits are filtered and extended for local alignment. The spaced seed in the PatternHunter took a leap over the consecutive seed used in BLAST. The optimized spaced seed can improve the sensitivity significantly while maintaining similar speed as BLAST. Experimental results have shown that the carefully selected multiple spaced seeds on the protein coding regions can even achieve same sensitivity as the Smith-Waterman algorithm and run at the similar speed of BLAST. The seeding algorithms have been an extremely active area of research.

Two main phases exist in seeding based approaches. In the first phrase, a k-mer seed is used to find a hit. For protein sequences, a hit requires that the score of the subsequence ungapped alignment is greater than a predefined threshold. In the second phase, the hits are extended for finding local alignments. The seeding based approach looks intuitive, but its impacts on sensitivity and speed are influential and profound. Bigger values of k of k-mer result in longer seed, faster speed, and weakened sensitivity. Larger values of the hit score threshold result in less number of qualified hits, faster speed, and weakend sensitivity. Usually, the first phase is referred to as the hit generation, and the second phase is referred to as the hit extension.

In this thesis, two improvements concerning both phases of seeding based protein similarity search are presented. First, for the hit generation, a new seeding idea, namely spaced k-mer neighbors, is presented. We present our effective algorithms to find a good set of spaced k-mer neighbors. The experimental shows that our spaced k-mer neighbors are more efficient because better sensitivity and same selectivity can be achieved. Secondly, for the hit extension, a new method called HexFilter, is presented. We propose our algorithm for generating the HexFilters with optimized configurations. The experimental results show that our HexFilter can efficiently reduce the numbers of hit extensions while achieving better selectivity.

1.1 Chapter Outlines

This thesis is organized into the following Chapters:

Chapter 1 gives introduction to problem of sequence similarity search.

Chapter 2 lays down the necessary information related to preliminaries and notations used through this thesis. We review important seeding algorithms and seeding based tools.

Chapter 3 introduces the new idea, namely spaced k-mer neighbors, for seeding protein sequences. We paid detailed attention to reason, design, and the study of our algorithms and related experimental results.

Chapter 4 presents the new filter, namely the HexFilter, for reducing the number of hit extensions. Details are given to the designing of an optimized HexFilter.

Chapter 5 draws the conclusion on this thesis.

Chapter 2

Background

A biological sequence is a series of continuous one kind of monomers in one-dimensional ordering. A monomer is a molecule that can form a polymer by chemically binding to another molecule. Nucleotides and amino acids are two important natural monomers. Nucleotides are polymerized to form nucleic acids that are biological molecules of DNA (deoxyribonucleic acid) and RNA (ribonucleic acid), which are essential for known forms of life on Earth. Amino acids are another natural monomer that polymerize at ribosomes to form proteins. Proteins are single chains of bounded amino acids that fold into different forms for facilitating its biological functions. And, proteins are essential parts of organisms that are involved in virtually every process in cells. Sequence analysis, a key process to study the evolution, function, and structure of new nucleic acids and proteins, helps in understanding the biology of organisms. There are various analytical methods used in sequences analysis. Direct experimentation is the most reliable method. However, computational sequence analysis is far easier and cheaper than direct experimentation. For example, some experiments are very expensive such as those involving chimpanzees, not only does the experiment require longer generation time, but also but also chimpanzees are endangered and expensive. Some experiments are difficult to complete in a laboratory environment such as some kinds of marine bacteria, etc. The Human Genome Project [2] produced more than 20,000 genes, which only a small portion had gone under direct experimentation. New sequences are not invented but come from some existing sequences because of their evolution. Hence, common methods of computational sequences analysis involved in comparing new sequences with other sequences have known functions. Prior to evaluating the similarity between two sequences, a good alignment between them must be found. Hence, sequence alignment is a crucial process to find similar regions between two sequences. The earliest trace of the computational sequence alignment problem can be thought of as perhaps the Edit Distance problem [3] in 1966. Three basic edit operations of insertion, deletion, and letter substitution were defined in order to transform one string to another string. A great, still growing, collection of literature has been developed following that. Some subsequent and strongly influential papers including the original Needleman-Wunsch's dynamic-programming solution [4] for global alignment, the seminal solution on local alignment by Smith-Waterman [5], the BLAST [6] tool in the searching of related sequences for a whole genome, and the PatternHunter [7], have brought us to the new era of seeding based tools.

2.1 Introduction

In this Chapter, some of the necessary notations and preliminaries on sequences alignment will be reviewed. Among many tools available today, FASTA, BLAST, and PatternHunter have been chosen for reviews because they exemplify seeding based tools.

2.2 Preliminaries and Notations

A *sequence* s of length n can be denoted as a string of letters in an alphabet, say Σ.

$$s = a_1 a_2 a_3 \dots a_n$$

In the field of Bioinformatics, for DNA sequences, there are 4 nucleotides in the alphabet. The names and abbreviations of the four nucleotides are in Table 2.1.

For protein sequences, there are 20 commonly used amino acids in the alphabet. The names and abbreviations of the 20 amino acids are in Table 2.2.

Nucleotide	**Abbreviation**
Adenine	A
Cytosine	C
Guanine	G
Thymine	T

Table 2.1: The four nucleotides in DNA.

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

Table 2.2: The most commonly used twenty amino acids in protein.

An *alignment* between two sequences is often referred to as a pairwise sequence alignment, which can be represented as two sequences with same length appearing in a top-down manner. For example, an alignment between sequences of TGAGATA and TGCGAATA is in Table 2.3.

In an alignment, an exact *match* occurs when two letters at a same position are exactly same, for example, the pairs at the first position of (T$\sim$T) is a match.

T	G	A	G	–	A	T	A
|	|	|	|	|	|	|	|
T	G	C	G	A	A	T	A

Table 2.3: An example of an alignment.

A substitution occurs when two letters are not the same, for example, at the third position, the pair of (A∼C) is a substitution.

An *indel* in an alignment is denoted by '-', which means a letter of the sequence aligns to nothing ('-') at that position. Insertion and deletion are two types of indels. A *gap* is a serie of continuous either insertions or deletions. For example, in Figure 2.3 a gap is at the fifth position between ('-'∼A).

There are two types of gap models that are commonly used, namely the linear gap penalty and the affine gap penalty. For the linear gap penalty, a negative constant is given for a single penalty. The gap penalty for a longer gap is simply the summation of a single type of indel penalty at maximal length. The end result is an alignment with less gaps; that is favoured. For the affine gap penalty, a function for the gap penalty of length i is given as $P(i) = g + hi$, where g is the gap opening cost and h is the gap extension cost. The values of g and h are used to control the length of gaps in an alignment, for example, if $g > 0$, then a longer gap is likely to occur.

Optimal alignments are the best alignments that can be found between two sequences with respect to the values of a set of parameters for the letters in Σ.

Global and local alignments are two main types of alignment. Two dynamic programming algorithms are available for finding such optimal alignment. The Needleman-Wunsch algorithm [4] can find the optimal global alignment. And, the Smith-Waterman algorithm [5] can find the optimal local alignment. Gaps are allowed in both algorithms. Both algorithms attempt all possible alignments between two sequences, and the time complexity of both algorithms are $O(mn)$ where m and n are the lengths of two sequences.

Global alignment assumes that similarity between two sequences is from the beginning to the end of two sequences uninterruptedly. For example, between sequence s =TGGGTACTA and sequence t =AGGTACATC, using the BLOSUM62 scoring matrix, the gap opening cost is 10 and the gap extension cost is 0.5. The optimal global alignment between s and t is in Table 2.4.

T	G	G	G	T	A	C	T	A	–
	·	|	|	|	|	|	·	·	
–	A	G	G	T	A	C	A	T	C

Table 2.4: An example of a global alignment.

Hence a global alignment may contain a segment of alignment with a lower score that is not the best interest.

Local alignment, instead of forcing to align from the beginning to the end between two sequences, searches for any segments that are similar enough for one's interest. For example, using sequences s and t and the same parameters in the global alignment, the optimal local alignment is in Table 2.5.

G	G	T	A	C
|	|	|	|	|
G	G	T	A	C

Table 2.5: An example of a local alignment.

While both algorithms can return optimal solutions, the local alignment is usually preferred over the global alignment because the region with a lower similarity score usually can be in the global alignment. For example, the local alignment in Table 2.5 is preferred over the global alignment in Table 2.4 because the mismatches and gaps are not included in the local alignment.

The Needleman-Wusch and Smith-Waterman algorithms, which are quadratic in time complexity, become impractical in involving a large-scale sequence which leads to the investigation of heuristic based algorithms for an efficient trade-off between speed and sensitivity, for example, seeding based algorithms.

2.3 Seeding Based Algorithms

In this Section, three delegable heuristic approaches in sequences similarity search will be reviewed, which are the FASTA program, the BLAST program from NCBI, and the breakthrough of PatternHunter.

2.3.1 FASTA

FASTA is a software package developed by Lipman and Pearson. Because in 1985 the first release of the software package supported only protein to protein homology search, it was released under the name of FASTP [8]. In 1988, DNA to DNA homology search became supported together with other added features, so it was renamed as the FASTA [9]. Since then, the name FASTA has been used to refer to the whole software package.

The FASTA package is perhaps the earliest use of the approximation approach on the seeding algorithm for local similarity search against the protein and DNA sequences database. The FASTA uses the length k identities between two sequences to quickly anchor a potential similar sequence. The more time-consuming Smith-Waterman algorithm would be performed if the length of k identities are found. Therefore, the value of k affects the trade off between the sensitivity and speed of the program. Increasing value of k will decrease the sensitivity but increase the speed. The default values of k for protein homology search is 2 or 3, and 5 or 6 for DNA homology search.

The FASTA algorithm involves few steps. First, a look-up table is built for quickly finding the positions of k-tups of the query sequence. A k-tup is the length k

of consecutive letters. When reading a database sequence, each time a word length of k will be read and compared to the lookup table. If the lookup table also contains the same word, then a *hit* is found. The top 10 best regions of each diagonal, shown in Figure 2.1(a), will be returned by the diagonal algorithms. The details of the diagonal algorithms are in [10, 8]. The diagonal algorithms endorse match and mismatch but not insertion and deletion.

Second, the FASTA uses the 250-PAM scoring matrix to re-score the diagonal segment. The top 10 regions of all diagonals are kept. The returned regions are the HSPs. This is shown as an example in Figure 2.1(b).

Third, a jointing procedure is applied; the HSPs could join together and form a single optimal alignment which are kept. This is shown as an example in the Figure 2.1(c).

Fourth, a modified version of either the Needleman-Wusch or Smith-Waterman algorithms is executed for optimal alignments. The modification is to look for the optimal alignment within a band. The band helps to speed up the alignment process because it constrains the alignment in a search window of interested diagonals. Only a portion of the total length of the sequences are used to create the final alignment result. This is shown as an example in the Figure 2.1(d).

Upon obtaining the final alignment result, FASTA uses a statistical procedure to analyze the significance of alignment results in order to distinguish them from the random alignments.

The FASTA program was very popular in Europe at that time. As a seeding based tool, it was a very successful attempt to speed up the database similarity search. The FASTA software package also contains a release of the Smith-Waterman algorithm. It is called Ssearch and many researchers use Ssearch as the standard implementation of the Smith-Waterman algorithm. Aside from all of the above,

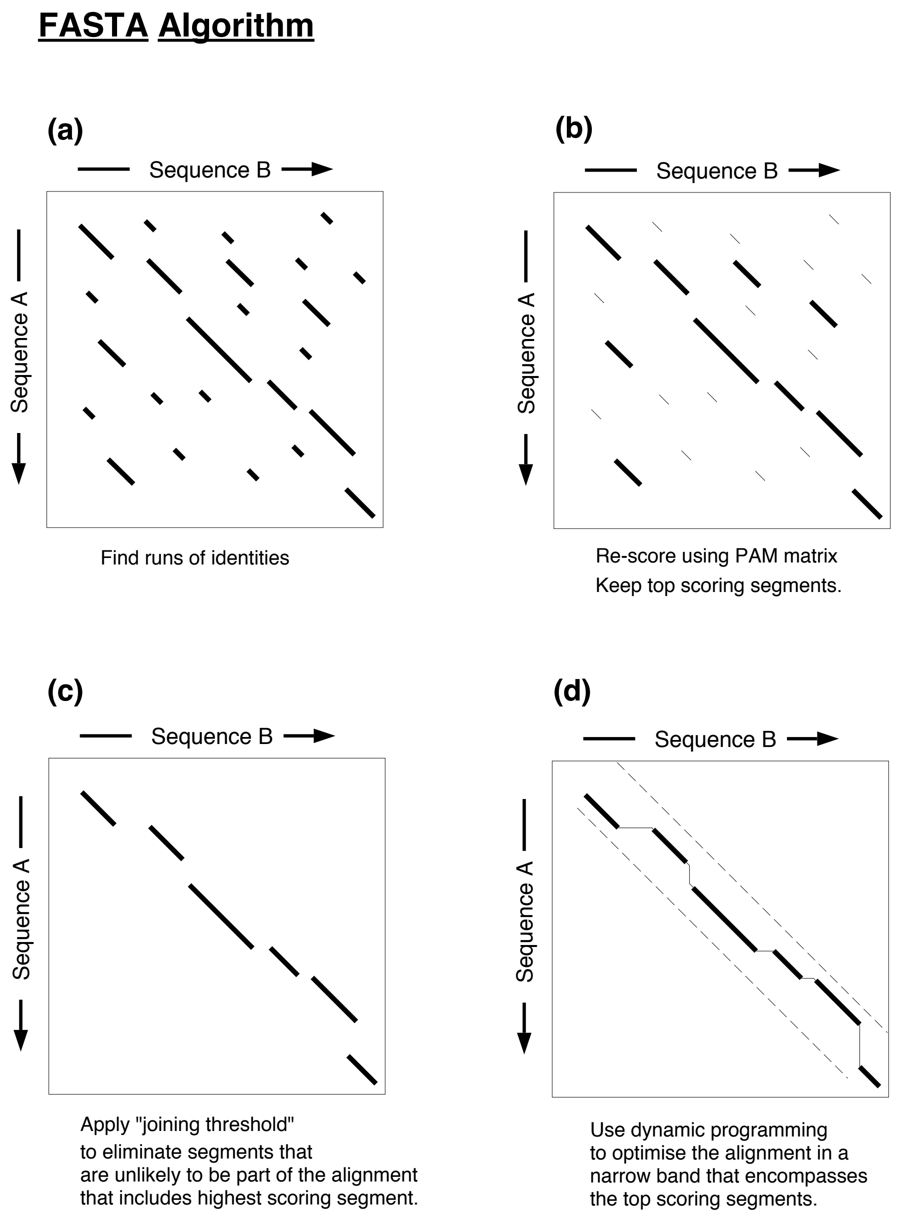
FASTA Algorithm
(a)
Sequence B
Sequence A
Find runs of identities
(b)
Sequence B
Sequence A
Re-score using PAM matrix
Keep top scoring segments.
(c)
Sequence B
Sequence A
Apply "joining threshold"
to eliminate segments that
are unlikely to be part of the alignment
that includes highest scoring segment.
(d)
Sequence B
Sequence A
Use dynamic programming
to optimise the alignment in a
narrow band that encompasses
the top scoring segments.

Figure 2.1: The basic steps of a FASTA algorithm, an original Figure from the book [11].

FASTA is a legacy file format for data sequence which is widely used in Bioinformatics, e.g. NCBI uses it.

2.3.2 BLAST

BLAST, the famous **B**asic **L**ocal **A**lignment **S**earch **T**ool by NCBI (Nation Centre for Biotechnology Information), was first introduced in 1990 [6] for the purpose of quickly finding the ungapped alignments from a sequence database search. It trades sensitivity for speed. It was driven by the high demand of fast-run applications when the size of the biological database expanded exponentially.

BLAST is a combination of great usages of theoretical statistics and combinatorics with many clever heuristics. BLAST was able to identify potential alignment by first seeking a length k consecutive letters of a matched word (*hit*), a more time consuming step of the hit extension is followed on the identified hits. Using the hits and other heuristics help BLAST to stand out among other tools. The NCBI BLAST also provides a valid measurement on whether a sequence alignment result is related. Based on rigorous statistical analysis on random alignments, such measurement is an important indicator if an alignment is not a random alignment.

The BLASTP program is the *de facto* standard database homology searching tool used by researchers across different disciplines around the world. The BLAST program evolved with the development of current computer technology. The first release of BLAST was a standalone application in 1990. It lacked many important features [12]. For example, it did not support gapped alignment. Later, in 1996 [13], BLAST became a web service and expanded its capabilities. For example, the release of the MegaBLAST [14] and gapped BLAST [15] contain most features of today's BLAST. The latest release of BLAST in 2009 [12] improved source codes flexibility by using modular design of ADT (Abstract Data Type). It also started to use the non-consecutive seed (spaced seed) in DNA comparisons. It optimized the usage of cache memory by carefully packing the lookup tables into the cache after splitting the longer sequences. Today, BLAST is capable of comparing different kinds of sequences against a sequences database, for example, tBLAST for translated nucleotide sequences against translated nucleotide database, BLASTp for protein sequence against protein database, BLASTn for DNA sequences against a DNA database, etc.

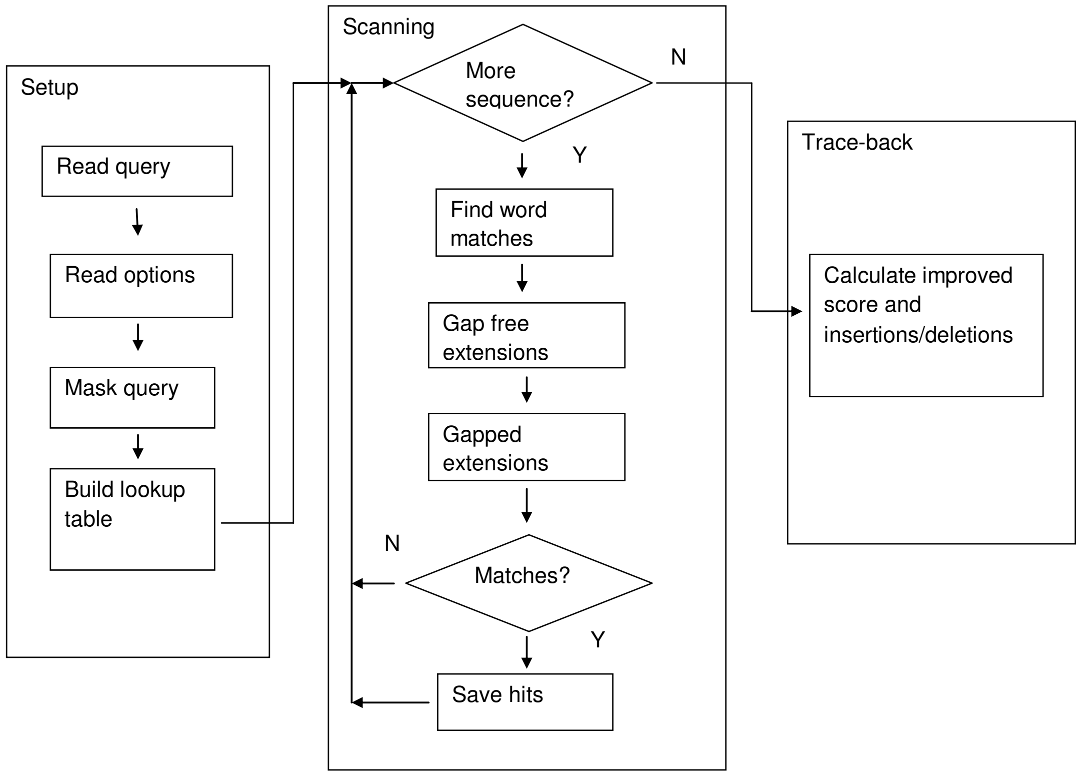

Figure 2.2: This is an original figure from paper [12]. It shows the general process flow of the BLAST service. Many heuristics, for example, the masking sequences during setup, hit identification (finding word matches), and hit extensions during scanning help to speed up the application run time.

The key steps of the BLAST algorithm corresponding to Figure 2.2 are the following:

1. **Setup(Hit generation)** Preparing the protein sequences is essentially the transformation of the query sequences to a lookup table for quicker k-mer word positions allocation. The end result is a hash table, say h, that will be

created for the query sequences. Each entry of k-mer, say u, contains the positions of u in the query sequences. And, the hash table of the protein query sequence contains not only the positions of us but also the positions of the *neighbors* of u. A neighbor may be any other k-mer having at least a pre-defined score with u when a specified scoring matrix is used, and u is a neighbor of itself. Additional care has been given to some special regions of the data sequences. For example, the letters of a low complexity region and interspersed repeats will be marked as X in the process of hard-marking. Or, if soft-marking is used, these regions will simply be skipped in the word matching step. The database sequences used by BLAST usually are marked by soft-marking. The marking can help to eliminate spurious results. Sequences marking is also a response to the edge-effect [16] which occurrs in sequences comparisons.

2. **Scanning(Hit Extension)** While scanning a k-mer, say v, from a sequence in a marked database, if the entry for v in h has a non-empty list of positions of v's neighbors, then each member of the list for v is a *hit*. For example, given a query sequence, say q, and a target sequence, say t, from a database, a hash table h_q has been built for 3-mer words. When scanning the database, a 3-mer, say AAA, has been encountered, then the list of the entry for AAA in h_q contains all the hits for this AAA from q. And, a hit can be further denoted as a coordinate of the positions in q and t, for example, $hit = (i, j)$. An ungapped extension, in both forward and backward directions starting at the hit location, will extend for a highest scored alignment. The alignment returned by the ungapped extension is called an *HSP* (Highest-Scoring Segment Pair). Based on some rigorous mathematical theorems in [6, 15, 17, 16, 18], the significance level of an HSP is estimated. And, the E-value, returned by the calculation of the

theorems, is used to examine the likelihood that an HSP with a score of at least S is a random alignment. Equation 2.2 and 2.1 are the part of the results from the theorems. Equation 2.2 shows the formulae of obtaining the E-value using normalized HSP score S'. The normalized score is shown in Equation 2.1 where m and n are the effective length of search sequences and the λ and K are constant.

$$S' = \frac{\lambda S - lnK}{\ln 2} \tag{2.1}$$

$$E = mn2^{-S'} \tag{2.2}$$

The gapped extension is similar to the ungapped extension, except the gapped X-drop dynamic algorithm [19] is more complicated.

3. **Trace-back** The same trace-back algorithm as in the Smith-Waterman algorithm will be used on the sequences to produce an optimal local alignment between two sequences which qualifies the E-value for gapped alignments. The ranked local alignments will be shown as the final result.

The length of k continuous letters of a matching word is used to identify a hit that is called a consecutive *seed*. A consecutive seed can be denoted by a string of 1s. And the length of the matching word is the length of the seed. For example, 111 is a length 3 of consecutive seed. In BLASTp, the default value of k for a seed in protein homology search is 3. And k has an optional value of 2.

Note that, for the gapped BLAST, the ungapped extension is triggered with a more restrictive condition, namely the two-hit algorithm. The condition specifies that two non-overlap hits need to be on the same diagonal within a pre-defined distance. Any two hits are said to be on a same *diagonal* if the difference of their starting positions in the query sequence is the same as the difference of their starting positions in the target sequence. For example, there are $hit_1 = (i_1, j_1)$ and $hit_2 = (i_2, j_2)$

between sequence q and t. The diagonal of hit_1 is $d_1 = (i_1 - j_1)$. The diagonal of hit_2 is $d_2 = (i_2 - j_2)$. The hit_1 and hit_2 are at the same diagonal if and only if $d_1 = d_2$.

Another note is that the E-value is not a probability. The E-value, as shown in Equation 2.2, means that the expected number of an HSP having a score of at least S. It had been proven [16] that the number of random HSPs with a score of at least S follows by a Poisson distribution. The mean of this Poisson distribution is the E-value. This means that the actual probability of exactly α HSPs with a score of at least S is given in Equation 2.3.

$$P(\alpha) = e^{(-E)} \cdot \frac{E^{\alpha}}{\alpha!} \tag{2.3}$$

where E is the E-value of S appeared in the Equation 2.2.

$$E = mn2^{-(\frac{\lambda S - lnK}{\ln 2})} \tag{2.4}$$

The E-value can be rewritten for score S as in the Equation 2.4. And, when $\alpha = 0$ as in Equation 2.3, it means that the probability of zero number of HSPs with a score of at least S is $P(0) = e^{(-E)}$. Hence, the probability of at least one HSP with a score of at least S is $P(1) = 1 - e^{(-E)}$. This is the P-value regarding to this score S. Moreover, when E-value is < 0.01, an empirical study has shown that the values of P-value and E-value are roughly the same. The choice of using the E-value over the P-value is because E-value is easier to understand. For example, it is easier to see the difference between the E-value of 1 and 10 than the P-values of 0.99 and 0.99995. The choice of the value of E-value controls the number of HSPs that is returned in BLAST. A larger E-value returns more HSPs by BLAST.

Last note is that the statistical results related to the random alignments has been proven for the ungapped alignment only. As for the gapped alignment, no theories have been proven. However, empirical studies have shown that the ungapped theories are still a good estimation for the gapped alignment.

There are different BLAST-like programs which are not owned by NCBI. For example, the CS-BLAST [20] is derived based on six consecutive amino acids when finding a hit. The WU-BLAST [21] by Washington University is very much a resemblance of BLAST. Besides BLAST-like programs, a wide variety of similar tools were also introduced [22, 23] as the complimentary tools. Other tools, as a by-product of BLAST, are also available, for example BLAT [24].

2.3.3 PatternHunter

When a pattern of consecutive letters is used as a seed for finding a hit, a natural question to ask is whether a more complicated pattern may also be a seed, i.e. non-consecutive letters which are separated by spaces. The spaced seed from PatternHunter has shown greater sensitivity and at least the same speed as BLAST.

A *spaced seed*, as an advanced innovation of the consecutive seed, can be defined as a binary string with two parameters: the length of a seed, say k, and the weight of the seed as the number of 1s in the binary string, say w. For example, a spaced seed of 1101, is a weight 3 and length 4 binary string, and often the positions of 1s are called cared positions because of required matches on it, and 0s are the “don’t care” positions because the match is not compulsory on it. For the purpose of a similarity search, the first and last positions of a spaced seed are required to be 1. And, a consecutive seed of 111 may be considered as a spaced seed with weight 3 and length 3.

The spaced seed was born in the release of PatternHunter software in 2002 [7]. These include the enhancing of the original spaced seed method, for example, [25, 26, 27, 28, 29, 30]; the developments on the optimization of spaced seeds and modified spaced seeds which led to the conclusion of optimization is a NP-hard problem, for example, [31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43]; and applications that had adapted spaced seeds, for example, [44, 45, 46, 47, 12]. As of today, a fairly good mathematical understanding which was previously unknown has been obtained. It includes added combinatorial and probabilistic studies and detailed analysis on new algorithms and complexity researches.

Investigation on using *multiple spaced seeds* [26] allows several spaced seeds to be present simultaneously which has shown more efficient trade-off between sensitivity and speed. The *vector seeds* [25], can be considered a type of multiple spaced seeds, that set less restrictive conditions on the w cared positions in the spaced seed, rather than an exact match on the cared position. The pair with a higher than pre-defined score is used to decide whether the cared position can be considered as a match. The neighbour seeds [30], is another type of multiple spaced seeds, allows to seed a hit not in the look-up table. It allows i out of w cared positions to be mismatches. At the same time, beside the w cared positions, it brings in additional j positions which can be considered as matched. All different types of multiple spaced seeds share a common overhead. They all require changes to the lookup table or the hits generation algorithm. More than one lookup table is needed because essentially each spaced seed needs its own table. This leads to an increase of the space complexity. Multiple queries on a single position of a k-mer in a query sequence is required because the w positions are considered as a w-dimension vector in the vector seeds. The changes on both hits generation and the following process of identified hits are needed for the neighbor seeds. In neighbour seeds, the values of i and j are 1 or 2 in order to minimize such impacts.

The advantage of the spaced seed over the consecutive seed is because the spaced seed has low internal periodicity. Hence, the hits created by the spaced seed are more independent. Given the same level of selectivity for both the spaced seed and consecutive seed implies that the total number of hits produced by the two kinds of seeds are roughly the same. More dependent hits tend to repeat on the HSPs that already have been hit. Then the hits will distribute over less numbers of different HSPs, and the numbers of HSPs that can be hit is reduced, so the level of sensitivity is lowered. Conside the event of at least one hit occurs at a random alignment. Let X be a random variable for the total number of hits at a random alignment. Then the expected number of hits of a random alignment is $E[X]$. Given that random alignment has been hit once, the expected number of hits to this random alignment is $E[X|X \geq 1]$. The value of $E[X|X \geq 1]$ for the spaced seeds are smaller than the consecutive seeds.

Consider the case of a simple DNA alignment, given a consecutive seed 1111. If it can hit at position i, then the chance of it hitting at a position of $i + 1$ is very big because the first 3 positions have been hit by the previous one. As for a spaced seed, say seed 10101001, if it can hit at a position of j of the same random alignment, when the position shifts from position of j to $j+1$, the probability of this spaced seed hitting at a position $j + 1$ is much smaller because none of positions from previous hit can be shared. If the chance of a nucleotide aligns to another nucleotide is independent, say $p = 0.25$, then the consecutive seed 1111 hits at i is $p^4 = 0.0039$, and the chance of 1111 hits at $i + 1$ would be $p = 0.25$. Then, for a spaced seed 10101001, it hits at a position of j with a probability of 0.0039, and the chance of it hits at a position of $j + 1$ is $p^4 = 0.0039$. Therefore, when considering the normal scenario of one hit as required for identifying an alignment, the value of $E[X|X \geq 1]$ for a space seed is the expected number of hits at an alignment after it has been hit; then $E[X|X \geq 1]$ is less than the value for a consecutive seed.

The idea of spaced seed is simple, but the problem of finding an optimal spaced seed is far more complicated. For example, given fixed values on the identity level and length of an ungapped alignment, the probability of a spaced seed hitting an HSP is

NP-hard [31]. Calculating the probability of a spaced seed to hit the uniform Bernoulli model is NP-hard [32]. Finding a single optimal spaced seed , or, a set of optimal multiple spaced seeds, is NP-hard for Bernoulli model [26]. But, there are various algorithms for constructing spaced seeds with substantial improvement on sensitivity in exponential time in seed length, for example, [34, 48, 41, 33]. Many heuristics have been developed to compute multiple spaced seeds with high sensitivity, for example, [49, 50, 51]. Moreover, the specifically designed spaced seeds for the protein coding regions can achieve the sensitivity of the Smith-Waterman algorithm while running at the similar speed of BLAST [44].

2.4 Scoring Matrices for Protein Homology Search

It is important to distinguish a related protein alignment from the random alignments because the random alignments are false positive for the final result and it affects the searching speed of a seeding based algorithm. The significance level of a related alignment against the random alignment is used to distinguish between the different alignments. And, a measurement on the fairness of such judgment can be enforced by a valid scoring matrix.

A *scoring matrix* may also be referred to as the *substitution matrix*. A simple example of an alignment between two DNA sequences using a unitary matrix (match=1, mismatch=-1) is shown in Table 2.6.

	a	a	g	t	t	t	c	t	t	g
	a	a	a	c	t	c	c	c	t	g
Individual scores:	1	1	-1	-1	1	-1	1	-1	1	1

Table 2.6: An example of simple scoring matrix.

The accumulative score using the unitary matrix for above alignment is $(6-4) = 2$. A more realistic substitution matrix with a refined substitution scenario is such that transition $(-\frac{1}{2})$ and transversion (-1) as a 4×4 scoring matrix as shown in Equation 2.5.

$$S = \begin{pmatrix} S_{a,a} & S_{a,c} & S_{a,g} & S_{a,t} \\ S_{c,a} & S_{c,c} & S_{c,g} & S_{c,t} \\ S_{g,a} & S_{g,c} & S_{g,g} & S_{g,t} \\ S_{t,a} & S_{t,c} & S_{t,g} & S_{t,t} \end{pmatrix} = \begin{pmatrix} 1 & -1 & -\frac{1}{2} & -1 \\ 1 & 1 & -1 & -\frac{1}{2} \\ -\frac{1}{2} & -1 & 1 & -1 \\ -1 & -\frac{1}{2} & -1 & 1 \end{pmatrix} \tag{2.5}$$

The new accumulative score of the alignment in Table 2.6 becomes $(6-2) = 4$.

A valid scoring matrix for the protein sequences comparison usually satisfies at least the following three criteria:

- Scores for identical amino acids are higher than the substitutions;
- Substitution scores of the conservatives are higher than the non-conservatives;
- When comparing different homologies, different scoring matrices are available for potential closed and divergent comparisons.

Over the past three decades a wide range of scoring matrices have been introduced that are based on different rationals. Among them, two scoring matrices, namely the PAM matrices [52] and BLOSUM matrices [53], are the most commonly used in the protein homology search. The BLOSUM62, one matrix in BLOSUM matrices family, is the *de facto* standard matrix used in many applications, e.g., both BLAST and FASTA use it.

In this Section, the basic mathematical models and fundamental algorithms used in constructing the PAM and BLOSUM matrices are reviewed. The detailed reviews of the PAM matrix is in Section 2.4.1. The BLOSUM matrix is in Section 2.4.2.

2.4.1 The PAM Matrix

The PAM scoring matrix was invented by Dayhoff and others in 1978 [52], and it was the first scoring matrix built on a protein evolutionary model. The PAM scoring matrix relies on three mathematical models: the Markov Chain Model to enforce the independence of mutation of each amino acid at a site of an alignment; the Phylogenetic Trees helps to count the mutations; and the log-odds ratio helps to convert a PAM probability matrix to a scoring matrix.

PAM, **P**oint **A**ccepted **M**utation, is defined as the replacement process of an amino acid by another amino acid under the natural selection. When a mutation is bounded by natural selection, it implies that the mutation will not change the natural functions of the protein. PAM scoring matrices are a series of matrices extrapolated on the multiplication of 1-PAM probability matrix. And, each PAM matrix implies different divergent alignments.

A PAM unit refers to, over a time period, 1% of amino acids change themselves by accepted mutations in a protein sequence. However, two protein sequences that are 100 PAM units apart does not imply 100% difference. As in Figure 2.3, we note that 100 PAM units show about 52% different positions in actual observation. Two main reasons are the cause. First, the mutations are likely to happen at a single site repeatedly. Second, the mutations could change back to its original amino acid. Note that there is no general correspondence between PAM distance and evolutionary time, as different protein families evolve at different rates.

By fitting each mutation into the Markov Chain Model, shown in Figure 2.4, it helps to simplify the analysis of the mutation because it assumes that each mutation

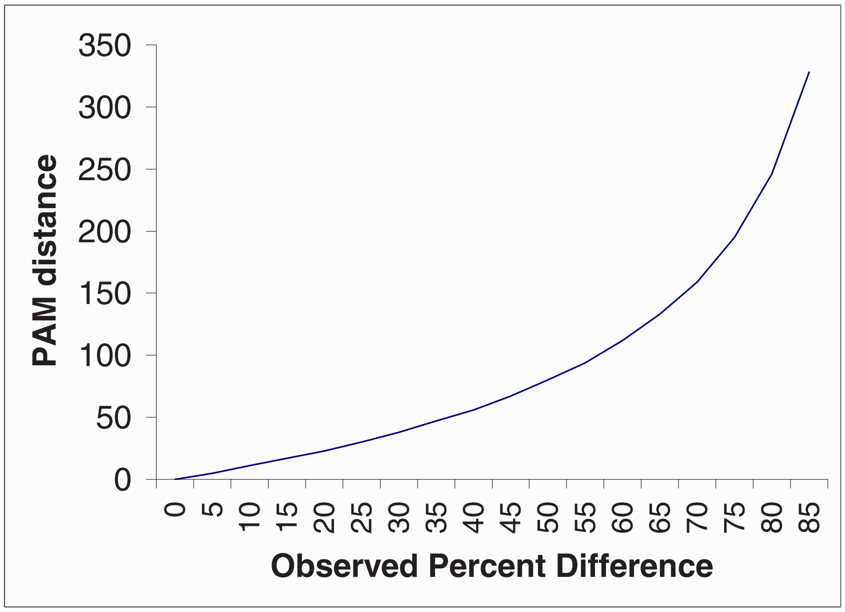

Figure 2.3: Y-axis is the PAM units (PAM distance) while X-axis is the actual differences in percentage. This the original drawing from [54].

happens independently at same time at a different site.

Basic steps involved for creating a 1-PAM mutation probability matrix are as following:

1. Select training data such that each training group contains sequences no more than 15% divergent in terms of pair-wise sequences identities level.

2. For each group of training data, build its phylogenetic trees.

3. Collect the occurrence of each mutation along a tree branch in each tree.

4. Compute relative mutability and mutation probability for the mutation probability matrix.

The Relative Mutation, m_i, for amino acid i is defined as how likely i changes to a different amino acid over a time period that is shown in Equation 2.6.

$$m_i = \frac{\text{changes}}{\text{exposure to mutation}} = \frac{f_i}{q_i \cdot \sum_i f_i} \tag{2.6}$$

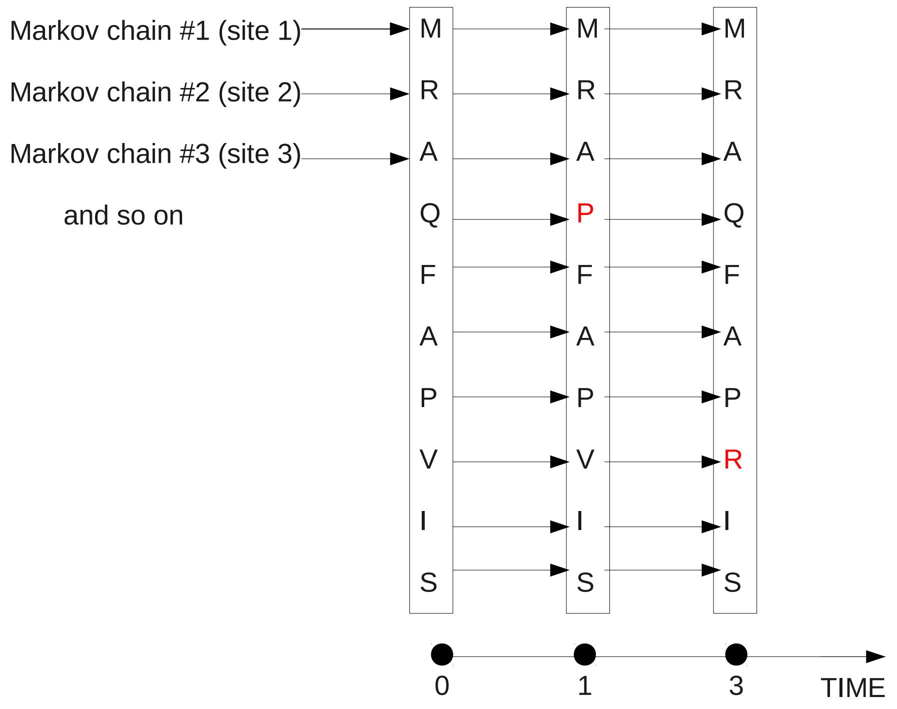

Figure 2.4: Each amino acid evolves according to one Markov Chain and is independent from the others through time.

Where q_i is the frequency of occurrence of the amino acid i, f_i is the number of times i is involved in a mutation, and $\sum_i f_i$ is the total number of changes for all amino acids (2 times the number of mutations).

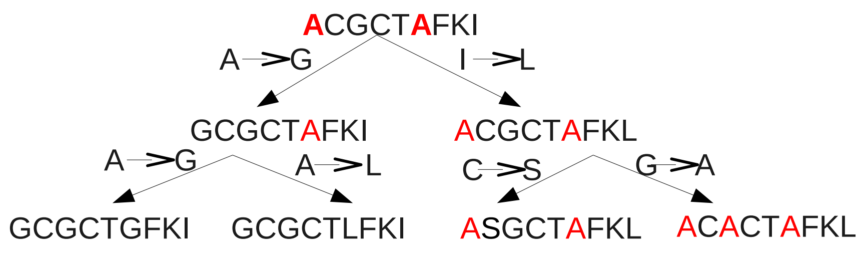

Figure 2.5: Example (from [55]) of a phylogenetic tree on 7 protein sequences in a multiple alignment. The internal nodes are the inferred ancestors.

Once the relative mutation is known, M_{ij} can be derived from Equation 2.7. M_{ij} is the probability that the amino acid j will be replaced by the amino acid i after a given evolutionary internal.

$$M_{ij} = \frac{\lambda m_j A_{ij}}{\sum_i A_{ij}} \tag{2.7}$$

where A_{ij} is the number of accepted point mutations between amino acids i and j. Equation 2.8 shows that the probability of amino acid j changes to amino acid j after a given evolutionary interval.

$$M_{jj} = 1 - \lambda m_j \tag{2.8}$$

Using the phylogenetic tree in Figure 2.5 as an example, we could calculate the value relative mutability of amino acid A, $m_A = 2.1$. Because $q_A = \frac{10}{63} = 0.15873$, the changes involve A as $f_A = 4$, the total changes for all amino acids is $6 \cdot 2 = 12$. The exposure to mutations of A is $12 \cdot 0.15873 = 1.9047$ Then the relative mutability $m_A = \frac{4}{1.9047} = 2.1$. As $A_{AG} = 3$, $\sum_j A_{Aj} = 4$, then $M_{AG} = 0.021 \cdot \frac{3}{4} = 0.01575$ when $\lambda = \frac{1}{100}$. Figure 2.6 shows the 1-PAM mutation probability matrix.

Because the expected proportion of the amino acids that mutates after one PAM unit is 1%, we can estimate the value of λ in Equation 2.9 and 2.10. Let $q_i \sum_{j \neq i} M_{ij}$ be the probability at one site as $i \rightarrow j$. The expected proportion of mutations of 1%, when considering all the positions, that is $\lambda \sum_i q_i \sum_{j \neq i} M_{ij}$, shown in Equation 2.9.

$$0.01 = \lambda \sum_i q_i \sum_{j \neq i} \frac{m_i A_{ij}}{\sum_j A_{ij}} \tag{2.9}$$

It is therefore that λ is given as in Equation 2.10.

$$\lambda = \frac{0.01}{\sum_i q_i \sum_{j \neq i} \frac{m_i A_{ij}}{\sum_j A_{ij}}} \tag{2.10}$$

The 1-PAM scoring matrix is converted from the 1-PAM mutation probability

A	9876	2	9	10	3	8	17	21	2	6	4	2	6	2	22	35	32	0	2	18
R	1	9913	1	0	1	10	0	0	10	3	1	19	4	1	4	6	1	8	0	1
N	4	1	9822	36	0	4	6	6	21	3	1	13	0	1	2	20	9	1	4	1
D	6	0	42	9859	0	6	53	6	4	1	0	3	0	0	1	5	3	0	0	1
C	1	1	0	0	9973	0	0	0	1	1	0	0	0	0	1	5	1	0	3	2
Q	3	9	4	5	0	9876	27	1	23	1	3	6	4	0	6	2	2	0	0	1
E	10	0	7	56	0	35	9865	4	2	3	1	4	1	0	3	4	2	0	1	2
G	21	1	12	11	1	3	7	9935	1	0	1	2	1	1	3	21	3	0	0	5
H	1	8	18	3	1	20	1	0	9912	0	1	1	0	2	3	1	1	1	4	1
I	2	2	3	1	2	1	2	0	0	9872	9	2	12	7	0	1	7	0	1	33
L	3	1	3	0	0	6	1	1	4	22	9947	2	45	13	3	1	3	4	2	15
K	2	37	25	6	0	12	7	2	2	4	1	9926	20	0	3	8	11	0	1	1
M	1	1	0	0	0	2	0	0	0	5	8	4	9874	1	0	1	2	0	0	4
F	1	1	1	0	0	0	0	1	2	8	6	0	4	9946	0	2	1	3	28	0
P	13	5	2	1	1	8	3	2	5	1	2	2	1	1	9926	12	4	0	0	2
S	28	11	34	7	11	4	6	16	2	2	1	7	4	3	17	9840	38	5	2	2
T	22	2	13	4	1	3	2	2	1	11	2	8	6	1	5	32	9871	0	2	9
W	0	2	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	9976	1	0
Y	1	0	3	0	3	0	1	0	4	1	1	0	0	21	0	1	1	2	9945	1
V	13	2	1	1	3	2	2	3	3	57	11	1	17	1	3	2	10	0	2	9901
	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V

Figure 2.6: Reproduction of the original 1-PAM mutation probability matrix from [52]. Note that the matrix is not symmetric. For example, the change between amino acids A and R, $A \sim R$ is not equal to $R \sim A$. According to [52], the value of M_{ij} defines the probability of amino acid in column j will be replaced by amino acid in row i after one accepted point mutation per 100 amino acids. The values are scaled by multiplying 10000 in this table.

matrix using the log-odds ratio. The process of converting to a scoring matrix is explained briefly here. Assuming we have two sequences S and S' that are given as the following:

$$S\ : a_1 a_2 \ldots a_n$$
$$S' : b_1 b_2 \ldots b_n$$

When two sequences S and S' are randomly generated and 1 PAM apart, let q_i be the background probability of i. Then we have two hypotheses of H_0 and H_A, whereas H_0: when S and S' are not related; and H_A: when S and S' are related.

Under H_0, when the letter at each position is i.i.d., we have the probability of

P_{H_0} as the following:

$$P_{H_0}(\text{random alignment}) = (\prod_{i=1}^{n} q_{a_i}) \cdot (\prod_{i=1}^{n} q_{b_i}) = \prod_{i=1}^{n} (q_{a_i} \cdot q_{b_i})$$

Under H_A, the letters that are at the same positions that are dependent; we have the probability P_{H_A} as the following:

$$P_{H_A}(\text{related alignment}) = \prod_{i=1}^{n} (q_{a_i} \cdot p_{a_i b_i})$$

Hence, the significance level of a related alignment is usually presented as the ratio of probability of H_A and H_0 as a natural choice in Equation 2.11.

$$\frac{P_{H_A}(\text{related alignment})}{P_{H_0}(\text{random alignment})} = \frac{\prod_{i=1}^{n} (q_{a_i} \cdot p_{a_i b_i})}{\prod_{i=1}^{n} (q_{a_i} \cdot q_{b_i})} = \prod_{i=1}^{n} \frac{q_{a_i} \cdot p_{a_i b_i}}{q_{a_i} \cdot q_{b_i}} = \prod_{i=1}^{n} \frac{p_{a_i b_i}}{q_{b_i}} \tag{2.11}$$

By applying a logarithm, we change Equation 2.11 to Equation 2.12.

$$\log \frac{P_{H_A}(\text{related alignment})}{P_{H_0}(\text{random alignment})} = \log \prod_{i=1}^{n} \frac{p_{a_i b_i}}{q_{b_i}} = \sum_{i=1}^{n} \frac{p_{a_i b_i}}{q_{b_i}} \tag{2.12}$$

For any pair of amino acids a and b, the score between them is defined as in Equation 2.13 and the alignment score between S and S' is defined as in Equation 2.14.

$$S_{a,b} = \log \frac{p_{ab}}{q_b} \tag{2.13}$$

When a aligns to b in a position of an alignment, P_{ab} is the same as mutation probability M_{ab}, and q_b is the probability of b as appears in the second sequence of the alignment. Because we apply the logarithm, we could sum up the score of each single position of the alignment to obtain the alignment score as in Equation 2.14.

$$S(\text{alignment}) = \sum_{i=1}^{n} S_{a_i b_i} \tag{2.14}$$

The 250-PAM mutation matrix is a result of multiplying 1-PAM mutation matrix 250 times, $M_{250} = M_1^{(250)}$, because of the m-steps transitions property in Markov Chain Model.

The term of the relatedness odds between i and j is given in Equation 2.15. The relatedness odds implies the relative odds of evolution rather than chance. Where f_i is the probability that i occurs in the second sequence of an alignment by chance.

$$R_{ij} = \frac{M_{ij}}{f_i} \tag{2.15}$$

Then the substitution score between i and j can be derived from Equation 2.15, shown in Equation 2.16. The value k is a scaling factor to clear the fractional values. And $k = 10$ is used for transforming the 250-PAM probability matrix to 250-PAM scoring matrix, shown in Figure 2.7.

$$S_{ij} = k \frac{\log R_{ij} + \log R_{ji}}{2} \tag{2.16}$$

An updated PAM matrices with extensive training data had been obtained in [56, 57].

2.4.2 The BLOSUM Matrix

BLOSUM (**BLO**cks **SU**bstition **M**atrix) [53] matrices were introduced in 1992. It is the amino acid substitution matrix derived from the direct estimation of statistics of amino acids pair from a BLOCKs database. BLOSUM62, one of the BLOSUM matrices, is the standard matrix used in BLAST. In this section, we will briefly review

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	2																			
R	-2	6																		
N	0	0	2																	
D	0	-1	2	4																
C	-2	-4	-4	-5	12															
Q	0	1	1	2	-5	4														
E	0	-1	1	3	-5	2	4													
G	1	-3	0	1	-3	-1	0	5												
H	-1	2	2	1	-3	3	1	-2	6											
I	-1	-2	-2	-2	-2	-2	-2	-3	-2	5										
L	-2	-3	-3	-4	-6	-2	-3	-4	-2	2	6									
K	-1	3	1	0	-5	1	0	-2	0	-2	-3	5								
M	-1	0	-2	-3	-5	-1	-2	-3	-2	2	4	0	6							
F	-4	-4	-4	-6	-4	-5	-5	-5	-2	1	2	-5	0	9						
P	1	0	-1	-1	-3	0	-1	-1	0	-2	-3	-1	-2	-5	6					
S	1	0	1	0	0	-1	0	1	-1	-1	-3	0	-2	-3	1	2				
T	1	-1	0	0	-2	-1	0	0	-1	0	-2	0	-1	-3	0	1	3			
W	-6	2	-4	-7	-8	-5	-7	-7	-3	-5	-2	-3	-4	0	-6	-2	-5	17		
Y	-3	-4	-2	-4	0	-4	-4	-5	0	-1	-1	-4	-2	7	-5	-3	-3	0	10	
V	0	-2	-2	-2	-2	-2	-2	-1	-2	4	2	-2	2	-1	-1	-1	0	-6	-2	4

🟦	Hydrophobic
🟩	Aromatic
🟨	Positive

Figure 2.7: 250-PAM scoring matrix is a reproduction from [52].

the creation of the BLOCKs database, which is used as the training data for creating BLOSUM matrices. Then, we will recap the details of creating a BLOSUM matrix.

Creation of a BLOCKs database is much involved as it needs to run in many enumerated steps with different software as depicted in paper [58]. Here, we only point out the general idea of making a BLOCKs database.

A BLOCKs database contains multiple entries of blocks. A *block* is an ungapped local multiple alignment of amino acid sequences from a group of related proteins. Given a group of protein sequences, a set of blocks can be created in two steps for them using the PROTOMAT [58] program. The first step finds candidate alignments. The second step extends the alignments and returns the best set of blocks. In the first step, the MOTOIF [59] program is called to find the candidate alignments. The MOTOMAT program is called in the second step to extend the alignments and assemble the best set of blocks.

In the process of creating a BLOCKs database, a scoring matrix is needed for both MOTOIF and MOTOMAT programs. An iteration of refining on a scoring matrix is employed. The first scoring matrix is a unitary matrix. Given a set of proteins S, a unitary matrix (1=match, 0=mismatch) was used to build a database of 2205 BLOCKs. Then each of the blocks is clustered at 60% identities level. Next on the clustered blocks the amino acid's frequency is counted and a scoring matrix of SM_1 is obtained; next SM_1 on S is used to re-build a database of 1961 BLOCKs; then each of the blocks are clustered at 60% identities level. Next on the clustered blocks the amino acid's frequency is counted and a scoring matrix of SM_2 is obtained. Finally, the scoring matrix SM_2 on the actual training data T is used to generate a training BLOCKs database. The training BLOCKs database is clustered at 60% identities level for each individual block. And, the amino acids frequencies on this training database will be used to generate the actual BLOSUM matrix.

The clustering inside each block at 60% identities level ensures that there are no two sequences with more than 60% identities level in a different cluster. The detailed procedure of clustering the sequences in a block is briefly explained here. Upon obtaining the training BLOCKs database, inside each block, the sequences will be clustered and weighed. This reduces the multiple contributions of amino acid pairs from the most closely related sequences inside a block. If the sequences were not clustered inside a block, then the computed scoring matrices will tend to be in favour of finding highly similar homologies. These highly similar homologies can be found easily. And, it is the distanced homologies that the biologist is most interested in. Hence, the clustering and weighting sequences adjust the fairness among the contributions of different amino acid pairs. At the end of the clustering at a level X, inside of each block, the sequences with similar level greater than or equal to X% will form a connected graph.

The most commonly used BLOSUM62 matrix implies that there are no two clusters with more than 62% similarity inside of any block.

CKGC
CAGC
CVGC
CAGC
CGGC
CGGC

Table 2.7: shows an actual entry of LIM domain protein with block id of BL00478A in Version 5.0 of BLOCKS database, where CGC is a motif.

Consider the twenty commonly used amino acids, let C_{ij} be the total number of amino acid (i, j) pairs $(1 \leq j \leq i \leq 20)$ for an entry in the frequency table. The values of C_{ij} of the block in Table 2.7 is shown in Table 2.8.

C_{ij}	A	C	G	K	V
A	0+1+0+0				
C	0	15+0+0+15			
G	0+4+0+0	0	0+1+15+0		
K	0+2+0+0	0	0+2+0+0	0	
V	0+2+0+0	0	0+2+0+0	0+1+0+0	0

Table 2.8: The frequency of each amino acid pairs. For example, pair (A, A) has value of $(0 + 1 + 0 + 0)$ implies the frequencies value of $C_{AA} = 1$ because it occurs 0 time in the first column, 1 time in the second column, 0 time in the third column, and 0 time in the last column.

Let w be the number of columns, and n be the number of sequences in a block; then the normalized frequency T is given in Equation 2.17.

$$T = \sum_{i \geq j} C_{ij} = w \frac{n(n-1)}{2} \tag{2.17}$$

And the T value of the sequences in Table 2.7 is $4 \left[\frac{(6)(5)}{2}\right] = 60$. Let q_{ij} be the observed (or target) probability of occurrence of each pair (i, j), then it can be defined in Equation 2.18.

$$q_{ij} = \frac{C_{ij}}{T} \tag{2.18}$$

q_{ij}	A	C	G	K	V
A	1/60				
C	0	30/60			
G	4/60	0	16/60		
K	2/60	0	2/60	0	
V	2/60	0	2/60	1/60	0

Table 2.9: The values of q_{ij} of sequences in Table 2.7.

Let p_i be the probability of occurrence of the ith amino acid in all pairs, shown in Equation 2.19.

$$p_i = q_{ii} + \sum_{i \neq j} \frac{q_{ij}}{2} \tag{2.19}$$

P_A	P_C	P_G	P_K	P_V
$\frac{1+\frac{(4+2+2)}{2}}{60} = \frac{5}{60}$	$\frac{30}{60}$	$\frac{16+\frac{4+2+2}{2}}{60} = \frac{20}{60}$	$\frac{0+\frac{2+2+1}{2}}{60} = \frac{2.5}{60}$	$\frac{0+\frac{2+2+1}{2}}{60} = \frac{2.5}{60}$

Table 2.10: The values of p_i of sequences in Table 2.7.

Let e_i be the expected (or background) probability of occurrence of the ith amino acid in an (i, j) pair, shown in Equation 2.20.

$$e_{ij} = \left\{ \begin{array}{ll} p_i^2 & \text{if } i = j \\ 2p_ip_j & \text{if } i \neq j \end{array} \right\} \tag{2.20}$$

The derived score of an amino acid pair s_{ij} is a result of log-odds (logarithm of

e_{ij}	A	C	G	K	V
A	$(5/60)^2$				
C	$2(5/60)(30/60)$	$(30/60)^2$			
G	$2(5/60)(20/60)$	$2(30/60)(20/60)$	$(20/60)^2$		
K	$2(5/60)(2.5/60)$	$2(30/60)(2.5/60)$	$2(20/60)(2.5/60)$	$(2.5/60)^2$	
V	$2(5/60)(2.5/60)$	$2(30/60)(2.5/60)$	$2(20/60)(2.5/60)$	$2(2.5/60)(2.5/60)$	$(2.5/60)^2$

Table 2.11: The values of e_{ij} of sequences in Table 2.7.

odds) ratio from p_{ij} and e_{ij}, shown in Equation 2.21.

$$s_{ij} = \lambda \log \frac{p_{ij}}{e_{ij}} \tag{2.21}$$

The λ in Equation 2.21 is a scaling factor for rounding fractional numbers. The scoring matrix derived from the block in the Table 2.7 that is shown in Table 2.12

s_{ij}	A	C	G	K	V
A	3				
C	-	2			
G	1	-	3		
K	7	-	1	-	
V	7	-	1	5	-

Table 2.12: The values of s_{ij} are scaled by 2, then rounded into the nearest integer value. The symbol of '-' indicates such pairs with undefined values resulting from unobserved pairs; however such is not common in a large set of real data.

After using the BLOSUM matrices for nearly two decades, to the surprise of many, due to a programming error in its source code, the BLOSUM matrices are actually incorrectly calculated. And in the paper [60], Styczynski and others pointed out such a programming error. They computed a new set of matrices from the bug-fixed program. But the 'fixed' BLOSUM matrices could not perform better than the 'incorrect' matrices. The authors tried to exploit the cause of such phenomena. In the end they concluded it was *"nothing more than a rare set of circumstances caused by miscalculated normalizations, low-resolution scaling, and some fortuitous rounding"*. There is no logical explanation found.

2.4.3 Discussion

A valid scoring matrix should be in the implicit form of the log-odds ratio as the theory in [17] suggests. The theory stated that, when a scoring matrix SM is used for finding the optimal ungapped local alignments from the comparison of random sequences, under the condition of at least one of the scores between i and j, namely SM_{ij}, is positive, and the expected score of SM follows $\sum_{i,j} q_i q_j S_{ij} < 0$, the amino acids i and j align with frequency p_{ij} as in Equation 2.23.

$$p_{ij} = q_i q_j e^{\lambda S_{ij}} \tag{2.22}$$

The theory also provides an Equation, shown in 2.23, to calculate the value of λ.

$$1 = \sum_{ij} q_i q_j e^{\lambda S_{ij}} \tag{2.23}$$

Therefore, the score between i and j can be found as in Equation 2.24. This implies that the general form of a valid scoring matrix should be a log-odds ratio.

$$S_{ij} \approx \frac{1}{\lambda} \cdot \ln\left(\frac{p_{ij}}{q_i q_j}\right) \tag{2.24}$$

A natural question to ask is, when using a scoring matrix, how to judge their alignment as significant from the random alignments. The *Relative Entropy* of a scoring matrix, based on rigours theorems [61], can be used to answer this question. We only briefly explain the basic idea of creating the relative entropy. Another theory [18] states that the expected number of the MSPs (Maximal- scoring Segment Pairs) with a score of at least S is given in Equation 2.25.

$$N = K n_1 n_2 e^{-\lambda S} \tag{2.25}$$

Where an MSP is the highest-scoring local ungapped alignment between two sequences, and setting λ to $\ln 2$, the Equation 2.25 can be rewritten for score s as in Equation 2.26.

$$S = \log_2 \frac{K}{N} + \log_2 (n_1 n_2) \tag{2.26}$$

In [18], they found if an alignment is considered as significant, the empirical values of K are usually less than 1 for a typical scoring matrix, and the value of N is less than 0.01. It is therefore that the term $\log_2 (n_1 n_2)$ in Equation 2.26 becomes dominated. For example, if we have $n_1 = n_2 = 250$ for a two sequences comparison, then an MSP needs to have at least a score of 16 ($\approx 2 \log_2 250$) in order to be statistically significant. Or if we have $n_1 = 250$ and $n_2 = 10000000$ for a database search, then an MSP needs to have at least a score of 31 ($\approx \log_2 2500000000$) in order to be statistically significant.

When using a scoring matrix to calculate the score of an alignment, an expected score per residue pair with respect to the target frequency p_{ij} is defined as the Relative Entropy (H). The Equation 2.27 shows how to derive H from Equation 2.24 when $\lambda = \ln 2$. The Table 2.13 shows the different values of λ and H for different types of scoring matrices.

$$H = \sum_{i,j} p_{ij} s_{ij} = \sum_{i,j} p_{ij} \cdot \log_2 (\frac{p_{ij}}{q_i q_j}) \tag{2.27}$$

	BLOSUM45	BLOSUM62	BLOSUM80	PAM250	PAM120	PAM70	PAM30
λ	$\frac{\ln 10}{10}$	$\frac{\ln 2}{2}$	$\frac{\ln 10}{10}$	$\frac{\ln 10}{10}$	$\frac{\ln 2}{2}$	$\frac{\ln 2}{2}$	$\frac{\ln 2}{2}$
Relative Entropy	0.3795	0.6979	0.9868	0.354	0.979	1.6	2.57

Table 2.13: Note the changes of H on PAM and BLOSUM matrices are in the same order. The values of λ are derived from Equation 2.24.

H can be interpreted as the expected score per amino acid pair in an alignment with respect to the target frequencies. The value of H in different scoring matrices can be used to estimate the effective length of a statistically significant alignment. A bigger value of H can help to distinguish a shorter alignment from chance. If we use the PAM120 matrix, a statistically significant alignment from two sequences of equal length of 250 with score of 16-bit implies its effective length is 17 ($\approx \frac{16}{0.979}$). While using BLOSUM62 matrix, a statistically significant alignment between a sequence of a length of 250 and database length of 10000000 with a score of 31-bit implies its effective length is 44 ($\approx \frac{31}{0.6979}$).

The PAM and BLOSUM matrices were built based on each of its own assumptions. One of the assumptions is the mutations between two amino acids. It assumes that the mutations are happening at the same probability rate and are reversal in both directions. This assumption on mutation is because most protein sequences we have obtained are from extant species, yet we have not many protein sequences being proved as an ancestor of the others.

One problem of this assumption is ignoring the gaps have occurred in the protein sequences evolution process. Only correctly considering the insertions and deletions in the process, or at least in a reasonable large scale during the process, the evolution between proteins will truly be reflected. It is especially true for two sequences that are highly divergent.

The PAM matrix, as a by-product, explicitly introduced a protein evolution model for understanding the changes in proteins. The BLOSUM matrix was built on a much more broad range of data on protein sequences compared to the PAM matrix. The BLOSUM matrix is simple and independent because it involves only the log-odds form from the statistics. It does not need phylogenetic trees, the max parsimony, and the Markov Chain Model. Note that tests [62] have shown that the BLOSUM matrix is slightly better than the PAM matrix.

Usually, the BLOSUM62 is best suited for local alignments, and 250-PAM is best suited for global alignments. And when possible, 60-PAM, 120-PAM, and 250-PAM matrices are recommended to be used together to generate better outcomes.

Chapter 3

Spaced k-mer Neighbors

3.1 Introduction

Sequence similarity search is vitally important for the study of DNA and proteins in modern molecular biology and has been actively researched in bioinformatics. Due to the large size of the DNA and protein sequence data, efficiency is a top-priority in the development of similarity searching algorithms. A commonly used technique is the tradeoff between speed and sensitivity by using filtration. A filtration method quickly identifies the potential similarity regions between the query and the database sequences; then only these similarity candidates are further examined by a more accurate (and usually slower) algorithm.

Earlier development of filtration was exemplified by the BLAST program, which uses a consecutive k-mer match as a seed to filter out the potential similarities between two DNA sequences [6]. Since the finding of exact k-mer matches is relatively simple, this seeding idea greatly improved the search speed of BLAST. However, because not all DNA similarities have a long k-mer match, some similarities are lost due to this filtration, resulting in reduced sensitivity. The parameter k can be used as a tradeoff between speed and sensitivity.

For a better tradeoff, the PatternHunter [7] program first proposed the optimized spaced seed algorithm. A spaced seed is represented by a binary string such as 111*11*1, where the 1 means "required match" and the * means "don't care". The number of 1s in the seed is called the weight of the seed. Given a spaced seed, the algorithm will use the exact match at the required matching positions as the filtration criterion. To the surprise of many, the PatternHunter paper demonstrated that by optimizing the configuration of the positions of the 1 and * in the spaced seed, a weight-k spaced seed is significantly more sensitive than the weight-k consecutive seed used by BLAST. This spaced seed idea was later implemented in the BLAST program for the searching of distant similarities.

Since PatternHunter significant research has been carried out to find more efficient ways to tradeoff search sensitivity and speed. In particular, the PatternHunter II paper [26] studied multiple spaced seeds, where several spaced seeds with different shapes are used simultaneously to increase the sensitivity. The vector seed and multiple vector seeds methods [45, 25] modifies the spaced seed by allowing one or a few of the required positions to be mismatches. An earlier review of some of these developments can be found in [63]. Most of these developments in more efficient seeding are designed for DNA similarity searchs and regard a mismatch between two letters a negative evidence to the similarity. Recently, the spaced seed method has also been adopted for efficient reads mapping for next generation genome sequencing analysis [46].

While for DNA sequences, a mismatch between two bases is considered a negative evidence for the DNA homology, protein sequences are different. According to the BLOSUM [53] and PAM [52] matrices that measure amino acid similarities, the mismatch between a pair of amino acids can be scored positively. So, the seeding methods used for DNA similarity search need to be adjusted to work on proteins. The BLASTp program extended the consecutive seed idea by using the approximate match of a pair of k-mers that have a matching score greater than or equal to a given threshold as seeds. The k-mer pairs with higher-than-threshold matching scores are pre-calculated and indexed for efficient finding of those approximate matches. Particularly, the BLASTp program builds a DFA to search for the positions of hits at the scanning stage. The multiple vector seeds method [45] combined BLASTp method with multiple spaced seeds. Another approach for protein similarity search in the literature is to classify amino acids into several classes [64] or hierarchical classes [65] so that the spaced seeds designed for DNA sequences can work on amino acid classes.

The k-mer pairs selection by using the BLOSUM [53] score threshold is not optimal for protein homology search. The reasons for this sub-optimality are examined, and a new filtration method, namely spaced k-mer neighbors, is proposed.

3.2 Preliminaries and Notations

Given a pair of query and database sequences, a hit denotes a pair of positions, each from one of the two sequences. In a filtration method for a similarity search, a *hit* indicates a possible similarity around the two positions that deserves further examination. A *seeding scheme* defines the criterion that two positions of the query and database sequences generate a hit. A spaced seed x is denoted by a binary string over alphabet $\{1, *\}$, where 1 indicates the required matching positions and $*$ for "don't cares". Given a spaced seed $x = x_0 \dots x_{l-1}$ and a sequence s, the *spaced k-mer* at position i of s is defined as the string $t_0 \dots t_{l-1}$ such that

$$t_j = \begin{cases} s[i+j] & \text{if } x[j] = 1 \\ * & \text{if } x[j] = * \end{cases}$$

For example, if the spaced seed is 11*1 and sequence is AMKMKK, then the spaced k-mer at position 0 is AM*M and at position 1 is MK*K. Thus, in a spaced seed method, two positions of the query and database sequences generate a hit if their spaced k-mers at those positions are identical. A *similarity matrix* defines the similarity score between two letters. For example, the BLOSUM matrix defines the similarity between each pair of amino acids. Given a similarity matrix M, the score between two sequences s_1 and s_2 with equal length l, denoted by $f_M(s_1, s_2)$, is calculated by $\sum_{j=1}^{l} M(s_1[j], s_2[j])$. A high-scoring segment pair (HSP), is a pair of sequences such that the $f_M(s_1, s_2)$ have a score higher than a specified threshold. For the sake of presentation clarity, for any given scoring matrix M, we define $M[*, *] = 0$. The similarity score between two spaced k-mers t_1 and t_2, are denoted by $f_M(t_1, t_2)$, and computed as $\sum_{j=1}^{l} M(t_1[j], t_2[j])$.

For protein homology search, a sequence alignment is usually the concatenation of several HSPs. The general procedure for a homology search algorithm first utilizes a seeding method to detect hits. Then an extension procedure is used to check if there is an HSP around the hit. Only when the HSP is successfully detected, a Smith-Waterman algorithm is called to generate sequence alignment. The main optimization goal of the seeding method is then becoming the efficient detection of the HSPs.

The sensitivity of a seeding method is the portion of HSPs that generate at least one hit. The selectivity of a seeding scheme is the probability that two random positions of a query and database sequences is a hit. If two variables x and y are linearly correlated to each other, we denote $x \propto y$.

3.3 The Dependencies that May Affect Sensitivity

If two seeding schemes have the same selectivity, they normally produce a similar number of hits in the HSPs. The key reason that they may have very different sensitivity is the different distributions of the hits. Since each HSP requires only one hit for its detection, having more than one hit in an HSP is a waste. Consequently, if one distributes hits more evenly across different HSPs, it will detect more HSPs, or in other words, have a higher sensitivity.

This appeared to be the main reason that the spaced seed proposed in PatternHunter outperformed the consecutive seed used by BLAST. More specifically, consider that an HSP has a consecutive seed hit at position i. Thus the two k-mers $(a_i a_{i+1} \ldots a_{i+k-1})$ and $(b_i b_{i+1} \ldots b_{i+k-1})$ exactly match each other. Consider the two k-mers $(a_{i+1} \ldots a_{i+k-1} a_{i+k})$ and $(b_{i+1} \ldots b_{i+k-1} b_{i+k})$, they would have a high probability to match because this second hit requires only one additional letter match between a_{i+k} and b_{i+k}. Thus, when consecutive seeds are used, there is a tendency that once an HSP is hit, it is hit multiple times. However, for spaced seeds, this tendency is greatly reduced.

Note that the above argument still holds even if the seeding scheme only requires the two k-mers to match approximately. Specifically, when the BLOSUM matrix M is used for the protein sequences, BLASTp requires the matching score $f_M(a_i a_{i+1} \ldots a_{i+k-1}, b_i b_{i+1} \ldots b_{i+k-1})$ to be greater than a threshold for considering $(a_i a_{i+1} \ldots a_{i+k-1})$ and $(b_i b_{i+1} \ldots b_{i+k-1})$ as a hit. Even under this approximate match scenario, the hit at position i will still increase the hit probability at position $i + 1$. This is the first dependency one needs to get rid of in order to increase the sensitivity. Based on this observation, the first proposed change over the BLASTp's seeding scheme is the following:

Change 1: Replace the k-mer high-scoring matches with spaced k-mer. This is similar to what was proposed in the multiple vector seeds paper [45].

The second dependency we examine is the dependencies between amino acids within a k-mer. For an amino acid a, let $P(a)$ be the probability that a occurs in the query and database sequences. For two amino acids a and b, $P(a, b)$ is the probability that they occur at the same position in the two sequences of an HSP. According to [53], the BLOSUM matrix M is such that

$$M[a, b] \propto \log \frac{P(a, b)}{P(a)P(b)}$$

Thus, for two k-mers a and b, if the amino acids at different positions of a k-mer are independent to each other, then

$$f_M(A, B) \propto \log \prod_{i=1}^{k} \frac{P(a[i], b[i])}{P(a[i])P(b[i])} = \log \frac{P(A, B)}{P(A)P(B)}$$

Here $P(A, B)$ is the probability that the k-mer pair (A, B) occurs in the HSPs, and $P(A)P(B)$ is the probability that the k-mer pair occurs randomly at two random positions of the query and database sequences. By using the high-scoring k-mer matches as hits, BLASTp essentially requires that the hitting k-mer pairs to have a high ratio between its occurring frequency in HSPs and in random sequence positions. As a result, the seeding scheme can maximize both the sensitivity and the selectivity simultaneously.

However, the above argument required the independence between the amino acids at different positions of a k-mer, which may not hold in reality. To illustrate this, the following experiment is carried out and the result is shown.

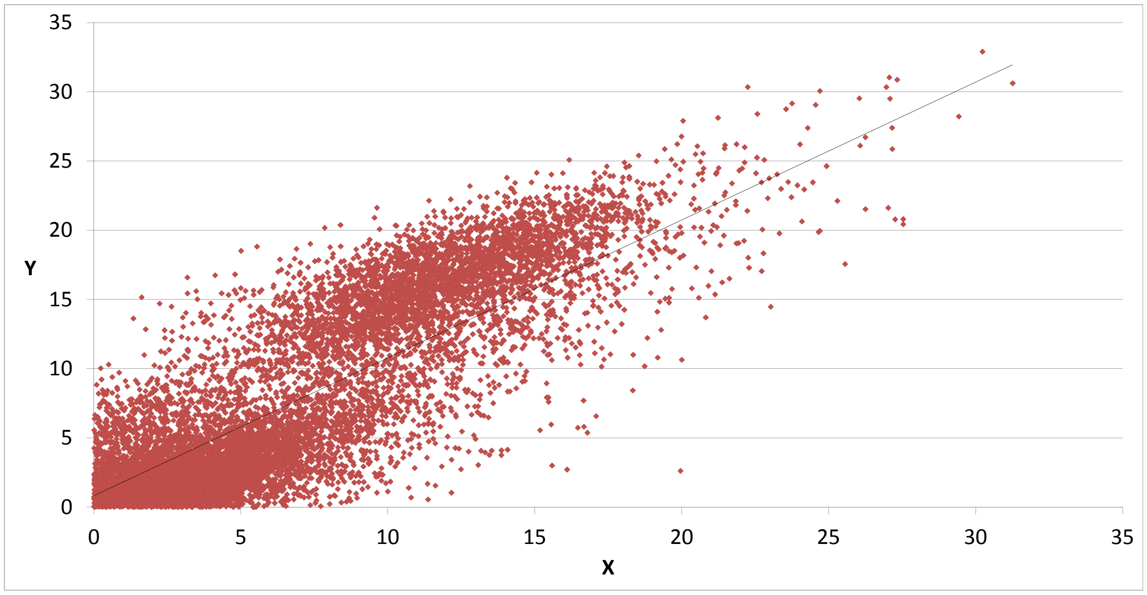

Figure 3.1: The two values of $2\log_2 \prod_{i=1}^{k} \frac{P(a[i],b[i])}{P(a[i])P(b[i])}$ (Y-axis) and $2\log_2 \frac{P(A,B)}{P(A)P(B)}$ (X-axis) may be different in reality.

On real HSPs between human and mouse proteins, for $k = 3$, Figure 3.1 shows the differences between $2\log_2 \prod_{i=1}^{k} \frac{P(a[i],b[i])}{P(a[i])P(b[i])}$ (Y-axis) and $2\log_2 \frac{P(A,B)}{P(A)P(B)}$ (X-axis), and Figure 3.2 shows the distribution of $2\log_2 \frac{P(A,B)}{P(A)P(B)}$ for 3-mer pairs with BLOSUM62 score greater than or equal to 11.

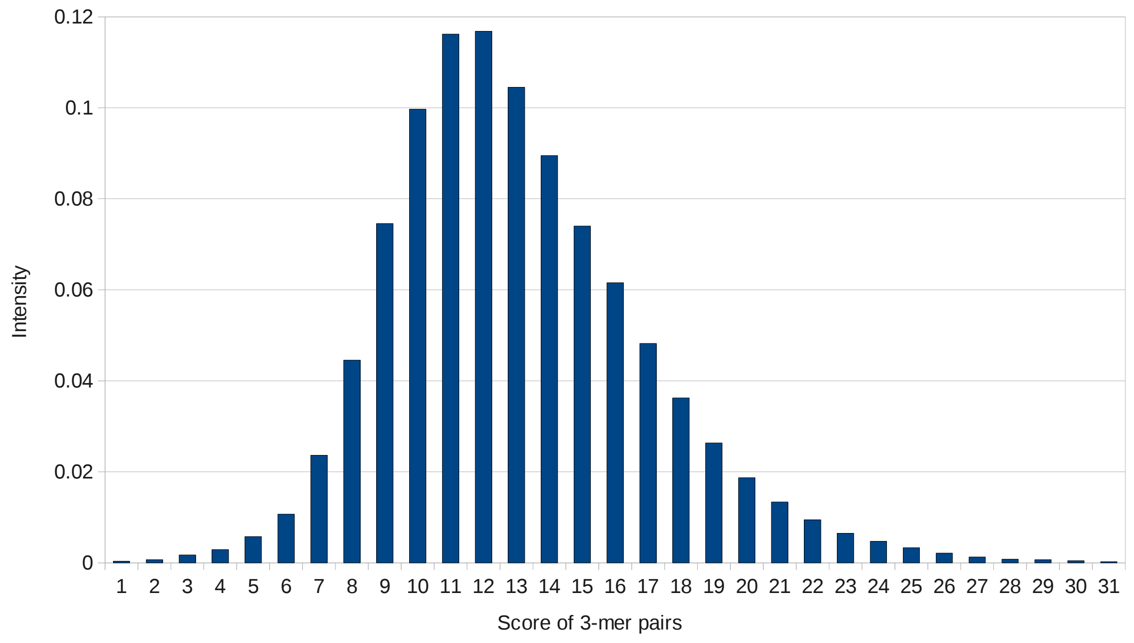

Figure 3.2: The distribution of values of $2\log_2 \frac{P(A,B)}{P(A)P(B)}$ for 3-mer pairs with BLOSUM62 score $>=$ 11 used in BLAST

Because of these discrepancies, the simple use of BLOSUM scores as selection criteria for k-mer pairs is not anymore optimal. Instead, the real $\frac{P(A,B)}{P(A)P(B)}$ should be used.

Change 2: Instead of using the BLOSUM score as hitting criterion, a set S of k-mer pairs are pre-selected. Based on the real $\frac{P(a,b)}{P(a)P(b)}$, the match of a pair of k-mers is a hit if and only if it belongs to S.

The choice of using a pre-selected set of k-mer pairs also enabled the removal of another type of dependencies between k-mer pairs. We illustrate the situation using consecutive seed first. Consider two k-mers pair $(a_1 \ldots a_k) \sim (b_1 \ldots b_k)$ and $(a_2 \ldots a_k\ x) \sim (b_2 \ldots b_k\ y)$. Each of these two pairs can detect their own set of HSPs.

However, the two detected HSP sets intersect each other and share all the HSPs that were detectable by the $(k+1)$-mer pair $(a_1 \dots a_k\ x) \sim (b_1 \dots b_k\ y)$. Thus, having one of the two k-mer pairs in S reduces the benefit of including the other.

This phenomena is illustrated in the following experiment. Consider the three 3-mer pairs:

`I`: $(a_1a_2a_3) \sim (b_1b_2b_3)$, `II`: $(a_2a_3x) \sim (b_2b_3y)$, and `II`$'$: $(xa_2a_3) \sim (yb_2b_3)$

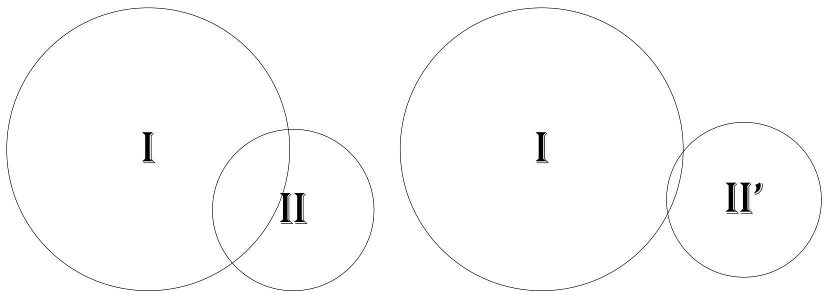

Figure 3.3: Comparison of `I`: (LSC)∼(LAC); `II`: (SCS)∼(ACA); `II`$'$: (SSC)∼(AAC). `I` and `II`$'$ share 3 HSP; `I` and `II` share 10,517 HSPs while `I` can hit 70,376 HSPs alone.

In Figure 3.3, the Venn Diagram of the HSPs hit by the three pairs indicate that `I` and `II` share a significant portion of HSPs.

Note that this dependency will be reduced by using spaced seed as two adjacent hits share a fewer number of positions. However, the further consideration of dependencies between (spaced) k-mer pairs will further reduce this dependency.

Change 3: The selection of k-mer pair set S should take into account of dependencies between different pairs.

3.4 Spaced k-mer Neighbors Method

In this section we propose the spaced k-mer neighbors method for increasing the similarity search sensitivity. A simple algorithm to optimize the k-mer pairs is also proposed.

3.4.1 Hit Generation

Algorithm 1 HitGeneration

```
Input: A set $\mathcal{S}$ of $k$-mer neighbors; a query sequence $P$, and a database sequence $T$
Output: A list of hits $(i, j)$ such that the $P$'s $i$-th $k$-mer and $T$'s $j$-th $k$-mer form a
  neighbor in $\mathcal{S}$
  for each position $i$ in $P$ do
    let $m_i$ be the $k$-mer at position $i$ of $P$
    for each $k$-mer $m_j$ such that $(m_i, m_j) \in \mathcal{S}$ do
      add the 2-tuple $\langle m_j, i\rangle$ in a hash table $H$
    end for
  end for
  for each position $j$ in $T$ do
    let $m_j$ be the $k$-mer at position $j$ of $T$
    find each integer $i$ such that $\langle m_j, i\rangle$ in $H$
    output $(i, j)$
  end for
```

For a given positive integer k and a spaced seed s, the algorithm pre-selects a set $\mathcal{S}$ of spaced k-mer pairs. $\mathcal{S}$ is used to guide the hit generation in the similarity search. For each spaced k-mer pair $(u, v) \in \mathcal{S}$, u and v are called *neighbors* of each other. We further restrict our selection so that neighborhood relation is symmetric.

With the pre-selected set $\mathcal{S}$, the algorithm can perform the hit generation as follows. First, for each sequence P in the query, the algorithm calculates all the neighbors of all the spaced k-mers of P, and indexes the neighbors in a hash table H. Secondly, the algorithm scans through the database sequence T. For each position i in T, let m_i be the spaced k-mer at the position. The hash table H is looked up to find all the k-mers in P that are neighbors of m_i. The position of each neighboring k-mer in the query sequence P provides a hit with the position i of the database sequence T. The hit generation algorithm is shown in Algorithm 1. Each hit generated is further examined by other methods (such as the Smith-Waterman algorithm) to verify whether there is a similarity around the hit.

3.4.2 Weighted Minimum Hitting Set Problem

3.4.2.1 Introduction

A good set of k-mer neighbors should hit the most of HSPs. Intuitively, if we have a set $\mathcal{H}$ of fine sampling of the real HSPs, then we can find a good set (according to some criteria) of k-mer neighbors to hit all the HSPs in $\mathcal{H}$. Since each HSP can be viewed as a set of k-mers, the problem of selecting a good set of k-mer neighbors becomes a hitting set problem.

As for the criteria of selecting hitting k-mer neighbors set, a good set $\mathcal{S}$ of k-mer neighbors should have high sensitivity. This means that each individual $s = (A, B) \in S$ should hit many HSPs implying that $P(A, B)$ should be relatively high. At the same time, a good set $\mathcal{S}$ of k-mer neighbors should not produce too many random hits. This means that $P(A) \cdot P(B)$ should be low. Therefore for an individual k-mer neighbor s, the ratio $\frac{P(A,B)}{P(A) \cdot P(B)}$ should be high. However, this will not solve the problem of k-mer pairs dependencies. Consider the situation that there is another $s_1 = (A_1, B_1)$ with a high $\frac{P(A_1,B_1)}{P(A_1) \cdot P(B_1)}$ ratio. Its detected HSPs have a large overlap with the detected HSPs of s. Including both s and s_1 will not increase much the ability to hit HSPs but will increase the probability of producing random hits. If there is a third k-mer pair $s_2 = (A_2, B_2)$ such that its detected HSPs cover the non-overlapping part of detected HSPs of s_1 and $P(A_2) \cdot P(B_2) < P(A_1) \cdot P(B_1)$, then including s and s_2 is a better choice.

Therefore, the criteria should be to find a hitting set $\mathcal{S}$ of k-mer neighbors that hits all HSP sets and minimizes $\sum_{s \in S} w(s)$, where $w(s) = P(A) \cdot P(B)$. With this criteria, random hits will be minimized and the amount of dependencies will be reduced.

3.4.2.2 Weighted Minimum Hitting Set Problem

The Hitting Set Problem is a well studied NP-hard problem. A broad range of problems can be reduced into either minimum hitting set problem or its close sibling minimum set cover problem. In this section, we will review the definition of minimum weighted hitting sets problem and a popular greedy solution for this problem.

A hitting set $\mathcal{H}$ is a set that intersects every set in a collection of sets $\mathcal{C}$. $\mathcal{H}$ is further defined as a Minimum Hitting Set if no elements can be removed from $\mathcal{H}$ without violating above hitting set property.

Given a finite set $\mathcal{U}$ as universe with $|\mathcal{U}| = m$, and a collection $\mathcal{C} = \{S_i | i \in I \subseteq \mathbb{N}\}$, then a *hitting set* is a set $\mathcal{S} \subseteq \mathcal{U}$ and $S_i \cap \mathcal{S} = \emptyset \; \forall i \in I$. That basically means $\mathcal{S}$ contains at least one element from the entire sets of $\mathcal{C}$. Let HS($\mathcal{C}$) be a collection of all hitting sets of $\mathcal{C}$, then MHS($\mathcal{C}$) be a collection of HS($\mathcal{C}$) with minimal cardinality.

In addition to the above definition, we add another constrain, a weight function $w : \mathcal{U} \rightarrow \Re^+$. Hence, determining a minimal cardinality element of MHS($\mathcal{C}$) with minimal $w(S) = \sum_{s \in S} w(s)$ is called the *Weighted Minimal Hitting Set* (WMHS) problem.

For example, Given

$$\mathcal{C} = \big\{\{\texttt{a,b,c,d}\},\ \{\texttt{a,b,d}\},\ \{\texttt{a,b}\},\ \{\texttt{c}\},\ \{\texttt{d}\}\big\}$$

and $w(a) = w(b) = w(c) = w(d) = 1$. Then both $\{a, c, d\}$ and $\{b, c, d\}$ are a correct solution of WMHS($\mathcal{C}$).

For the sake of completeness, the original algorithm of the Greedy algorithm is shown in Algorithm 2. This greedy algorithm has a known approximation ratio of

$O(\ln m)$. Because in paper [66], it had shown that the approximation factor of set cover problem is bounded by $O(\ln n)$.

Algorithm 2 GreedyWMHSP

```
Input: A finite set $\mathcal{U} = \{1, \ldots, m\}$, A collection $\mathcal{C} = \{S_1, \ldots, S_n\}$ of subsets of $\mathcal{U}$.
    A set $\{w_1, \ldots, w_m\}$ of weights associated with the elements of $\mathcal{U}$
Output: A hitting set $\mathcal{H}$ for $\mathcal{C}$
1: $\mathcal{H} = \phi$ and $temp = \mathcal{C}$
2: while $temp \neq \phi$ do
3:     Compute $\mathcal{T} = \{H_1, \ldots, H_m\}$ s.t. $H_i$ is the number of subsets in $temp$ hit by $i$
       and $i \in \mathcal{U}$
4:     Choose $i$ s.t. $H_i/w_i$ is the maximum
5:     $H = H \cup \{i\}$
6:     $temp = temp \setminus \{S_j : i \in S_j\}$
7: end while
8: return $\mathcal{H}$
```

As in Algorithm 2 which is reviewed in paper [67], at each iteration, the algorithm only concerns the remaining elements. It greedily picks the element which maximizes the ratio between the number of hit sets and the associated weight.

3.4.2.3 Reduction of WMHS problem

The problem of selecting a good set of k-mer neighbors becomes a weighted minimum hitting set problem as follows:

Let $\mathcal{K}$ be the set of all the k-mers over a finite set alphabet Σ, such that:

$$\mathcal{K} = \{u | u \text{ is a } k\text{-mer}\}$$

In case of protein homology search, Σ is the twenty commonly used amino acids.

Let $\mathcal{U}$ be a finite set contains all the spaced k-mer pairs, such as:

$$\mathcal{U} = \left\{ (u, v) | \ \forall \ u, v \in \mathcal{K} \ \right\} \tag{3.1}$$

Consider an HSP $\mathcal{P}$ of two sequences of t_1 and t_2 with length l, then $\mathcal{P} \subseteq \mathcal{U}$ can be represented as a set of k-mer pairs as following:

$$\mathcal{P} = \left\{ (u,v) \left| \begin{array}{l} u \in t_1[i \dots i+k-1] \\ v \in t_2[i \dots i+k-1] \\ \forall i = \{1, \dots, l-k+1\} \end{array} \right. \right\} \tag{3.2}$$

Given an HSP p which is from two sequences, shown in Table 3.1.

ARSYDGDFVFDDEF
ARDLEEDFVYEDEF

Table 3.1: An example of two sequences.

where p is derived from two length of 14 amino acids sequences, and p can be considered as a set of 12 of 3-mer pairs when using seed 111, hence p can be translated into its 3-mer pairs representation as following:

$$\begin{aligned} p = \{&(\texttt{ARS}{\sim}\texttt{ARD}),\ (\texttt{RSY}{\sim}\texttt{RDL}),\ (\texttt{SYD}{\sim}\texttt{DLE}), \\ &(\texttt{YDG}{\sim}\texttt{LEE}),\ (\texttt{DGD}{\sim}\texttt{EED}),\ (\texttt{GDF}{\sim}\texttt{EDF}), \\ &(\texttt{DFV}{\sim}\texttt{DFV}),\ (\texttt{FVF}{\sim}\texttt{FVY}),\ (\texttt{VFD}{\sim}\texttt{VYE}), \\ &(\texttt{FDD}{\sim}\texttt{YED}),\ (\texttt{DDE}{\sim}\texttt{EDE}),\ (\texttt{DEF}{\sim}\texttt{DEF})\} \end{aligned} \tag{3.3}$$

And given a set $\mathcal{H}$ which contains a finite number of HSPs such as:

$$\mathcal{H} = \{\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_n\} \tag{3.4}$$

Whereas to find a k-mer neighbors set $\mathcal{S} \subseteq \mathcal{U}$ to hit each HSP in $\mathcal{H}$ at least once, or $\mathcal{S} \cap \mathcal{P}_i \neq \emptyset, \forall i = 1, \dots, n$, such that the selectivity $\sum_{s \in S} w(s)$, where $s = (A, B)$ and $w(s) = P(A) \cdot P(B)$, is minimized.

The above described problem of finding a set $\mathcal{S}$ with respect to a weight function $w : \mathcal{U} \rightarrow \Re^+$ is the well-known Weighted Minimum Hitting Set (WMHS) problem. Since comparisons are made against the high scoring matches methods having selectivity M, further constrain is needed such that $\mathcal{S}$ also has a selectivity less than or equal to M, i.e. $\sum_{s \in S} w(s) \leq M$. In reality, with an iterative incremental process of constructing $\mathcal{S}$, k-mer neighbors will be added to $\mathcal{S}$ until the weight reachs M.

Finding an optimal set $\mathcal{S}$ is not feasible because WMHS is a known NP-hard problem [68]. Hence the proposed algorithm of finding a good set $\mathcal{S}$ is the result of adopting the popular greedy solution, which is reviewed in paper [67], of the WMHS problem.

3.4.3 Selection of a Good Spaced k-mer Neighbors

The k-mer neighbors set $\mathcal{S}$ is greedily selected by using a training set $\mathcal{H}$ of HSPs. Let $F_{\mathcal{H}}(m_1, m_2)$ be the total counts of the k-mer pair (m_1, m_2) in $\mathcal{H}$, and $f(m)$ be the frequency of a k-mer m in a protein sequence. The selection algorithm greedily selects the k-mer pair that maximizes $\frac{F_{\mathcal{H}}(m_1,m_2)}{f(m_1)f(m_2)}$, and dynamically updates $F_{\mathcal{H}}(m_1, m_2)$ by removing the HSPs that are hit by the currently selected k-mer pairs.

The detailed algorithm is given in Algorithm 3. Notice that $F_{\mathcal{H}}(m_1, m_2)$ is a counter of each k-mer pair that occurs in a training set of HSPs, and $f(m)$ is fixed and calculated using all the protein sequences released in July 2011 from the NCBI Refseq [69] database. In the actual implementation of Algorithm 3, each HSP in $\mathcal{H}$ has been indexed with a unique id, and for each k-mer pair with counter $F_{\mathcal{H}}$, we need to maintain a list of ids of the contributing HSPs. Because of maintaining such list, in the update process, we could decrease the value of counter $F_{\mathcal{H}}$ and remove the corresponding HSPs directly from the list rather than scanning though $\mathcal{H}$ for it.

Notice that SelectNeighbor can be implemented efficiently. When executed for the first time, steps 3-5 require the enumeration of every HSP in $\mathcal{H}$ to calculate

Algorithm 3 SelectNeighbor

```
Input: A set $\mathcal{H}$ of HSPs, a positive integer $k$, a spaced seed $s$, and a targeted
    selectivity $M$
Output: A set $\mathcal{S}$ of spaced $k$-mer neighbors
1: $\mathcal{S} \leftarrow \phi$; $t \leftarrow 0$
2: repeat
3:   for all $k$-mer pair $(m_1, m_2)$ do
4:     count the frequency $\frac{F_{\mathcal{H}}(m_1,m_2)}{f(m_1)f(m_2)}$ occurring in $\mathcal{H}$
5:   end for
6:   Let $(m_1, m_2)$ be the pair that maximize $\frac{F_{\mathcal{H}}(m_1,m_2)}{f(m_1)f(m_2)}$
7:   $S \leftarrow S \cup \{(m_1, m_2)\}$ and $t \leftarrow t + f(m_1)f(m_2)$
8:   Remove all HSPs that are hit by $(m_1, m_2)$ from $\mathcal{H}$
9: until $\mathcal{H}$ is empty or $t \geq \frac{1}{M}$
```

$F_{\mathcal{H}}$. This takes $O(N)$ time and N is the total length of the HSPs. However, in the future repetition, $F_{\mathcal{H}}$ does not need a complete recalculation and can be updated dynamically. When an HSP is being removed from $\mathcal{H}$ in Step 8, we only need to reduce the counter $F_{\mathcal{H}}(m'_1, m'_2)$ by $\frac{1}{f(m'_1,m'_2)}$ for each pair of k-mers (m'_1, m'_2) occurring in $\mathcal{H}$. A priority queue data structure is used to store all the counters and efficiently find the optimal (m_1, m_2) in Step 6. Each update of a counter takes only $\log N$ time because the size of the queue is at most N. So there are at most N updates. Therefore, the total execution time is $O(N \log N)$ if the algorithm is implemented as described above.

Theoretically, our algorithm of SelectNeighbor works on selection of good neighbors for any k-mer. However when k increases and the training data size becomes large, due to the limitation on the available computational resource, SelectNeighbor may not run effectively. For example, in order to greedily select a good k-mer neighbor, at each round of the loop at steps 3-5 in SelectNeighbor algorithm we have to consider 20^{2k} k-mer pairs, which alone may exceed the available memory space. In addition to that, when increasing k by 1, there are roughly 400 time more k-mer pairs available to choose from; we have to use relative large data set to train a good k-mer set S. Loading these data and maintaining the list of ids of the contributing HSPs of each k-mer pair would easily consume a large amount of memory space. Under such circumstance, running SelectNeighbor for k- mer neighbors becomes not feasible. Therefore, another algorithm has been developed as a work-around solution. As it will be shown in next section, SelectKmerNeighbor produces adequately positive results.

Algorithm SelectKmerNeighbor scans each HSP $h \in \mathcal{H}$ one by one, and if no existing neighbors in $\mathcal{S}$ could hit h then it will look for the k-mer pair (m_1, m_2) with highest BLOSUM62 score, such that

$$f_M(m_1, m_2) = MAX(f_M(m_i, m_j)\ \forall (m_i, m_j) \in h).$$

Then pair (m_1, m_2) will be added to $\mathcal{S}$ as a new neighbor and the selectivity of $\mathcal{S}$ will be updated accordingly. SelectKmerNeighbor stops when either all HSPs $\mathcal{H}$ have been examined or target selectivity is reached.

Algorithm 4 SelectKmerNeighbor

```
Input: A set $\mathcal{H}$ of HSPs, $|\mathcal{H}| = N$, a weight $k$ spaced seed $s$, and a targeted selectivity $M$
Output: A set $\mathcal{S}$ of spaced $k$-mer neighbors
1: $\mathcal{S} \leftarrow \phi$; $t \leftarrow 0$ ; $i = 0$
2: repeat
3:    Let $k$-mer pair $(m_1, m_2)$ be the highest scoring pair in $\mathcal{H}[i]$
4:    if $\mathcal{H}[i] \cap \mathcal{S} = \emptyset$ then
5:       $S \leftarrow S \cup \{(m_1, m_2)\}$ and $t \leftarrow t + f(m_1)f(m_2)$
6:    end if
7:    $i = i + 1$
8: until $i \geq N$ or $t \geq \frac{1}{M}$
```

In SelectKmerNeighbor, since we scan the HSPs in $\mathcal{H}$ only once, the total run time is proportional to the size of $\mathcal{H}$. Therefore, the total run time is $O(N)$ when N is the total length of the HSPs.

3.5 Experiments and Results

Four different seeding schemes were compared in our experiments. The first seeding scheme is BLASTp's default scheme. That is, with a consecutive seed 111, all 3-mer pairs that have BLOSUM62 score greater than or equal to 11 are regarded as neighbors. The second scoring scheme is similar to the first one, except that a spaced seed 11*1 is used. The third and fourth schemes all use the SelectNeighbor algorithm to train the 3-mer neighbors from training HSPs. But they use a consecutive seed 111 and a spaced seed 11*1, respectively.

The protein sequences used in the study are the complete proteomes of human, mouse, drosophila (fruit fly), cow, and pig from the Uniprot [70] database. The sequences were downloaded from the Uniprot database on April 18, 2012. The five proteomes contain 65,481, 46,439, 17,516, 26,588, and 19,572 proteins, respectively.

To compare the performances of different seeding methods, we restricted each method's selectivity to be approximately the same, and compared their sensitivity on the HSPs between a pair of the above-mentioned five proteomes. On SHARCNET, SSearch program [71], which is an efficient implementation of the Smith-Waterman algorithm, was used to generate the benchmark HSPs. Computing the local alignments results from SSearch program is actually a very time consuming task because the Smith-Waterman algorithm runs in quadratic time and tens of thousands of proteins sequences are used. However, we were able to simultaneously run many SSearch programs by spreading it on different nodes across the SHARCNET clusters after efficiently splitting the task into smaller ones; we reduced our wait time for the same results by ten folds.

The sensitivity of each seeding method is measured by the percentage of the benchmark HSPs detected by the method. The selectivity of a set $\mathcal{S}$ of k-mer neighbors is calculated by $\frac{1}{\sum_{(m_i,m_j)\in\mathcal{S}} P(m_i)P(m_j)}$. This is the reciprocal of the probability that two random positions of a query and database sequences produce a hit.

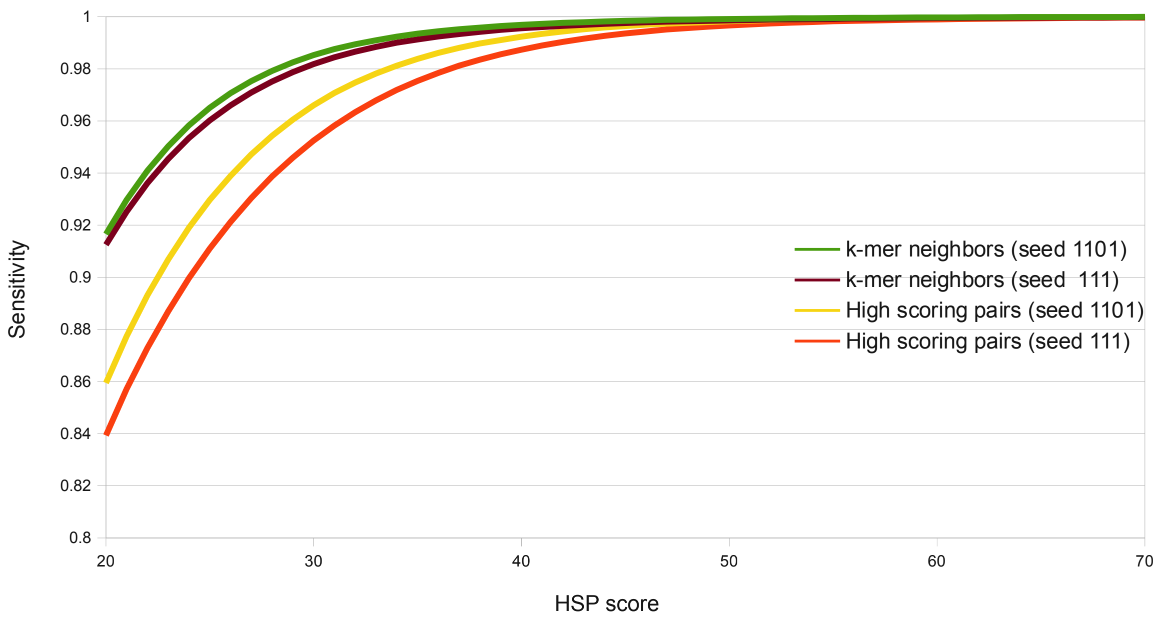

Figure 3.4: Sensitivity of each method in comparison on HSPs of human versus mouse with BLOSUM62 score $\geq X$.

For each such comparison, half of the protein sequences are randomly selected for parameter training for the seeding methods, and the remaining half are used for testing the performance. Figure 3.4 shows the experiment we conducted on the complete proteome of human and mouse. There are 38,982,312 and 38,891,790 HSPs generated by SSearch from the training and testing proteins, respectively. The sensitivity curves clearly indicate that the spaced k-mer neighbors selected by Algorithm SelectNeighbor has a much better sensitivity. The same experiment with the human and drosophila produced a similar result (Figure 3.5). There are 19,807,012 HSPs in the training data and 19,908,095 HSPs in the testing data from the protein sequences of human versus drosophila generated by SSearch.

Figure 3.8 shows the experimental result of training data from 2.165×10^8 HSPs from 134,065 protein sequences of human, mouse, drosophila, and half of the randomly

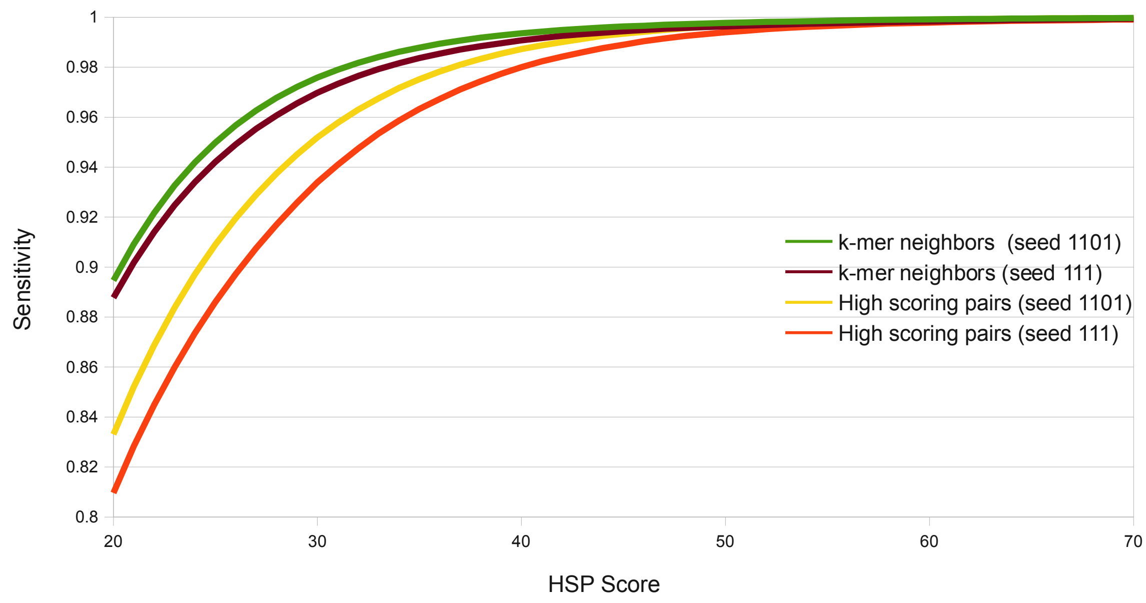

Figure 3.5: Sensitivity of each method in comparison on HSPs of human versus drosophila with BLOSUM62 score $\geq X$.

selected sequences from pig and cow respectively. Testing with 11,781,836 HSPs are from 12,248 and 9,897 unselected protein sequences from pig and cow respectively.

In the above experiments, the training and testing are on different sets of proteins from the same organisms. We have further studied the performance of spaced k-mer neighbors when the training and testing were on different pairs of organisms. The same set of k-mer neighbors trained using the training HSPs of human versus mouse was used. Figure 3.6 shows the results with the testing HSPs of human versus drosophila; and Figure 3.7 shows the testing with 995,423 HSPs generated with 1,957 randomly selected pig proteins versus 2,659 randomly selected cow proteins. Both figures suggest that the spaced k-mer neighbors still work very well even if the training data and testing data are from different organisms.

The above experiments shows the 3-mer neighbors on a moderate set of training

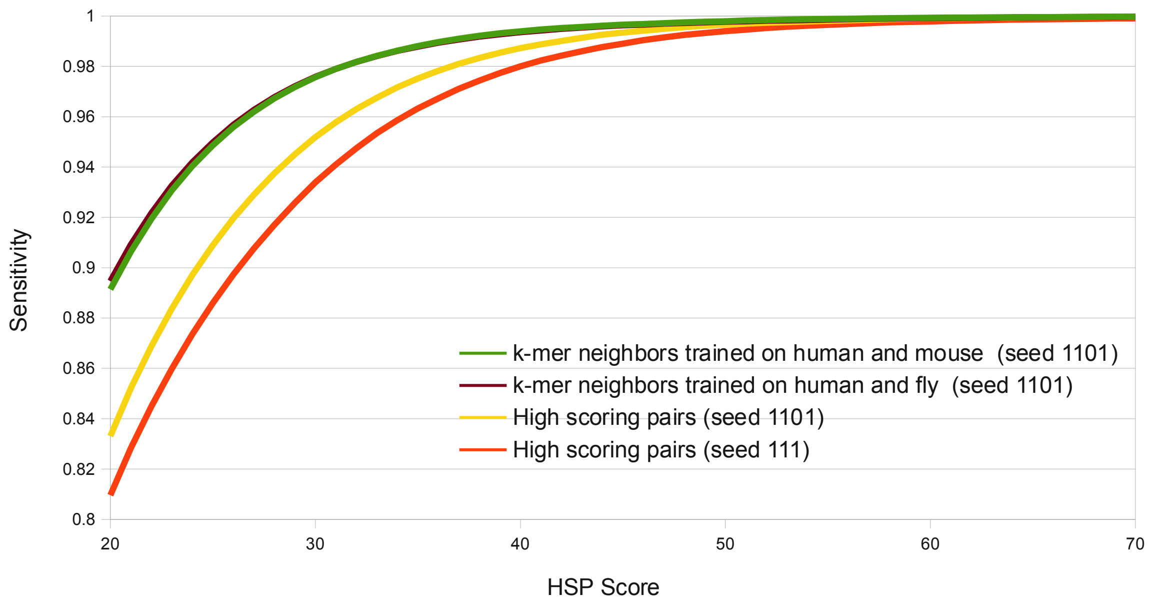

Figure 3.6: Sensitivity of each method in comparison on HSPs of human versus drosophila with BLOSUM62 score $\geq X$.

data on a few species using SelectNeighbor. Now developing a good neighbor set on the complete protein sequences is the next goal. Then a larger set of training data from the entire curated protein sequences of the December 2010 release of the Protein Clusters Database [72] from NCBI had been obtained. The protein sequences indexed to the July 2011 release of the RefSeq database had been downloaded. There are 793,848 curated proteins, when using all of them as training data, SSearch program generated billions of HSPs literately which occupy over 3 TB of hard drive space. Consider the memory requirement on this training data when using SelectNeighbor and currently the computational resource we have available; we opted to use SelectKmerNeighbor for finding good 4-mer neighbors.

Similar to the approach of 3-mer neighbors experimental results comparison, we used four different seeding schemes in our 4-mer neighbors comparison to BLASTp.

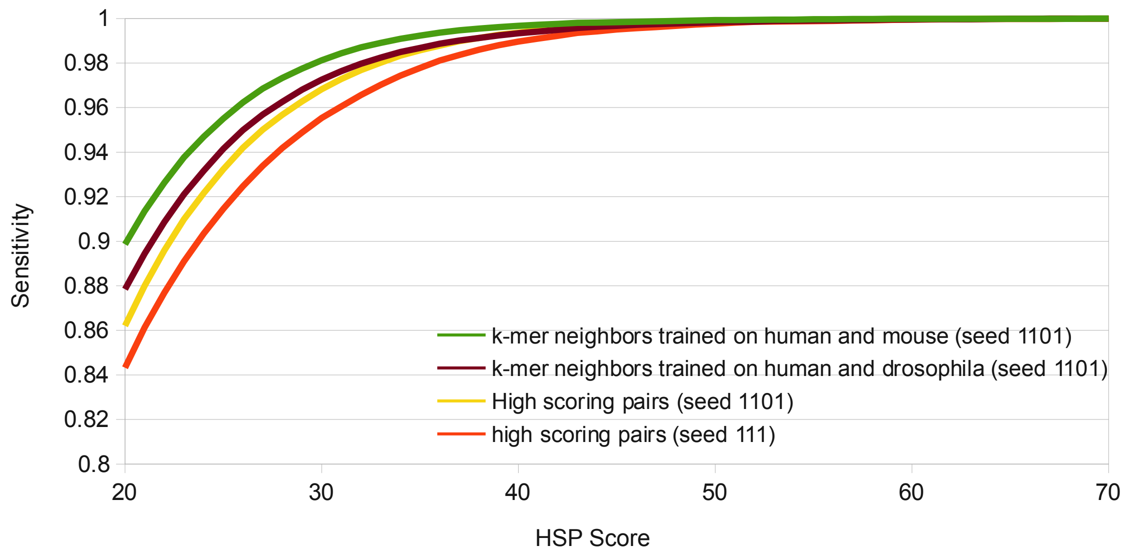

Figure 3.7: Sensitivity of each method in comparison on HSPs of pig versus cow with BLOSUM62 score $\geq X$.

The first seeding scheme was consecutive seed 1111 with BLASTp's 4-mer high scoring pairs with BLOSUM62 score greater than or equal to 12. Next scheme, we used spaced seed 11*11 instead of 1111. The last two schemes we used the 4-mer neighbors from SelectKmerNeighbor algorithm with consecutive seed 1111 and spaced seed 11*11, respectively.

The selectivity was set to match the BLASTp's 4-mer high-matching pairs with BLOSUM62 score greater than or equal to 12.

Figure 3.9 shows the experimental results on the testing data between two sets of protein sequences of 1,000 each, and the testing data were randomly selected proteins sequences not occurring in the training data from the same RefSeq release; there are 552,411 HSPs generated from the testing data by SSearch. It shows SelectKmerNeighbor algorithm produced 4-mer neighbors outperforms the program using

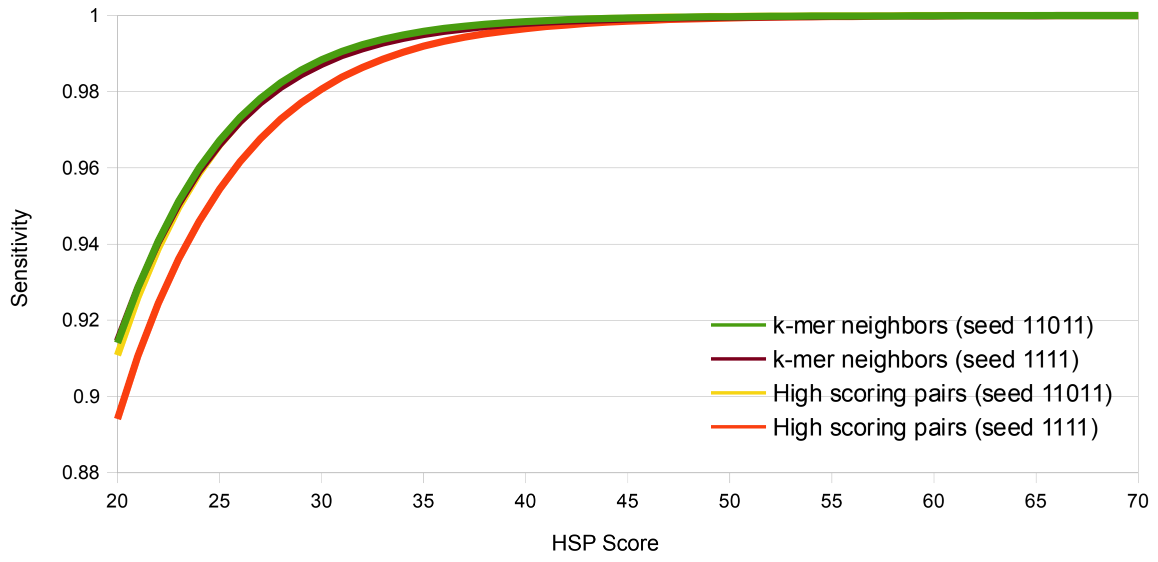

Figure 3.8: Sensitivity of each method in comparison on HSPs of randomly selected testing data with BLOSUM62 score $\geq X$.

high scoring pairs as shown in the Figure 3.9.

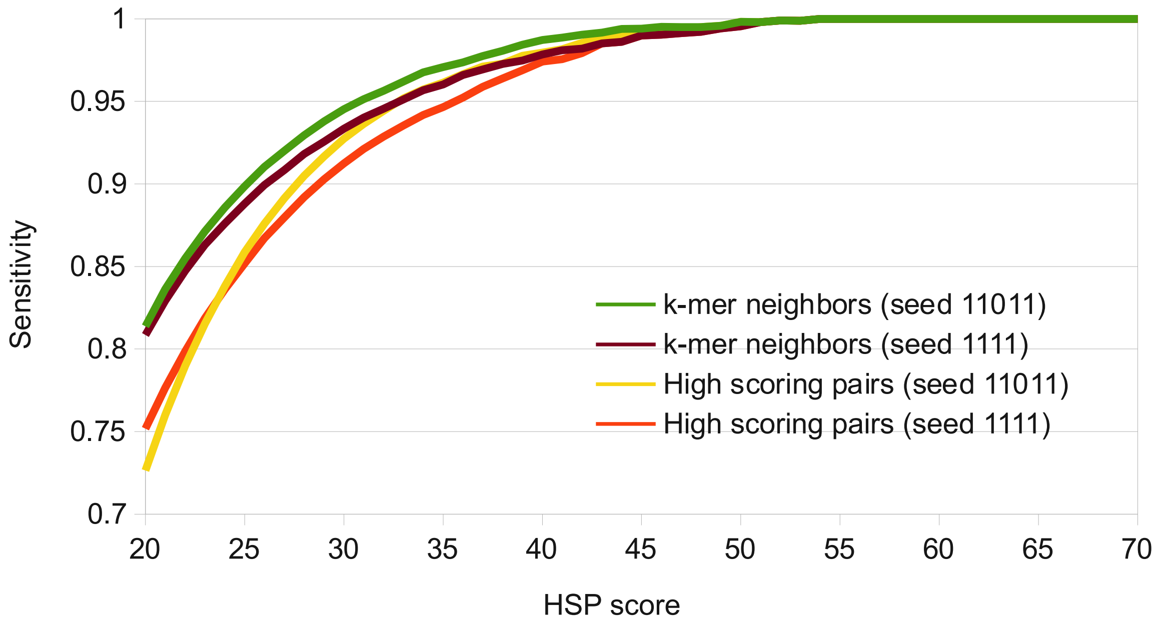

Figure 3.9: Sensitivity of each method in comparison on HSPs of randomly selected testing data with BLOSUM62 score $\geq X$.

3.6 Applying Spaced k-mer Neighbors on the Two-hit Method

In this section, the experimental results of our spaced k-mer neighbors on the two-hit method will be shown. The two-hit method was introduced in BLAST to control the quality of the hits and the execution of the hit extensions. The two-hit method requires the existence of two non-overlapped hits within a pre-defined distance on a same diagonal. The hit extensions will not be triggered prior to satisfy the conditions of the two-hit method. Our experiments are the comparisons of applying the two-hit method between the hit identified by the spaced k-mer neighbors and the hit identified by the high-scoring pairs with BLOSUM62 matrix. For each experiment, the selectivity of each method is matched.

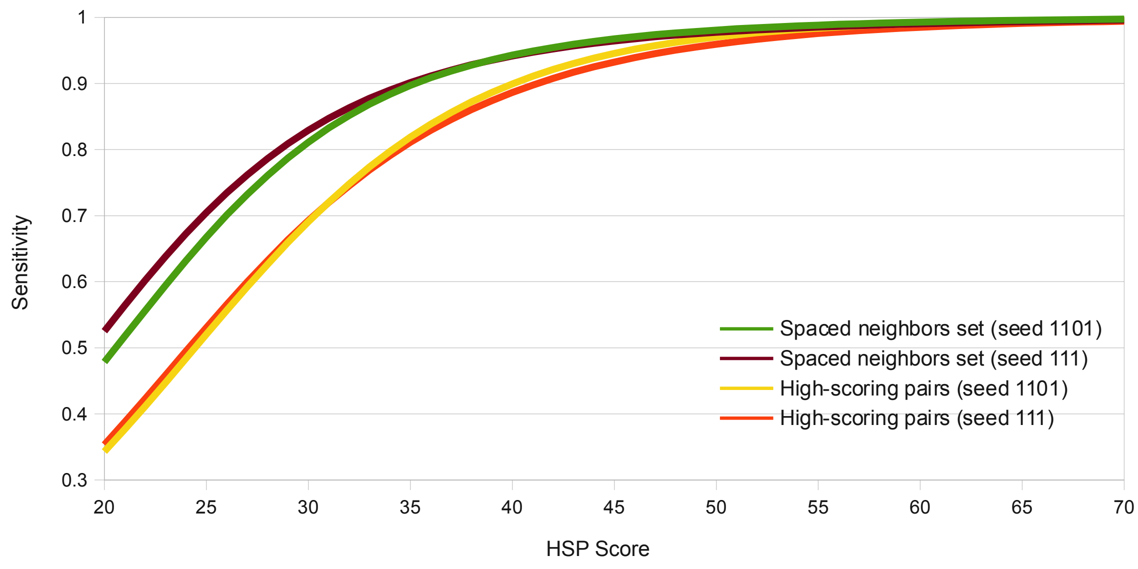

Figure 3.10: Sensitivity of each method in comparison on HSPs of randomly selected testing data with BLOSUM62 score $\geq X$.

In Figure 3.10, the testing data is the 38,891,790 testing HSPs that are generated from the randomly selected human and mouse proteins in the RefSeq database. And, the set of spaced 3-mer neighbors are trained by the training data of human and mouse proteins.

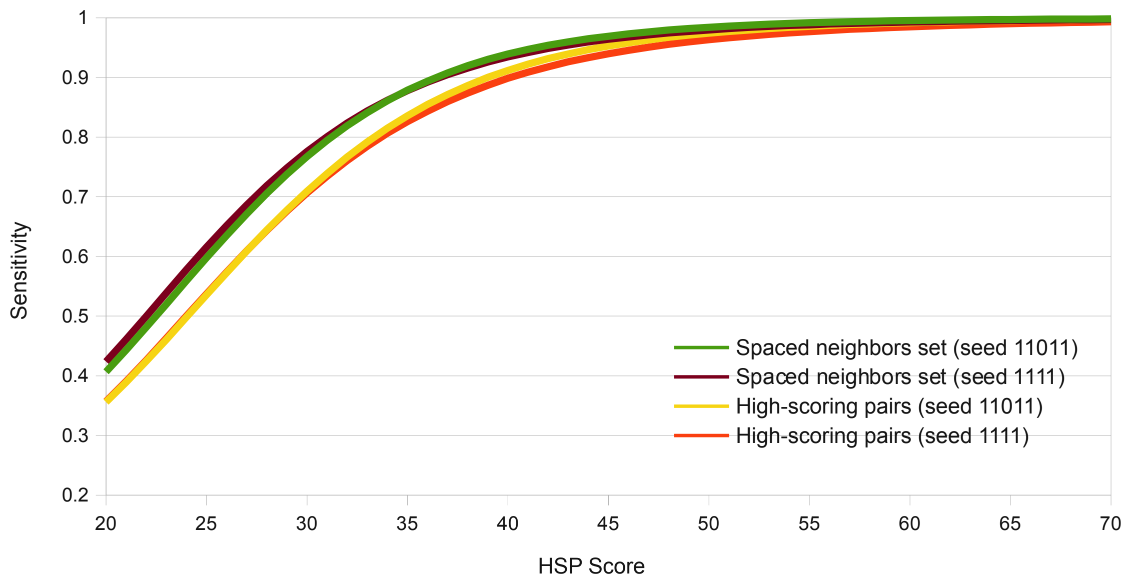

Figure 3.11: Sensitivity of each method in comparison on HSPs of randomly selected testing data with BLOSUM62 score $\geq X$.

In Figure 3.11, the testing data is the 552,411 testing HSPs that are generated between 1,000 randomly selected testing protein sequences in the RefSeq database. And, the set of spaced 4-mer neighbors are trained by 793,848 curated proteins when using SelectKmerNeighbor algorithm.

Our spaced k-mer neighbors on the 2-hit method work very effectively as the results have shown.

3.7 Conclusion

A new spaced k-mer neighbor method is proposed for more efficient tradeoff between the sensitivity and selectivity in protein similarity search. An efficient heuristic algorithm is provided to pre-select the spaced k-mer neighbors from a large set of training HSPs. Experiments showed that the method can significantly increase the search sensitivity at the same selectivity for both single-hit and 2-hit methods.

Chapter 4

HexFilter

A perfect protein homology search method aims to be both sensitive and fast. Unfortunately, in practise, sensitivity and speed are always two competing factors in the design of a homology search method. A good method should provide good tradeoff efficiency between the two factors. In Chapter 3, we introduced a new way to quickly generate a hit that indicates a potential homology. In this Chapter we focus on the next step after the hit generation, before the use of the local alignment algorithm (such as the Smith-Waterman algorithm) to construct the alignment. This step is usually called a "hit extension", which aims to further filter out the random hits before the costly local alignment. For the hit extension, BLASTp performs an ungapped alignment around the identified hit to recover a maximal scored HSP. And, local alignment is only conducted if the HSP score is above a threshold. Since the ungapped alignment can be done in linear time, comparing to the quadratic time of the local alignment, this step significantly reduces the time needed for local alignment. However, since the hit generation step produces a significant number of hits, the hit extension is still expensive. For example, the hit extension steps could take up more than 90% of the execution of the original BLASTp [15]. Therefore it is beneficial to reduce the number of hit extensions. In this Chapter, a new method will be presented, namely HexFilter, which aims to reduce the number of hit extensions.

The HexFilter method is introduced in Section 4.1. Background-related reviews are in Section 4.2. The method for quickly counting the number of identities is in

Section 4.3. The detailed experimental results on selectivity, sensitivity, and running time are in Section 4.5. At last conclusions and discussions are in Section 4.6.

4.1 Introduction

In a seeding based program, after a hit has been successfully identified, the Smith-Waterman algorithm usually will be triggered for finding the optimal local alignment. But, when many hits are identified, there will be too many calls of the Smith-Waterman algorithm. The performance of the application will be greatly weakened because the Smith-Waterman algorithm is very time consuming. Due to such case, the BLASTp introduced a heuristic method, namely the hit extension, to speed up the searching time by reducing the number of calls to the Smith-Waterman algorithm. The hit extension is executed before the Smith-Waterman algorithm. And, the heuristic of hit extension returns a maximal scored HSP around the identified hit. If the returned HSP scores higher than the pre-defined threshold, then the Smith-Waterman algorithm will be triggered. Calling the hit extensions helps to filter out some unnecessary calls to the Smith-Waterman algorithm. The hit extension runs faster than the Smith-Waterman algorithm because it is a linear time algorithm. When many calls to the Smith-Waterman algorithms are replaced by the hit extension, the searching speed of the BLASTp will be improved. But the hit extensions lose some sensitivity because the hit extension may not be successful if the hit is within a short ungapped segment of the final alignment.

As tested in [15], the time involved on the heuristic of hit extension could take up 90% of total running time of BLASTp. Hence, it is necessary to reduce the numbers of the hit extensions. For the HSPs returned by the heuristic of hit extension, it tends to have a high similarity rate and contains many identities. Intuitively, around an identified hit, rather than calculating accumulated BLOSUM62 scores and finding a maximal scored HSP, seeking for identities seems an attractive alternative. We propose a new heuristic method, namely HexFilter, to efficiently filter out potential HSPs by quickly counting the number of identities over a region around the identified hit. Special considerations need to be given to account for the similar but not identical amino acid pairs, as well as to count the number of identities efficiently. Our experimental results show that our HexFilter is efficient and can effectively reduce the number of hit extensions.

4.2 Preliminaries and Notation

4.2.1 Hit Extension

In this subsection, we will explain the algorithm of the hit extension used in BLASTp.

The heuristic of hit extension in BLASTp is a process of finding an ungapped HSP with maximal score around the initial hit. An ungapped hit extension will extend the hit along the forward and backward directions on the same diagonal. A popular solution of the hit extension is the X-drop greedy algorithm developed in [19]. The X-drop greedy algorithm sets a threshold value for the maximum allowance that the accumulative alignment score can drop. In other words, it defines the maximal difference between the current alignment score and highest alignment score that has been seen. And, the X-drop algorithm stops when the current alignment score has dropped more than the given threshold. Usually, the extension stops before reaching the full length of the sequences.

An example of an ungapped hit extension is shown in Table 4.1.

F	S	F	L	K	D	S	A	G	V	V	D	S	P	K	L	G	A	H	A	E	K	V
F	G	D	L	S	N	P	G	A	V	M	G	N	P	K	V	K	A	H	G	K	K	V

Table 4.1: The identified hit is blue coloured pair of AHA and AHG, the X-drop algorithm extends in forward for 3 positions and backward for 17 positions.

4.2.2 The Two-hit Method

BLASTp uses the hit extensions to improve searching speed, but loses some sensitivity. To recover some of the sensitivity, BLASTp has to lower its hit score threshold to generate more hits. Lowering the hit score threshold helps to increase sensitivity, but many unwanted false hits are created at the same time. The unwanted hits waste the application's running time and induce false positive results. Therefore, BLASTp introduced another heuristic, called two-hit method, to filter out the unwanted hits. The two-hit method requires that two non-overlapped hits on the same diagonal within a predefined distance must be found prior to executing the hit extensions. The hit extension is invoked on the second hit found in the two-hit method. The two-hit heuristic was first adopted in BLASTp when the gap penalty was allowed [15]. As the examples had been illustrated in paper [15], when using different hit score thresholds, the evaluation results of the one-hit and two-hit methods show that they share roughly the same sensitivity, and the numbers of calls to the hit extension are reduced, the speed of BLASTp is not slower.

The idea of examining multiple hits on the same diagonal within a search window space first appeared in 1983 [10]. The two-hit heuristic was based on the fact that on the same diagonal an HSP usually contains more than one hit within a reasonable short distance. A position of a hit h is defined as a coordinate as (i, j) between two sequences x and y if the hit occurs at the i-th position of x and the j-th position of y. The diagonal of h is the value of $i - j$. Another hit h_1 from (i_1, j_1) is considered on the same diagonal of h if and only if $i - j = i_1 - j_1$. The distance between h and h_1 is the value of $i - i_1$.

To implement the two-hit method efficiently, a hit look-up table is needed for storing the newest hit in a diagonal. Because the database is scanned in the order of increasing values of indexed positions of the sequences, so only the newest hit of a diagonal needs to be saved to the look-up table. Whenever a new hit is found, comparing with the old hit in the look-up table will return the distance of the two hits in the same diagonal. So we only update the old hit with the new hit when either two hits are overlapped or two hits are further away than the pre-defined distance threshold. However, the overheads in terms of tracking the hits in the two-hit method and maintaing the look-up table are often not properly analyzed for the related running time estimation.

In BLASTp, the default seed of the two-hit method is 111, the hit score threshold is 11 by the BLOSUM62 matrix, the maximal distance allowed between two hits is 40, and two hits are required to be non-overlapped.

4.2.3 Amino Acids Clustering

To build an efficient HexFilter, we need to quickly count the pairs of similar amino acids between two sequences. Also, regardless of the types of algorithms being used for comparing two data sequences, the data sequences need to be loaded into memory first. Finding identities between two sequences are realized by comparing each memory block containing the sequences. And, each of the amino acids are compared while scanning the memory blocks. Most of the current computer systems are 64-bit. This means a 64-bit long word is a primitive block in the computation. Therefore, it is beneficial to store multiple amino acids in one long word and carry out the comparison at once. For the 20 commonly used amino acids in Bioinformatics, a 64-bit long word can store at most 12 amino acids because each of them takes up 5 bits. So, in a 64-bit long word, 60 bits are used, but the last 4 bits will be wasted.

While counting for identities between two amino acid sequences, we also like to include some highly similar pairs. If amino acids are clustered into 16 groups, some groups may contain more than one amino acid. The amino acids were not identities but currently in the same clustered group will be considered as new identity. Therefore, some highly similar pairs become identities when using the clustered amino acids. Using the clustered 16-group amino acids can also help to compare more amino acids in a primitive block of memory. And no bits in the memory block will be wasted. If we opt to a clustered 16 groups of amino acids, then we will encode 16 amino acids in a 64-bit of long word because each clustered group of amino acids only requires 4 bits.

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

Figure 4.1: This is a rearranged BLOSUM 62 Scoring Matrix. The clustered 16-group of amino acids are in yellow coloured background. The clustered 16-group amino acids are {{C}, {G}, {A}, {T}, {S}, {N}, {D}, {E}, {Q}, {K, R}, {V, I}, {L, M}, {W}, {F, Y}, {H}, {P}}

Here, we use the classification algorithm directly from our earlier research results [64, 73]. Figure 4.1 shows the 16 groups of amino acids returned by our algorithm. The main idea of the algorithm is, initially considering each amino acid as an individual group, then clustering two amino acid groups into one if they have the highest BLOSUM62 score in the matrix, and updating the new scores between this new group and other groups. We have shown that the classification of amino acids helps to increase the sensitivity while maintaining the same selectivity if using the same spaced seeds. Our classification algorithm also respects the natural properties of different amino acids, e.g. aromatic for both F and Y, aliphatic for both V and I. Intuitively, the 16 groups of amino acids will help to increase the identities rate of an alignment over the 20 groups of amino acids. Each individual group of (K, R), (V, I), (L, M), (F, Y) will be considered as an identity if they appear in a same position of an alignment. For example, conside the case of pair (K, R), in addition to identity of the pairs (K∼K) and (R∼R), the pair (K∼R) will also be treated as identity.

4.3 Counting The Identities

Algorithm 5 countIdentities

```
Input: $x$ as a 64-bit long word for 16 amino acids from one sequence, $y$ as a 64-bit
    long word for 16 amino acids from another sequence
Output: Integer value of $i$ as the number of identities between $x$ and $y$
1: let value = ∼(x ∧ y)
2: value = value & (value ≫ 1)
3: value = value & 0x5555555555555555LL
4: value = value & (value ≫ 2)
5: value = value & 0x1111111111111111LL
6: return $i$= POPCNT(value)
```

Clustering the amino acids into smaller groups can also help to quickly count the identities between two sequences. The detailed steps of counting the identities between two clustered 16 groups of amino acids are described in Algorithm 5. For the clustered 16 groups of amino acids, each encoding of an amino acid takes 4 bits, a total of 16 amino acids can be stored in a 64-bit long word. In a long word, the i-th of the 16 amino acids starts at the index positions $4i$. Any of two amino acids are an identity if and only if their 4-bit encodings are the same. The XOR($\wedge$) bit operation turns the same bits into 0 and others into 1. Given two long words of x and y with exactly 16 amino acids inside of each, if we XOR the two long words, then the encoding of the identities will be 4 bits of 0s. As in step 1 of Algorithm 5, after negate(∼) the XOR results, in *value*, the bits of 1s represent the identity bits. Starting at the index positions of $4i$ in *value*, 4 consecutive 1s imply an identity of two amino acids. If two amino acids are not identity, in their encodings they may still share a few bits but not all 4 bits. Hence, in *value*, the bits with value 1 in non-identity amino acids need to be removed. And, we want only bit $4i$ in *value* to be 1 if it is an identity. This is achieved by the bits manipulation from step 2 to step 5 in Algorithm 5. After the bits manipulation, only the $4i$-th positions are 1 if and only if the amino acids are identity at the i-th positions from x and y. And, all other bits are 0. At last, we count the number of 1s in *value* to induce the number of identities with the POPCNT. The CPU machine instruction of POPCNT was first introduced in 2008 when Intel released its Nehalem-based Core i7 processor. The POPCNT instruction is the extension of the SSE4.2 instruction set. The POPCNT machine instruction counts the total number of bits with value 1 in a 64-bit long word. Because POPCNT is a native machine instruction, it runs very fast. And, the bit-wise operations from steps 2 to 5 in Algorithm 5 are also very fast. So, when we load the encoding of 16 clustered amino acids into one long word, we can quickly count the identities between two long words using the POPCNT after transforming the identities information into one long word. Hence, the identities between two lengths of 16 amino acids can be found quickly.

4.4 An Efficient HexFilter

In this Section, we propose the HexFilter algorithm for reducing the number of hit extensions. A simple algorithm to optimize the configuration of an HexFilter is also proposed.

As mentioned in paper [15], in order to retain reasonable sensitivity on the weaker alignments, the hit extensions are configured to run very intensively, so that, more than 90% of the computational time of BLASTp are resulted from the hit extensions. The HSP needs to have some positive scored pairs in order to get a higher accumulated score. Because only the identities and similar amino acids have positive BLOSUM62 scores, the short length and higher scored HSPs usually have higher identities rate. Therefore counting the identities in the HexFilter is a desired method. An HexFilter is invoked on the identified hits. Then, around the hit, the HexFilter seeks the least required number of identities inside a search window. Only after the conditions on the HexFilter are satisfied, the hit extensions will be triggered. Hence, the HexFilter helps to reduce the number of calls to hit extensions. The HexFilter also runs faster than the hit extensions, whereas the searching time can be improved.

A HexFilter has two parameters: i as the number of required identities, w as the search size. Our HexFilter has a fix search window size, $w = 16$. Hence a region of the length of 16 around the initial hit will be further examined. Setting $w = 16$ allows us to take advantage of the 16 groups of clustered amino acids, and effectively use the native POPCNT CPU instruction [74], so that similar amino acids between two segments of length of 16 can be found very quickly. If a hit is identified at i-th position of x and the j-th position of y, then the most common of a search window covers a space of $[i-8, i+8]$ of sequence x and $[j-8, j+8]$ of sequence y.

Because the positions that were initially hit had been evaluated through the hit generation step, they do not need to be checked again. So, it is only the other positions that were not hit will be the ones to be checked in the HexFilter. The parameters of our HexFilter for 3-mer are $(i, w = 13)$ when ignoring the identities at the 3 positions from the hit. For 4-mer, our HexFilter has parameters $(i, w = 12)$ when ignoring the identities at the 4 positions from the hit.

Our HexFilter is an outcome of iterations of empirical testing on the values of the parameters. So the tradeoff between sensitivity and selectivity is optimized. For both 3-mer and 4-mer, we test the values of $i = \{3, 4, 5, 6, 7\}$.

4.5 Experimental Results

In this Section, we present the related experimental results of the comparisons between our HexFilters and the BLASTp's methods.

4.5.1 Comparisons With the 2-hit Method

In this subsection, we test our HexFilters to benchmark selectivity and sensitivity.

Both the HexFilter and the 2-hit method can be regarded as filters to the single-hits so that only a portion of the single-hits are used for the ungapped extension.

The probability of a HexFilter (i, w) can be estimated as its selectivity if we assume that the probability of each amino acid is independent. For a hit score threshold t, the probability of a HexFilter implies that the probability of finding a hit and the probability of finding at least i identities out of w positions besides the hit.

Let Σ be the alphabet of all amino acids. The Table 4.2 shows the background probability of the amino acids in the RefSeq database.

The probability of finding an identity at a position of an alignment is shown in Equation 4.1.

$$P_{id} = \sum_{i \in \Sigma}^{|\Sigma|} P_i^2 \tag{4.1}$$

Given a finite number k, let $\mathbb{S}$ contain all k-mers; then the probability of a k-mer in an alignment, say α, is the product of the background probability of each amino acid in α, say P_α, as shown in Equation 4.2.

Amino Acid	Background Probability
A	0.089414978
R	0.055770873
N	0.040195758
D	0.054208358
C	0.012022210
Q	0.039021419
E	0.063014797
G	0.071312787
H	0.021813064
I	0.059396702
L	0.098792183
K	0.052934061
M	0.024139023
F	0.039662204
P	0.046659939
S	0.065496184
T	0.054787207
W	0.012224589
Y	0.030423743
V	0.068605104
B	0.000000069
Z	0.000000023
X	0.000104353
*(U,O,J)	0.000000372

Table 4.2: The background probability of each of amino acid occurring in the RefSeq database.

$$P_\alpha = \prod_i^k P_{\alpha[i]} \tag{4.2}$$

The probability of finding any pair of k-mer, say $(\alpha \sim \beta)$, is $P(\alpha \sim \beta)$. This implies that the probability of α and β can be found at a same position in an alignment, as shown in Equation 4.3.

$$P(\alpha \sim \beta) = P_\alpha \cdot P_\beta \tag{4.3}$$

When using a scoring matrix SM, the score of a k-mer pair $(\alpha \sim \beta)$ can be defined as $SM(\alpha, \beta)$. For a hit score threshold t, a hit of a k-mer pair occurs when they have a score of least t; the probability of finding a hit from the k-mer pairs, say p_k^t. This implies that the summation of the probability of each of k-mer pair has a score of at least t, as shown in Equation 4.4.

$$P_k^t = \sum_{\alpha,\beta \in \mathbb{S}} P(\alpha \sim \beta) \quad \text{if } SM(\alpha, \beta) \geq t \tag{4.4}$$

The probability of finding i identities within a region of w amino acids implies the following: the number of ways to choose i out of w positions, the probability of i identities out of w positions, and the probability of $(w - i)$ non-identities out of w positions, as shown in Equation 4.5.

$$P_{(i,w)} = \binom{w}{i} \cdot P_{id}^i \cdot (1 - P_{id})^{w-i} \tag{4.5}$$

For a region of 16 amino acids, the probability of a HexFilter (i, w) on k-mer, say $P_{\mathrm{HF}(i,w)}^k$, is the probability of a k-mer hit in the region of 16 amino acids for a hit score threshold t and the probability of finding at least i identities out of a region of w amino acids, shown in Equation 4.6. And, this probability can be used as its selectivity for a HexFilter.

$$P_{\mathrm{HF}(i,w)}^k = P_k^t \cdot \sum_{j=i}^{w} P_{(j,w)} \tag{4.6}$$

The selectivity of a filter can also be estimated using the probability if the k-mer pair could pass it. Consider the total number of k-mer pairs between any two sequences as the product of the lengths of two sequences. For a database search, the total number of k-mer paris becomes the product of lengths of query sequences and total database sequences. Let L_q and L_d be the total lengths of query and databases sequences, shown in Equation 4.7 and 4.8 respectively.

$$L_q = \text{Total length of query sequences} \tag{4.7}$$

$$L_d = \text{Total length of database sequences} \tag{4.8}$$

The selectivity of a filter can be estimated as a ratio of the number of k-mer pairs passing though a filter over the total number of k-mer pairs, as in Equation 4.9.

$$Selectivity = \frac{\#\text{ of } k\text{-mer pairs passing a filter}}{L_q \cdot L_d} \tag{4.9}$$

The Figures 4.2 and 4.3 in this subsection plot the results between selectivity and sensitivity of different methods at different hitting score thresholds, namely t, for spaced seeds 1101, and 11011, respectively. Sensitivity is estimated using the 38,891,790 testing HSPs found between randomly selected protein sequences from human and mouse in the RefSeq database when using the Ssearch. The selectivity of a HexFilter is calculated using Equation 4.6. The selectivity of a single-hit and 2-hit method is estimated using a query sequence and a database. For estimating the selectivity, the query sequence is a length of 335 sequences which is randomly generated according to the background probability in Table 4.2. The length of the query sequence is 335 which represents the average length of sequences in RefSeq database. The database is 3.6 millions randomly selected protein sequences from the RefSeq database. The total length of the randomly selected database sequences is 1,202,486,163. The threshold values of t are ranged as $\{9, 10, 11, 12, 13, 14, 15\}$. And, the value of t is an accumulative BLOSUM62 score of 20 groups of amino acids as per spaced seeds.

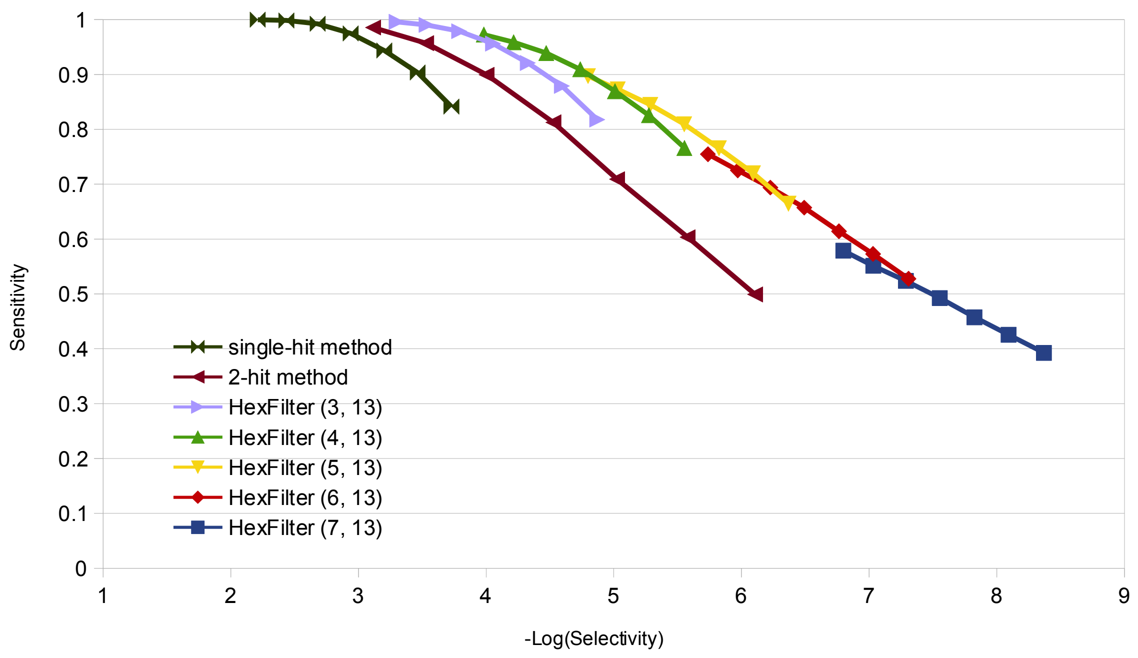

Figure 4.2: The sensitivity-selectivity curves when spaced seed 1101 is used. X-axis is the $-\log_{10}(Selectivity)$, Y-axis is the sensitivity of the HSPs with score ≥ 40. For each plotting of different heuristics, the value of hit score threshold, t, increases from left-top to right-bottom (from $t = 9$ to $t = 15$). As the value t increases, the sensitivity decreases, while the selectivity increases. Note that at roughly the same sensitivity level of 90%, HexFilter (4, 13) with $t = 12$ has better selectivity than the single-hit method with $t = 14$ and 2-hit method with $t = 11$.

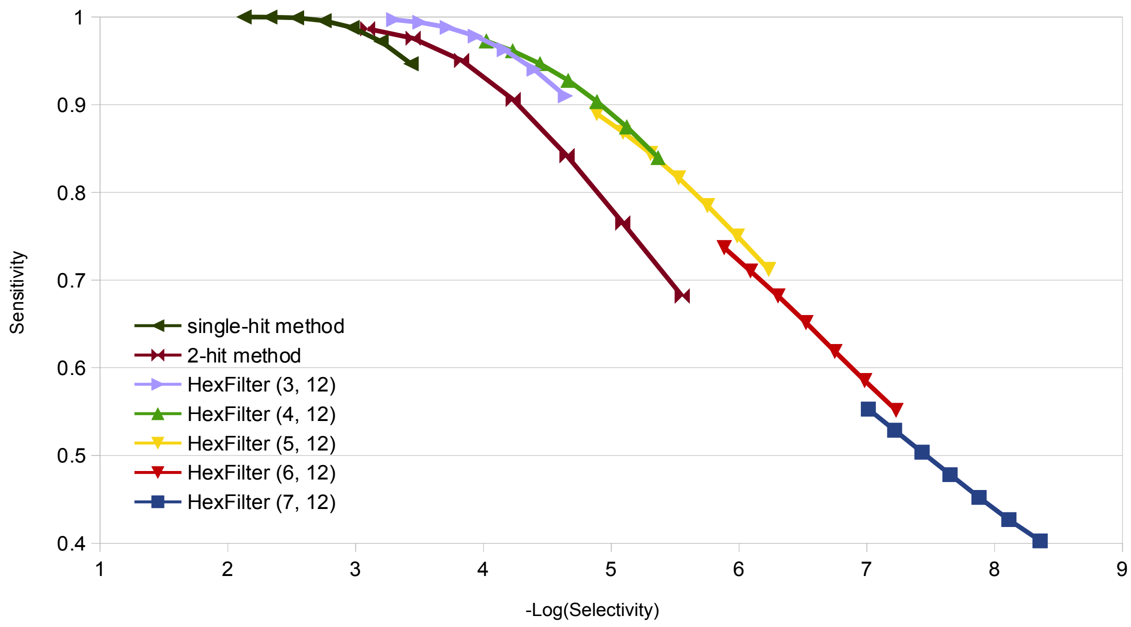

Figure 4.3: The sensitivity-selectivity curves when spaced seed 11011 is used. X-axis is the $-\log_{10}(Selectivity)$, Y-axis is the sensitivity of the HSPs with score ≥ 40. For each plotting of different heuristics, the value of hit score threshold, t, increases from left-top to right-bottom (from $t = 9$ to $t = 15$). As the value t increases, the sensitivity decreases, while the selectivity increases. Note that at roughly the same sensitivity level of 90%, HexFilter (4, 12) with $t = 13$ has a better selectivity than the 2-hit method with $t = 12$.

Figure 4.4 and Figure 4.5 show the sensitivity on HSPs with different score comparisons between our HexFilter and both the 2-hit and single-hit methods when using spaced seed 1101 and 11011 respectively. All methods use the same hit score threshold value of 11 for spaced seed 1101, and 12 for spaced seed 11011.

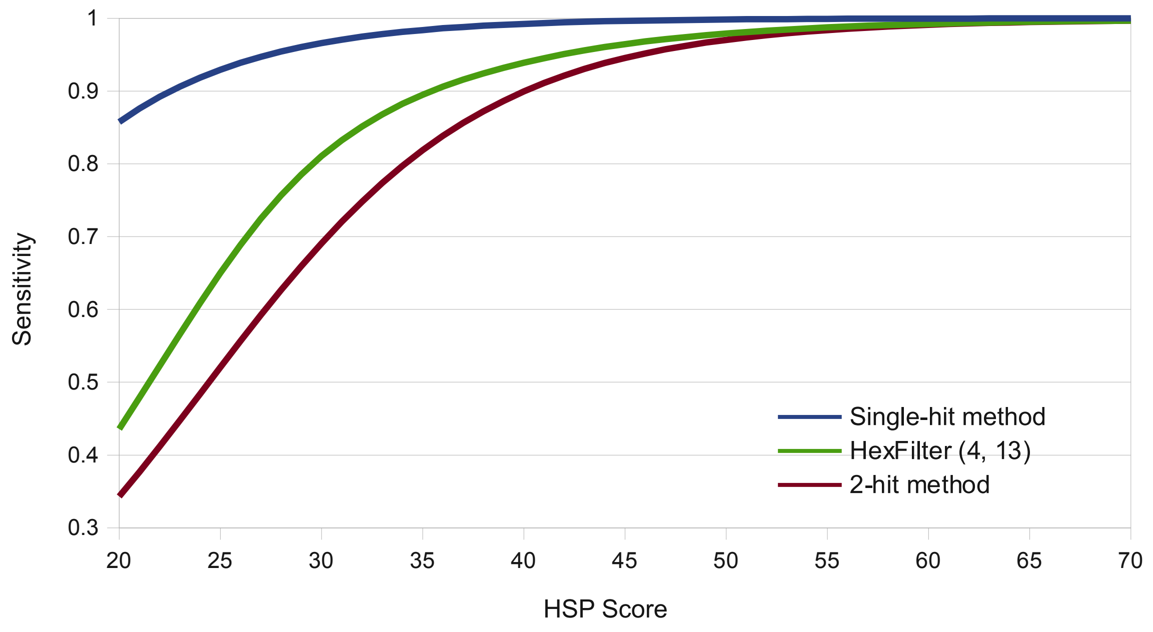

Figure 4.4: When spaced seed 1101 and hit score threshold 11 are used, the curves plot the sensitivity of each method in comparison to HSPs of human versus mouse with BLOSUM62 score $\geq X$.

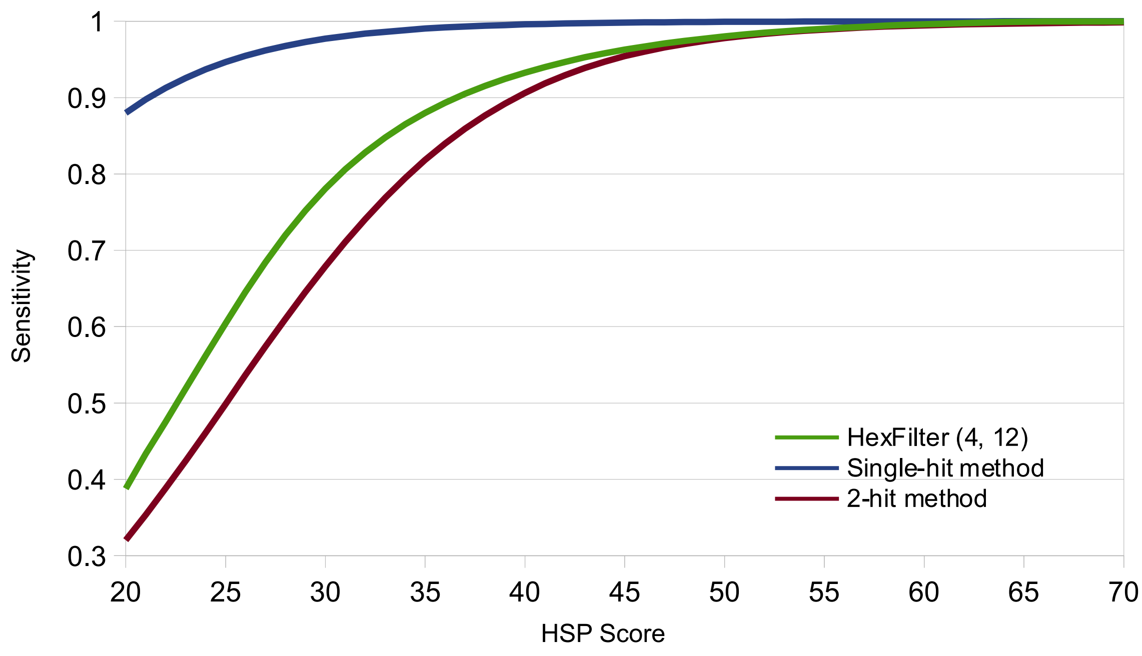

Figure 4.5: When spaced seed 11011 and the hit score threshold score 12 are used, the curves plot the sensitivity of each method in comparison to HSPs of human versus mouse with BLOSUM62 score $\geq X$.

4.5.2 Application Running Time

After examining the sensitivity and selectivity of the HexFilter method, the running time of the HexFilter method is investigated in this subsection. One large set of testing data has been used to compare the running time between our implementation of the two-hit method and HexFilter. The computer environment used for our experiments is a distribution of Debian 2.6.32-5-amd64 on an Intel i7 CPU with 24G ram.

The values of hit score threshold are ranged from 9 to 15. The spaced seeds of 1101 and 11011 are used.

The testing database has 3.6 million of randomly selected protein sequences from the RefSeq database. One randomly selected sequence, not in the testing database, of a length of 469 has been used as the testing query sequence.

No. of Sequences	3,600,000
Length of database	1,202,486,163

Table 4.3: Testing database statistics.

In general, compared to our implementation of the two-hit method, our HexFilter method triggers less number of ungapped hit extensions but can find more HSPs. When running under a similar amount of time, our HexFilter can find more HSPs than the two-hit method. The HSPs from the query sequence and anyone of the database sequences under the comparison have no repeats. An HSP is an alignment with length l and score s between two sequences x and y, say $h(i, j, l, s)$, that starts at position i in x and position j in y. It is a repeat of another HSP also between x and y, say $h_1(i_1, j_1, l_1, s_1)$, if and only if $(i = i_1, j = j_1, l = l_1, s = s_1)$. The HSPs returned by successful hit extensions that will go through a process to remove repeated HSPs. The comparison details are as follows.

When using the spaced seed 1101 and different of hit scoring threshold, Table 4.4, 4.5, and 4.6 show the detailed breakdown of the number of hits identified, the number of the hit extensions attempted and succeeded, and the number of HSPs found by our HexFilters and the two-hit method respectively. And, Table 4.7, 4.8, and 4.9 show similar information for spaced seed 11011.

When using the spaced seed 1101 and different of hit scoring threshold, Figure 4.6 shows the number of hit extensions attempted and succeeded by our HexFilters and the 2-hit method respectively. Figure 4.7 shows the number of hit extensions attempted and HSPs found by our HexFilters and the 2-hit method respectively. Figure 4.8 shows the running time and number of successful hit extensions by our HexFilters and the two-hit method respectively. Figure 4.9 shows the running time and number of HSPs found by our HexFilters and the two-hit method respectively. And, Figure 4.10, 4.11, 4.12, and 4.13 show similar information when spaced seed 11011 is used.

			Extensions		
Hit Score	Time (sec.)	Single Hits	Attempted	Succeeded	HSPs Found
9	297	3,267,230,634	281,104,457	10,999,707	3,728,155
10	197	1,905,858,653	163,790,793	8,063,930	3,370,786
11	130	1,062,232,329	91,615,508	5,694,890	2,896,533
12	92	582,334,570	50,039,676	3,934,811	2,351,943
13	61	310,454,664	26,504,876	2,638,956	1,800,352
14	42	161,765,976	13,825,558	1,713,342	1,297,870
15	25	86,963,181	7,459,236	1,133,301	922,296

Table 4.4: For our HexFilter (3, 13) method, when using the testing sequences and spaced seed 1101 for different hit score thresholds, the table shows the application's running time and the number of single hits, hit extensions attempted, hit extensions succeeded, and HSPs found.

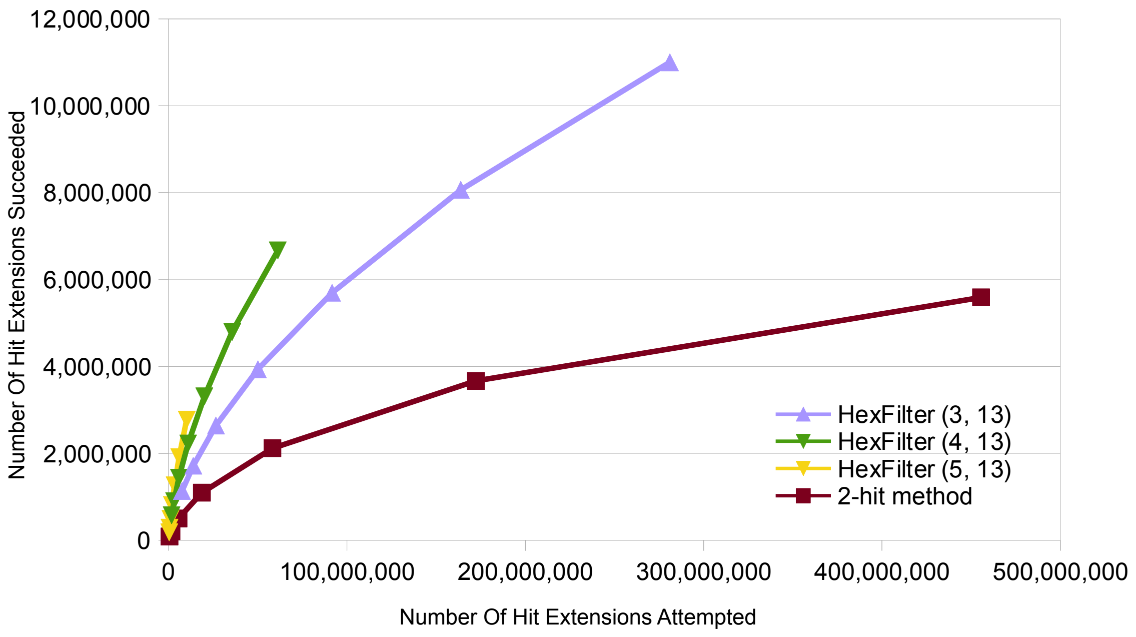

Figure 4.6: When spaced seed 1101 is used, the curves plot the number of hit extensions attempted and the number of the hit extensions succeeded for the two methods. The values of the hit score threshold are ranged from 9 to 15. A smaller value of t results in more hit extensions.

			Extensions		
Hit Score	Time (sec.)	Single Hits	Attempted	Succeeded	HSPs Found
9	243	3,267,230,634	61,146,590	6,670,331	2,714,636
10	163	1,905,858,653	35,653,115	4,796,452	2,310,091
11	108	1,062,232,329	19,987,388	3,308,657	1,867,763
12	69	582,334,570	10,922,117	2,224,986	1,428,802
13	44	310,454,664	5,771,765	1,439,234	1,027,184
14	30	161,765,976	3,019,927	899,286	696,696
15	22	86,963,181	1,644,013	574,309	471,653

Table 4.5: For our HexFilter (4, 13) method, when using the testing sequences and spaced seed 1101 for different hit score thresholds, the table shows the application's running time and the number of single hits, hit extensions attempted, hit extensions succeeded, and HSPs found.

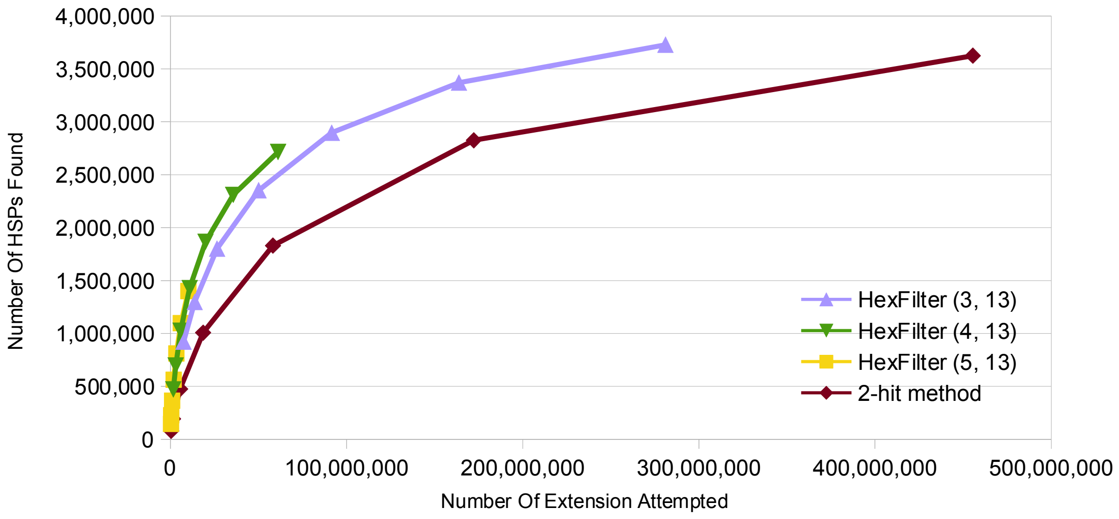

Figure 4.7: When spaced seed 1101 is used, the curves plot the number of hit extensions attempted and the number of HSPs found for the two methods. The values of the hit score threshold are ranged from 9 to 15. A smaller value of t results in finding more HSPs and more hit extensions.

			Extensions		
Hit Score	Time (sec.)	Single Hits	Attempted	Succeeded	HSPs Found
9	221	3,267,230,634	455,549,558	5,587,640	3,625,581
10	131	1,905,858,653	172,219,789	3,665,965	2,825,000
11	79	1,062,232,329	58,217,427	2,118,053	1,830,530
12	49	582,334,570	18,568,282	1,094,338	1,007,225
13	32	310,454,664	5,508,789	499,401	474,121
14	22	161,765,976	1,533,252	202,674	193,837
15	18	86,963,181	457,880	84,294	79,748

Table 4.6: For the 2-hit method, when using the testing sequences and spaced seed 1101 for different hit score thresholds, the table shows the application's running time and the number of single hits, hit extensions attempted, hit extensions succeeded, and HSPs found.

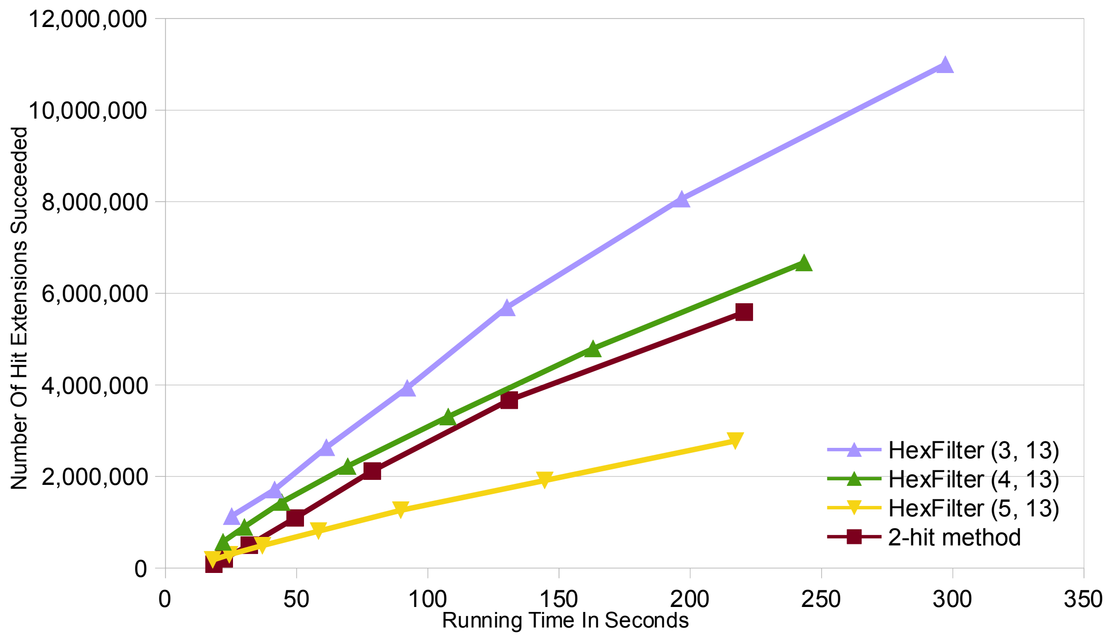

Figure 4.8: When the spaced seed of 1101 is used, the curves plot the running time and the number of the hit extensions succeeded for the two methods. The values of the hit score threshold are ranged from 9 to 15. A smaller value of t results in finding more successful extension but longer running time.

			Extensions		
Hit Score	Time (sec.)	Single Hits	Attempted	Succeeded	HSPs Found
9	383	4,007,364,658	284,599,138	13,924,051	3,922,558
10	285	2,499,036,149	177,610,930	10,917,844	3,632,502
11	208	1,530,784,354	108,872,646	8,404,428	3,274,159
12	147	920,608,930	65,442,477	6,337,265	2,847,144
13	99	546,517,642	38,877,228	4,683,704	2,392,426
14	70	318,406,868	22,669,895	3,377,204	1,930,269
15	51	180,854,182	12,890,650	2,369,008	1,499,270

Table 4.7: For our HexFilter (3, 12) method, when using the testing sequences and spaced seed 11011 for different hit score thresholds, the table shows the application's running time and the number of single hits, hit extensions attempted, hit extensions succeeded, and HSPs found.

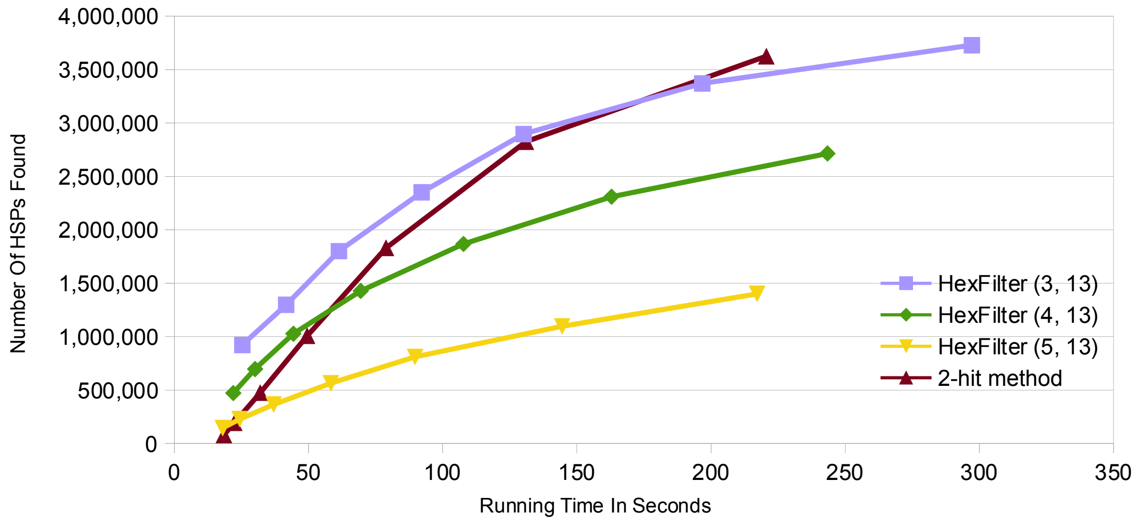

Figure 4.9: When the spaced seed of 1101 is used, the curves plot the running time and the number of HSPs found for the two methods. The values of the hit score threshold are ranged from 9 to 15. A smaller value of t results in finding more HSPs but longer running time.

			Extensions		
Hit Score	Time (sec.)	Single Hits	Attempted	Succeeded	HSPs Found
9	320	4,007,364,658	56,151,803	7,626,898	2,801,172
10	241	2,499,036,149	35,099,368	5,848,095	2,447,767
11	168	1,530,784,354	21,552,902	4,383,741	2,065,414
12	124	920,608,930	12,977,741	3,205,096	1,778,912
13	81	546,517,642	7,718,309	2,282,815	1,314,621
14	56	318,406,868	4,509,646	1,580,632	993,874
15	41	180,854,182	2,579,638	1,057,844	718,863

Table 4.8: For our HexFilter (4, 12) method, when using the testing sequences and spaced seed 11011 for different hit score thresholds, the table shows the application's running time and the number of single hits, hit extensions attempted, hit extensions succeeded, and HSPs found.

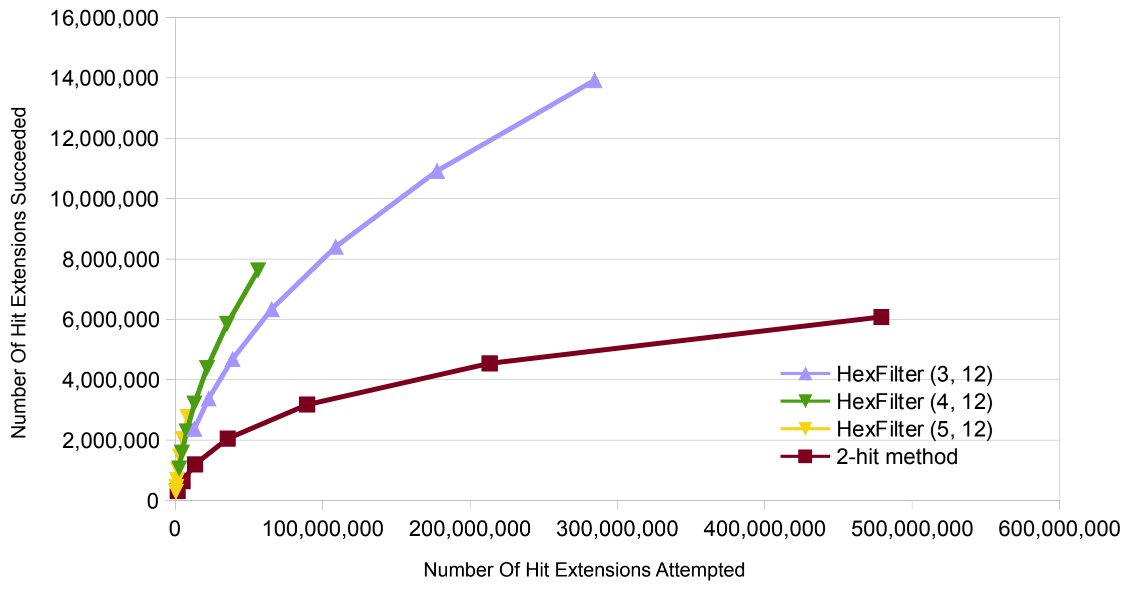

Figure 4.10: When spaced seed 11011 is used, it plots the number of hit extensions attempted and the number of hit extensions succeeded for the two methods. The hit score thresholds are ranged from 9 to 15. A smaller value of t results in more hit extensions.

			Extensions		
Hit Score	Time (sec.)	Single Hits	Attempted	Succeeded	HSPs Found
9	357	4,007,364,658	479,222,047	6,074,062	3,902,753
10	287	2,499,036,149	213,148,909	4,538,522	3,387,131
11	210	1,530,784,354	89,619,309	3,176,588	2,652,328
12	131	920,608,930	35,598,974	2,045,020	1,694,458
13	96	546,517,642	13,508,145	1,194,407	1,115,321
14	70	318,406,868	4,889,215	636,523	606,068
15	54	180,854,182	1,666,229	306,884	293,512

Table 4.9: For the 2-hit method, when using the testing sequences and spaced seed 11011 for different hit score thresholds, the table shows the application's running time and the number of single hits, hit extensions attempted, hit extensions succeeded, and HSPs found.

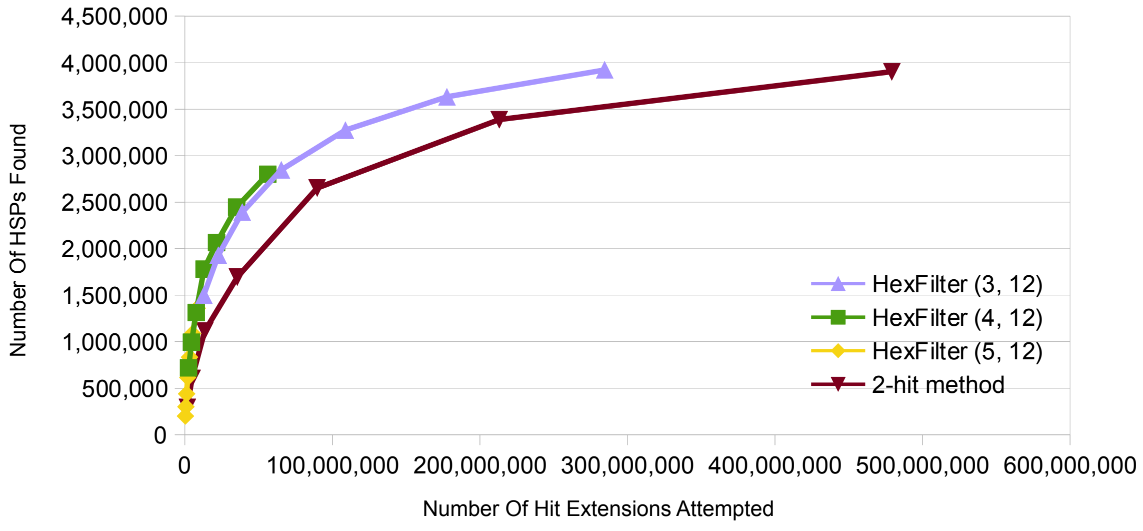

Figure 4.11: When spaced seed 11011 is used, it plots the number of hit extensions attempted and the number of HSPs found for the two methods. The hit score thresholds are ranged from 9 to 15. A smaller value of t results in finding more HSPs and more hit extensions.

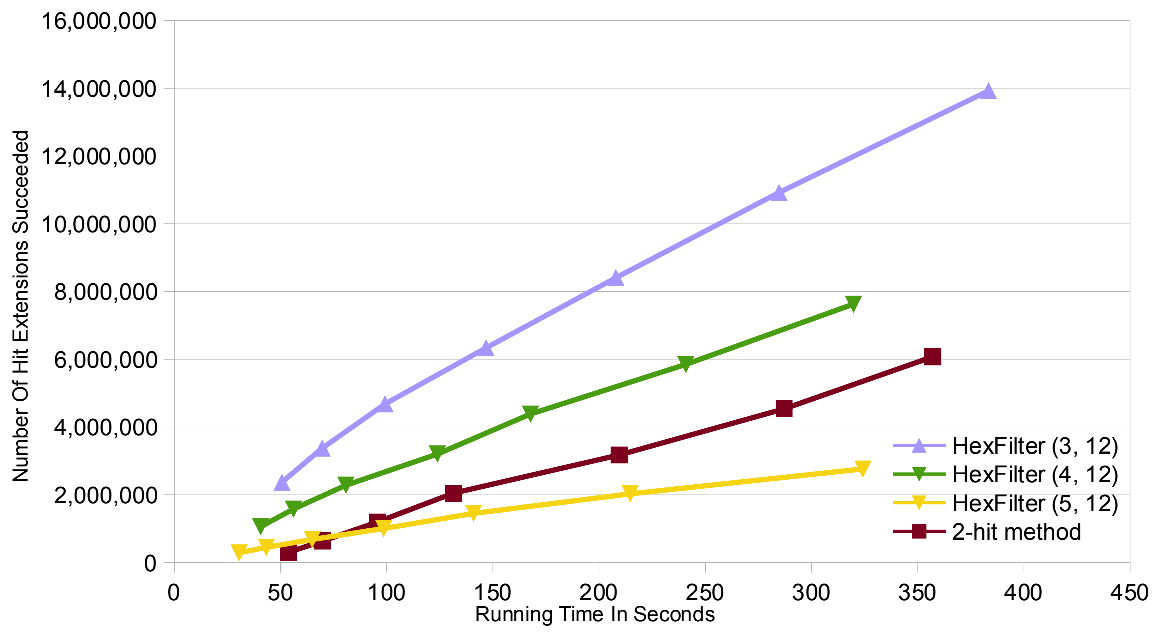

Figure 4.12: When spaced seed 11011 is used, the curves plot the running time and the number of hit extensions succeeded for the two methods. The hit score thresholds are ranged from 9 to 15. A smaller value of t results in finding more successful extensions but longer run time.

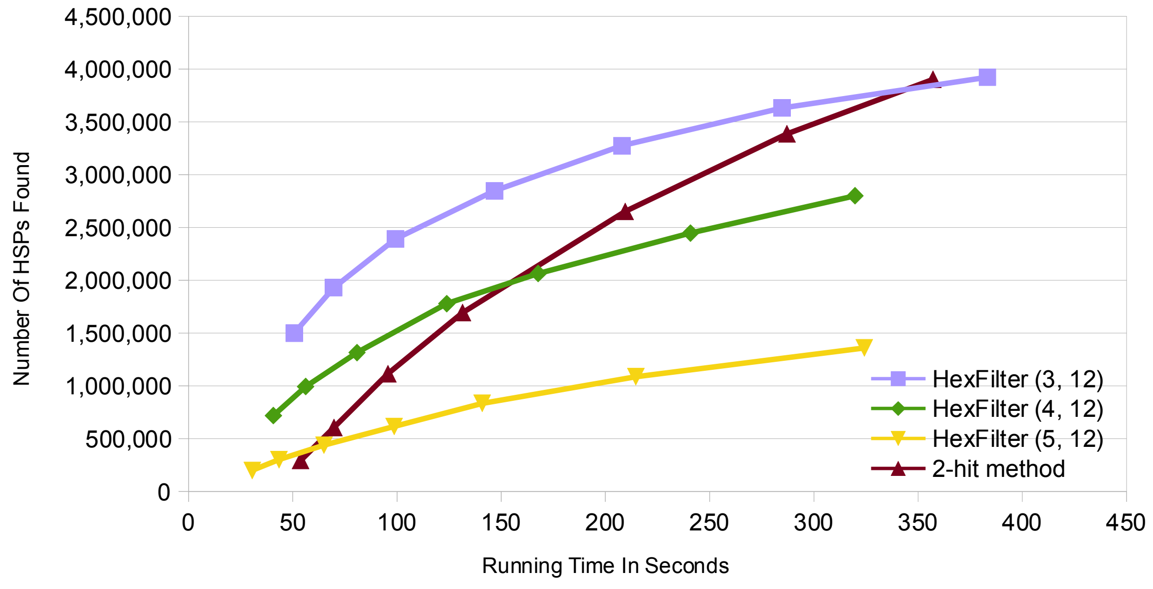

Figure 4.13: When spaced seed 11011 is used, the curves plot the running time and the number of HSPs found for the two methods. The hit score thresholds are ranged from 9 to 15. A smaller value of t results in finding more HSPs but longer run time.

4.6 Conclusion and Discussion

The new HexFilter method is highly effective to reduce the number of ungapped hit extensions while achieving better selectivity. We used the clustered 16 groups of amino acids in our HexFilter; it allows us to include highly similar pairs as identities. And, our HexFilter could quickly count the number of identities between the two sequences using the POPCNT machine instruction.

In theory, our HexFilter can be combined with the two-hit method to create a more efficient filter. More specifically, after the two-hit method suggests an extension, our HexFilter would be used on the second hit to determine if the extension is really necessary. The ungapped hit extension will only be triggered if the HexFilter has also passed. Hence, even fewer ungapped hit extensions will be attempted. As a result, the running time will be further reduced.

The combination of spaced seeds, two hit, and HexFilter will provide a great deal of flexibilities in fine tuning the performance of a practical homology search system. Such fine tuning flexibilities can be very powerful when the concerned protein database, the type of search, and the similarity level of homology are specific.

Chapter 5

Conclusion and Future Work

Almost half a century has passed since the initial publication of the sequence alignment algorithm by Levenshtein [3] in 1965. Yet the sequence similarity search is still an actively researched problem in bioinformatics. Two reasons keep this problem active. First of all, sequence similarity serves as the foundation of function similarity, which provides an essential research method for genetic and proteomic researchers. Secondly, as the production of the data is getting cheaper and easier, huge quantities of data are generated on a daily basis. However the growth of computer technology is slower than the expansion of the database. Therefore, new or improved algorithms are in constant demand to accommodate the growth of data.

In this thesis, we have proposed two new ideas, the spaced k-mer neighbors and the HexFilter, to help improve the speed and sensitivity of sequence similarity search. Computer programs have been implemented to test out the ideas.

The spaced k-mer neighbor aims to improve the seed generation phase of a homology search method. In the protein homology search, seeding controls the overall sensitivity of the final results and speed of application. We have shown that the spaced k-mer neighbors method is an efficient tradeoff between sensitivity and speed. Our proposed algorithms are simple and effective in finding a good set of spaced k-mer neighbors. When comparing to the BLASTp, as the experimental results have shown, our set of spaced k-mer neighbors can significantly increase the sensitivity while maintaining same level of selectivity. Moreover, the spaced k-mer neighbors are easy to adapt because it involves very little modification of existing codes. For example, BLASTp can use our set of spaced k-mer neighbors to directly replace its k-mer high-scoring pairs.

The HexFilter method aims to improve the seed extension phase of a homology search method. Our HexFilter has better selectivity and can effectively reduce the number of ungapped hit extensions. Our HexFilter takes advantage of our previously published amino acids clustering. It also takes advantage of the fast CPU native instruction, POPCNT. The HexFilter method compares favourably with the 2-hit method that had been used for the same purpose. In addition to that, our HexFilter can be potentially used in combining with the 2-hit method to achieve even more efficient tradeoff between sensitivity and speed.

As for future works, we envision that better spaced k-mer neighbors can be produced from a larger training data set when the required computational resources become available with sufficiently large memory. We would also like to produce a set of spaced 5-mer neighbors.

Bibliography

[1] D. Benson, I. Karsch-Mizrachi, D. Lipman, J. Ostell, D. Wheeler, D. Benson, I. Karsch-Mizrachi, D. Lipman, J. Ostell, D. Wheeler *et al.*, "Genbank," *Nucleic Acids Research*, vol. 36, no. Database issue, pp. D25–D30, 2008.

[2] E. Lander, L. Linton, B. Birren, C. Nusbaum, M. Zody, J. Baldwin, K. Devon, K. Dewar, M. Doyle, W. FitzHugh *et al.*, "Initial sequencing and analysis of the human genome," *Nature*, vol. 409, no. 6822, pp. 860–921, 2001.

[3] V. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Cybernetics and Control Theory*, vol. 10, no. 8, pp. 707–710, 1966.

[4] S. Needleman and C. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of Molecular Biology*, vol. 48, no. 3, pp. 443–453, 1970.

[5] T. Smith and M. Waterman, "Identification of common molecular subsequences," *Journal of Molecular Biology*, vol. 147, pp. 195–197, 1981.

[6] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of Molecular Biology*, vol. 215, no. 3, pp. 403–410, 1990.

[7] B. Ma, J. Tromp, and M. Li, "PatternHunter: faster and more sensitive homology search," *Bioinformatics*, vol. 18, no. 3, pp. 440–445, 2002.

[8] D. Lipman and W. Pearson, "Rapid and sensitive protein similarity searches," *Science*, vol. 227, no. 4693, pp. 1435–1441, 1985.

[9] W. Pearson and D. Lipman, "Improved tools for biological sequence comparison," *Proceedings of the National Academy of Sciences*, vol. 85, no. 8, pp. 2444–2448, 1988.

[10] W. Wilbur and D. Lipman, "Rapid similarity searches of nucleic acid and protein data banks," *Proceedings of the National Academy of Sciences*, vol. 80, no. 3, pp. 726–730, 1983.

[11] M. Sternberg, *Protein structure prediction: a practical approach.* Oxford University Press, USA, 1996, vol. 170.

[12] C. Camacho, G. Coulouris, V. Avagyan, N. Ma, J. Papadopoulos, K. Bealer, and T. Madden, "Blast+: architecture and applications," *BMC Bioinformatics*, vol. 10, no. 1, pp. 1–9, 2009.

[13] T. Madden, R. Tatusov, and J. Zhang, "Applications of network blast server," *Methods in enzymology*, vol. 266, pp. 131–141, 1996.

[14] A. Morgulis, G. Coulouris, Y. Raytselis, T. Madden, R. Agarwala, and A. Schäffer, "Database indexing for production megablast searches," *Bioinformatics*, vol. 24, no. 16, pp. 1757–1764, 2008.

[15] S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman, "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs," *Nucleic Acids Research*, vol. 25, no. 17, pp. 3389–3392, 1997.

[16] S. Altschul and W. Gish, "Local alignment statistics," *Methods in enzymology*, vol. 266, no. 2, pp. 460–480, 1996.

[17] S. Karlin and S. F. Altschul, "Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes," *Proceedings of*

the National Academy of Sciences of the United States of America, vol. 87, no. 6, pp. 2264–2268, 1990.

[18] A. Dembo, S. Karlin, and O. Zeitouni, "Limit distribution of maximal non-aligned two-sequence segmental score," *The Annals of Probability*, vol. 22, no. 4, pp. 2022–2039, 1994.

[19] Z. Zhang, S. Schwartz, L. Wagner, and W. Miller, "A greedy algorithm for aligning dna sequences," *Journal of Computational Biology*, vol. 7, no. 1-2, pp. 203–214, 2000.

[20] A. Biegert and J. Söding, "Sequence context-specific profiles for homology searching," *Proceedings of the National Academy of Sciences*, vol. 106, no. 10, pp. 3770–3775, 2009.

[21] R. Lopez, V. Silventoinen, S. Robinson, A. Kibria, and W. Gish, "Wu-blast2 server at the european bioinformatics institute," *Nucleic acids research*, vol. 31, no. 13, pp. 3795–3798, 2003.

[22] S. Schwartz, W. Kent, A. Smit, Z. Zhang, R. Baertsch, R. Hardison, D. Haussler, and W. Miller, "Human-mouse alignments with BLASTZ," *Genome Research*, vol. 13, pp. 103–107, 2003.

[23] W. Gish and D. J. States, "Identification of protein coding regions by database similarity search," *Nature Genetics*, vol. 3, no. 3, pp. 266–272, 1993.

[24] W. J. Kent, "BLAT–the BLAST-like alignment tool," *Genome Research*, vol. 12, no. 4, pp. 656–664, 2002.

[25] B. Brejova, D. G. Brown, and T. Vinar, "Vector seeds: an extension to spaced seeds," *Journal of Computer and System Sciences*, vol. 70, no. 3, pp. 364–380, 2005.

[26] L. Ming, M. Bin, D. Kisman, and J. Tromp, “Patternhunter ii: Highly sensitive and fast homology search,” *Journal of Bioinformatics and Computational Biology*, vol. 2, no. 03, pp. 417–439, 2004.

[27] B. Brejová, D. Brown, and T. Vinař, “Optimal spaced seeds for homologous coding regions,” *Journal of Bioinformatics and Computational Biology*, vol. 1, no. 4, pp. 595–610, January 2004.

[28] G. Kucherov, L. Noe, and M. Roytberg, “Multi-seed lossless filtration,” in *Proc. of the 15th Symposium on Combinatorial Pattern Matching (CPM), LNCS 3109*, 2004, pp. 297–310.

[29] M. Csuros, “Performing local similarity searches with variable length seeds,” in *Combinatorial Pattern Matching: 15th Annual Symposium, CPM 2004, Istanbul, Turkey, July 5-7, 2004, Proceedings*, vol. 15. Springer, 2004, pp. 373–387.

[30] M. Csürös and B. Ma, “Rapid homology search with neighbour seeds,” *Algorithmica*, vol. 48, no. 2, pp. 187–202, 2007.

[31] F. Nicolas and E. Rivals, “Hardness of optimal spaced seed design,” *Journal of Computer and System Sciences*, vol. 74, no. 5, pp. 831–849, 2008.

[32] M. Li, B. Ma, and L. Zhang, “Superiority and complexity of the spaced seeds,” in *Symposium on Discrete Algorithms: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, vol. 22, no. 26, 2006, pp. 444–453.

[33] U. Keich, M. Li, B. Ma, and J. Tromp, “On spaced seeds for similarity search,” *Discrete Applied Mathematics*, vol. 138, no. 3, pp. 253–263, 2004.

[34] B. Ma and M. Li, “On the complexity of the spaced seeds,” *Journal of Computer and System Sciences*, vol. 73, no. 7, pp. 1024–1034, 2007.

[35] I.H.Yang, S.H.Wang, H. Chen, P. Huang, and K.M.Chao, "Efficient methods for generating optimal single and multiple spaced seeds," in *Proc. of IEEE 4th Symp. on Bioinformatics and Bioengineering*, 2004, pp. 411–418.

[36] J. Xu, D. Brown, M. Li, and B. Ma, "Optimizing multiple spaced seeds for homology search," *Journal of Computational Biology*, vol. 13, no. 7, pp. 1355–1368, 2006.

[37] L. Ilie and S. Ilie, "Fast computation of good multiple spaced seeds," in *Algorithms in Bioinformatics*, vol. 4645, 2007, pp. 346–358.

[38] U. Keich, M. Li, B. Ma, and J. Tromp, "On Spaced Seeds of similarity search," *Discrete Appl. Math.*, vol. 138, pp. 253–263, 2004.

[39] J. Buhler, U. Keich, and Y. Sun, "Designing seeds for similarity search in genomic DNA," in *Proc. of the 7th International Conference on Computational Biology (RECOMB)*, 2003, pp. 67–75.

[40] K. Choi and L. Zhang, "Sensitive analysis and efficient method for identifying optimal spaced seeds," *Journal of Computer and System Sciences*, vol. 68, pp. 22–40, 2004.

[41] Y. Sun and J. Buhler, "Designing multiple simultaneous seeds for dna similarity search," *Journal of Computational Biology*, vol. 12, no. 6, pp. 847–861, 2005.

[42] G. Kucherov, L. Noe, and Y. Ponty, "Estimating seed sensitivity on homogeneous alignments," in *Proc. of the 4th IEEE Symposium on Bioinformatics and Bioengineering (BIBE)*, 2004, pp. 387–394.

[43] K. Choi, F. Zeng, L. Zhang *et al.*, "Good spaced seeds for homology search." *Bioinformatics (Oxford, England)*, vol. 20, no. 7, pp. 1053–1059, 2004.

[44] B. Brejova, D. Brown, and T. Vinar, "Optimal spaced seeds for hidden markov models, with application to homologous coding regions," *Lecture Notes in Computer Science*, vol. 2676, pp. 42–54, 2003.

[45] D. Brown, "Multiple vector seeds for protein alignment," in *Proc. of 4th Workshop on Algorithms in Bioinformatics (WABI)*, 2004, pp. 170–181.

[46] H. Lin, Z. Zhang, M. Zhang, B. Ma, and M. Li, "Zoom! zillions of oligos mapped," *Bioinformatics*, vol. 24, no. 21, pp. 2431–2437, 2008.

[47] D. Kisman, M. Li, B. Ma, and L. Wang, "tpatternhunter: gapped, fast and sensitive translated homology search," *Bioinformatics*, vol. 21, no. 4, pp. 542–544, 2005.

[48] J. Buhler, U. Keich, and Y. Sun, "Designing seeds for similarity search in genomic dna," *Journal of Computer and System Sciences*, vol. 70, no. 3, pp. 342–363, 2005.

[49] L. Ilie, S. Ilie, and A. M. Bigvand, "Speed: fast computation of sensitive spaced seeds," *Bioinformatics*, vol. 27, no. 17, pp. 2433–2434, 2011.

[50] Y. Sun and J. Buhler, "Designing multiple simultaneous seeds for dna similarity search," *Journal of Computational Biology*, vol. 12, no. 6, pp. 847–861, 2005.

[51] G. Kucherov, N. LAURENT, and M. Roytberg, "A unifying framework for seed sensitivity and its application to subset seeds," *Journal of bioinformatics and computational biology*, vol. 4, no. 02, pp. 553–569, 2006.

[52] M. O. Dayhoff, R. M. Schwartz, and B. C. Orcutt, "A model of evolutionary change in proteins," *Atlas of Protein Sequence and Structure*, vol. 5, no. 1.52, pp. 345–352, 1978.

[53] S. Henikoff and J. G. Henikoff, “Amino acid substitution matrices from protein blocks,” *Proceedings of the National Academy of Sciences of the United States of America*, vol. 89, no. 22, pp. 10 915–10 919, 1992.

[54] K. Chao and L. Zhang, *Sequence comparison: theory and methods.* Springer-Verlag New York Inc, 2009, vol. 7.

[55] D. Krane and M. Raymer, *Fundamental concepts of bioinformatics.* Benjamin Cummings San Francisco, 2003, vol. 1.

[56] D. Jones, W. Taylor, and J. Thornton, “The rapid generation of mutation data matrices from protein sequences,” *Computer applications in the biosciences: CABIOS*, vol. 8, no. 3, pp. 275–282, 1992.

[57] G. Gonnet, M. Cohen, and S. Benner, “Exhaustive matching of the entire protein sequence database,” *Science*, vol. 256, no. 5062, pp. 1443–1445, 1992.

[58] S. Henikoff and J. Henikoff, “Automated assembly of protein blocks for database searching,” *Nucleic Acids Research*, vol. 19, no. 23, pp. 6565–6572, 1991.

[59] H. Smith, T. Annau, and S. Chandrasegaran, “Finding sequence motifs in groups of functionally related proteins,” *Proceedings of the National Academy of Sciences*, vol. 87, no. 2, pp. 826–830, 1990.

[60] M. Styczynski, K. Jensen, I. Rigoutsos, and G. Stephanopoulos, “Blosum62 miscalculations improve search performance,” *Nature biotechnology*, vol. 26, no. 3, pp. 274–275, 2008.

[61] S. Altschul, J. Wootton, E. Gertz, R. Agarwala, A. Morgulis, A. Schäffer, and Y. Yu, “Protein database searches using compositionally adjusted substitution matrices,” *Febs Journal*, vol. 272, no. 20, pp. 5101–5109, 2005.

[62] S. Henikoff and J. G. Henikoff, "Automated assembly of protein blocks for database searching," *Nucleic Acids Research*, vol. 19, pp. 6565–6572, 1991.

[63] D. G. Brown, M. Li, and B. Ma, "A tutorial of recent developments in the seeding of local alignment," *Journal of Bioinformatics and Computational Biology*, vol. 2, no. 4, pp. 819–842, 2004.

[64] W. Li, B. Ma, and K. Zhang, "Amino acid classification and hash seeds for homology search," *Bioinformatics and Computational Biology*, vol. 5462, pp. 44–51, 2009.

[65] A. Gambin, S. Lasota, M. Startek, M. Sykulski, L. Noé, G. Kucherov *et al.*, "Subset seed extension to protein blast," in *Proceedings of the International Conference on Bioinformatics Models, Methods and Algorithms*, 2011, pp. 149–158.

[66] U. Feige, "A threshold of ln n for approximating set cover," *Journal of the ACM (JACM)*, vol. 45, no. 4, pp. 634–652, 1998.

[67] A. Cincotti, V. Cutello, and F. Pappalardo, "An ant-algorithm for the weighted minimum hitting set problem," in *Swarm Intelligence Symposium, 2003. SIS'03. Proceedings of the 2003 IEEE*, 2003, pp. 1–5.

[68] R. Karp, "Reducibility among combinatorial problems," *50 Years of Integer Programming 1958-2008*, pp. 219–241, 2010.

[69] K. Pruitt, T. Tatusova, and D. Maglott, "Ncbi reference sequences (refseq): a curated non-redundant sequence database of genomes, transcripts and proteins," *Nucleic acids research*, vol. 35, no. suppl 1, pp. D61–D65, 2007.

[70] U. Consortium *et al.*, "Reorganizing the protein space at the universal protein resource (uniprot)," *Nucleic Acids Research*, vol. 40, pp. D71–D75, 2012.

[71] X. Huang, R. Hardison, and W. Miller, “A space-efficient algorithm for local similarities,” *Computer applications in the biosciences: CABIOS*, vol. 6, no. 4, pp. 373–381, 1990.

[72] W. Klimke, R. Agarwala, A. Badretdin, S. Chetvernin, S. Ciufo, B. Fedorov, B. Kiryutin, K. ONeill, W. Resch, S. Resenchuk *et al.*, “The national center for biotechnology information’s protein clusters database,” *Nucleic acids research*, vol. 37, no. suppl 1, pp. D216–D223, 2009.

[73] W. Li, “Amino acid clustering for protein homology search,” Master’s thesis, University of Western Ontario, London, ON, Canada, 2006.

[74] D. Knuth, *Art of Computer Programming, Volume 4, Fascicle 1: Bitwise Tricks & Techniques; Binary Decision Diagrams.* Addison-Wesley Professional, 2009.

Curriculum Vita

Name:	Weiming Li
Place of birth:	Shenyang, China
Education and Degrees:	Master of Science, 2006 B.Sc. Honors, 2005 in Computer Science The University of Western Ontario London, Ontario
Related Work Experience:	Teaching and Research Assistant The University of Western Ontario

Publications:

Weiming Li, Bin Ma, and Kaizhong Zhang,
Amino Acid Classification and Hash Seeds for Homology Search,
In *Bioinformatics and Computational Biology*,
vol. 5462, pp. 44-51, 2009.

Weiming Li, Bin Ma, and Kaizhong Zhang,
Efficient Filtration for Similarity Search with Spaced k-mer Neighbors,
In *Proceedings of 2012 IEEE International Conference on Bioinformatics and Biomedicine*, pp. 11-16, 2012. **Won Best Student Paper Award.**