A tree-structured index algorithm for Expressed Sequence Tags clustering

Benjamin Kumwenda
0408046X

Supervisor: Professor Scott Hazelhurst

April 21, 2008

Declaration

I declare that this dissertation is my own, unaided work. It is being submitted for the degree of Master of Science in the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination in any other university.

Benjamin Kumwenda
21 April 2008

Abstract

Expressed sequence tags (ESTs) are complementary deoxyribonucleic acid (cDNA) fragments, which are reverse transcribed from mature ribonucleic acid (mRNA), a direct gene transcript. ESTs are a readily rich information source of complete expressed gene sequences. They reveal the type and number of genes being expressed in an organism. Joining ESTs into complete gene sequences is computationally expensive because they are numerous, erroneous, redundant and mixed up. ESTs that originate from the same gene are grouped together. This enables efficient consensus sequences generation, which reveals underlying gene sequences and their possible alternative splicings. EST clustering enables efficient discovery of expressed genes based on which several fields rely such as: disease diagnostics, drug discovery, genetic engineering, alternative splicing and many others. Most clustering algorithms developed so far are quadratic and their running time is prohibitively high. A tree-structured index algorithm has been developed to efficiently cluster ESTs with respect to running time and quality of generated clusters. The algorithm clusters ESTs in a pseudometric space by recursively partitioning a data set of EST windows into two disjointed sets. Performance of the algorithm was tested with respect to running time and quality of generated clusters. Further experiments were performed to investigate the effectiveness of the triangle inequality, which was implemented to reduce distance computations during clustering. Experimental results show that the algorithm has a running time closer to linear with a 100% specificity, but it fluctuates in sensitivity. Implementation of the triangle inequality did not significantly improve the performance of the algorithm.

Dedication

To my lovely wife, Grace.

Acknowledgements

I am so grateful to my supervisor, Professor Scott Hazelhurst for all the time he has spent directing and guiding me throughout this research. I am thankful to my family for always supporting and encouraging me in my studies. I would like to thank the University of Malawi, College of Medicine and the National Bioinformatics Networks for sponsoring this work. The Wits Bioinformatics created such a great environment for me to work smoothly on this research for which I am grateful.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction

Livingthings are classified into two main groups. These are eukaryotes (multicellular) and prokaryotes (unicellular) [Hunter 1991]. Viruses are livingthings but they are not part of the two groups and they form their own group. Eukaryotes are further divided into five kingdoms. Plants and animals are two examples of the five kingdoms. Bacteria are an example of prokaryotes. This research focuses on eukaryotes.

Organs in multicellular organisms have different structures and functions. These organs collectively work together in systems such as digestive, skeletal, circulatory and reproductive and many others to achieve specific functions. The functions include: reproduction, locomotion, protection, sensation, digestion and excretion. Cells form building blocks of these organs and they get specialised for such functions in their respective organs. In unicellular organisms, these tasks are handled by a single cell. Genes found in deoxyribonucleic acid (DNA) regulate the biological development, functionality and structure of cells, consequently the organs and systems in multicellular organisms.

DNA is a nucleic acid with four molecules joined together to form double strands. They encode instructions that specify biological development, functionality and structure of all cellular forms of life. Genes are locations in DNA that encode these specific instructions. They determine types of proteins that are produced in the body. Proteins carry out these specific instructions in various ways. Some of the ways in which proteins carry out these instructions are: providing structural support, providing mechanisms for acquiring and transferring energy and acting as catalysts in chemical reactions [Yona 1999].

A gene is active when it gets transcribed into mRNA, which gets translated to proteins. And the resulting protein carries out the functions encoded by the gene. Different genes are active in different parts of the body at different times and rates. Genes that are active in the eye are different from those that are active in the liver. Hence, functionality of cells in the eye is different from cells in the liver. Active genes are generally referred to as expressed genes. Levels of gene expression vary in different parts of the body. Problems in gene expression such as overexpression, underexpression and expression of a damaged gene may result in diseases. Identifying and understanding expressed genes is very important for reasons such as disease diagnostics, drug discovery, genetic engineering, phylogenetic studies, single nucleotide polymorphism studies and many others. The need to identify and understand active genes has led to the production of expressed sequence tags (ESTs) [Davison 2001]. The fact that ESTs are part of expressed genes makes them so important and useful in identifying active genes.

The EST production process begins with DNA in the cell nucleus of multicellular organisms being transcribed to mRNA. mRNA is either exported out of the cell nucleus to cytoplasm where it is transcribed to proteins or, under special circumstances, it is naturally reverse transcribed to cDNA, which may get reinserted into the original DNA strand [Marques *et al.* 2005]. For purposes of gene sequencing, mRNA is artificially reverse transcribed into cDNA because unlike mRNA, cDNA is double stranded and stable for experimental purposes. However, there are problems in the reverse transcription process. A full length cDNA cannot be reverse transcribed at once due to limited capability of sequencing techniques available at present. Available techniques sequence shorter pieces of cDNA at a time. The transcription process begins and ends at random positions in mRNA. This causes overlapping and redundancy among the pieces. Different types of mRNA transcribed from different genes are randomly selected from the cell cytoplasm. This mixes up the different types of mRNA being reverse transcribed. The process gets even more complicated with alternative splicing, where the same gene transcribes two or more different mRNA. In addition to all these, different types of errors occur during the reverse transcription process, which causes the mixed up and redundant pieces of cDNA to be erroneous. To achieve meaningful and accurate results, ESTs are transcribed in large numbers, which contribute to the complexity of the research problem.

ESTs are joined together to form consensus sequences that reveal complete expressed gene sequences. Joining mixed up, redundant and erroneous ESTs to form complete expressed gene sequences is computationally expensive. EST clustering is a technique that reduces this computational complexity by grouping ESTs originating from the same expressed gene into one cluster. Grouped ESTs are efficiently joined to form consensus sequences to reveal the underlying complete expressed gene sequences. Different ways of generating consensus sequences from ESTs in the same cluster represent alternative splicings of the same expressed gene. This research focuses on the clustering problem and does not proceed to deal with consensus generation to reveal underlying expressed genes.

The EST clustering problem is well known and has attracted research attention from various fields and disciplines such as Bioinformatics, Molecular Biology, Statistics and Computer Science. Several algorithms have been developed to cluster ESTs. Most of the algorithms developed so far, are computationally expensive because of the way they perform clustering. Since EST are erroneous and redundant, they are clustered in huge volumes in order to achieve meaningful clusters. However, this causes the clustering problem to be even more complex. This has opened up a research area that aims at finding efficient algorithms to cluster ESTs. This research is one of such efforts and it has contributed to this field by developing an efficient EST clustering algorithm based on a tree-structured index. The algorithm is introduced in section 1.3 and is detailed in chapter three.

1.2 Research Problem

A large number of ESTs have been sequenced and are currently found in various public databases such as NCBI's dbEST, UniGene and TIGR. Joining ESTs into their original complete gene sequences is a challenge because they are numerous, redundant, erroneous and mixed up as a result of the reverse transcription process through which they are generated. ESTs originating from the same gene overlap. This is a useful property in determining levels of similarity in order to place them into one cluster [Hazelhurst 2003]. Clustering helps to reduce the computational cost when joining overlapping ESTs into consensus sequences to reveal underlying genes.

Most prior algorithms cluster ESTs by performing pairwise comparison between pairs of sequences to determine their levels of similarity. Sequences are divided into overlapping windows, which are compared against each other using distance functions that measure their level of similarity. A threshold value is placed on the level of similarity as a cut-off point at which two windows and eventually sequences are considered similar. Such approaches take a quadratic running time of $O(p^2)$ to complete the clustering process, where $p = mn$ is the total number of windows in a data set of n sequences and an average of m overlapping windows per sequence.

Distance is computed and compared for every window to all mn windows in a data set. Such distance computation and comparison among sequence windows in a huge data set is prohibitively expensive. This research has designed and implemented an EST clustering algorithm, which successfully clusters ESTs with a reduced number of distance computations. The goal of the research was to at least achieve a running time less than the quadratic running time.

1.3 A Tree-Structured Index Algorithm

This section briefly describes the tree-structured index algorithm developed in this research. Figure 1.1 aids in the description of the algorithm. The rectangles in the figure represent different stages in the clustering process and numbers in the rectangles are used to show the order of events. The algorithm starts by storing sequences into a data structure in linear time. The sequences are partitioned into overlapping windows, which are distributed in a metric space to form clusters as shown in the rectangle labelled step 1. They are distributed with respect to their level of similarity.

Two far apart windows are chosen as pivots and the rest of the windows are compared to them. Windows closer to pivot one are placed in one partition and those that are closer to pivot two are also placed in another partition. This divides the space into two as shown by the red circles thereby creating partitions p1 and p2 in step 2. A rectangle on the right side of the diagram shows the tree construction process on the left side of the diagram. The process recursively continues and divides partition p2 into two more partitions p3 and p4 as shown by blue circles in step 3. Partition p1 is also divided into two partitions, p5 and p6 in step 4. The process terminates when a threshold point is reached or there are no more windows to partition. A threshold point defines a cut-off at which two windows are considered similar.

This process eventually constructs a tree structure as shown by rectangles on the right side of the figure. At step 2, the first two partitions, p1 and p2 are created. Partitions p3 and p4 are created from p2 in step 3. A similar process continues on p1 as shown in step 4. Colours of the created partitions match space partitioning shown by rectangles on the left side of the figure. The algorithm constructs a tree-structured index in a pseudometric space by recursively selecting two far apart windows as pivots and partitioning the remaining windows into two disjointed groups. Windows are assigned to a group of a pivot to which they are relatively closer. When construction of the tree-structured index is completed, windows from the same sequence are distributed among several leaves (final partitions) of the tree. The algorithm then merges sequences with windows in the same leaf, thereby generating clusters.

Merging is efficiently achieved by the union-find data structure, which has an amortised running time of $O(\log^* p)$. An advantage of using this data structure is that it does not increase the overall complexity of the algorithm. The algorithm does $2p$ work per recursive call, where 2 is the number of pivots and p the total number of windows. When the resulting tree is balanced,

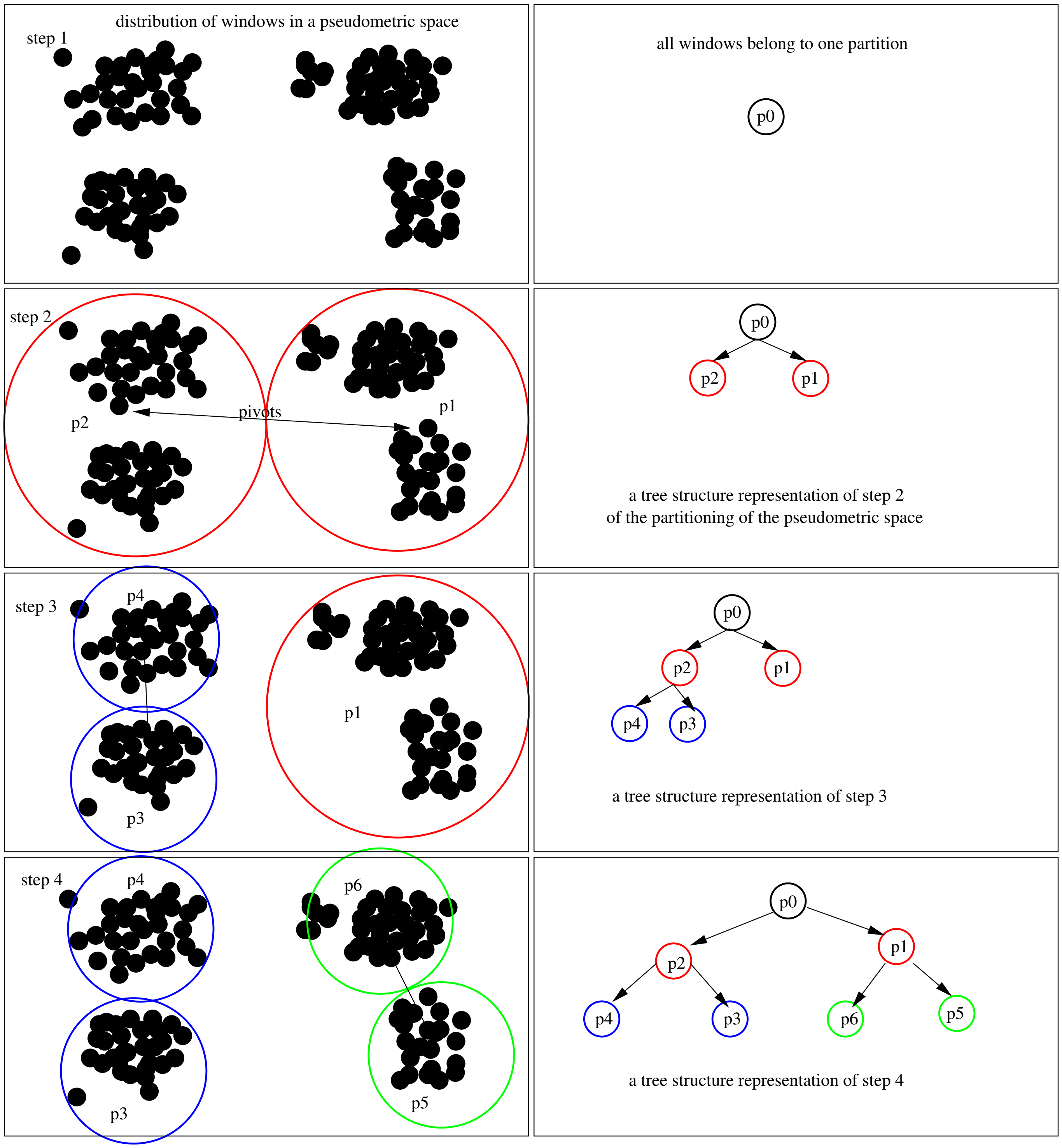

distribution of windows in a pseudometric space
step 1
all windows belong to one partition
p0
step 2
p2
pivots
p1
p0
p2
p1
a tree structure representation of step 2
of the partitioning of the pseudometric space
step 3
p4
p3
p1
p0
p2
p1
p4
p3
a tree structure representation of step 3
step 4
p4
p3
p6
p5
p0
p2
p1
p4
p3
p6
p5
a tree structure representation of step 4

Figure 1.1: Diagrammatic description of the algorithm

at most, it has a depth of $\log p$ and with $O(p)$ work per level, it yields an overall complexity of $O(p \log p)$. Experimental results have shown that the algorithm runs time closer to linear. The mathematical analysis provides an upperbound on the running time of the algorithm.

The algorithm has been developed to improve memory usage in two ways. Firstly, it stores sequences in one data structure. The algorithm only stores information that uniquely identifies windows in their respective sequences. A window is identified by a sequence index position in the input data structure and its starting position in a sequence. Secondly, the algorithm stores only resulting leaves of the tree at each level and it does not create internal nodes. Internal nodes are useful for purposes of searching but not clustering, hence storing them is a wastage of space.

Two main factors that affect the running time of the algorithm are pivot selection and the distance function used to determine similarity among sequence windows. A distance function with quadratic complexity forms a running time bottleneck and heavily slows down the algorithm. The manners in which pivots are chosen in the algorithm have got a bigger effect on the running time of the algorithm. Choosing pivots that are at the far ends of a partion is expensive and comprises the running time of the algorithm.

1.4 Research Questions

The algorithm presented in the previous section is based on ideas and the work of Giladi *et al.* [2002] and Chavez *et al.* [2001]. Giladi *et al.* [2002] developed successfully a Sequence Search Tree algorithm for searching in metric space. Chavez *et al.* [2001] explored ideas and techniques that could be applied in metric space for searching. Based on this work, it was conceived in this research that the same ideas could be applied for clustering. Hence, the development of the tree-structured index algorithm to cluster ESTs in metric space.

Two research questions arise from the implementation of the tree-structured index algorithm described in section 1.3. The first question examines whether the clustering process is achievable in the expected $O(p \log p)$ running time. The second one questions whether the tree-structured index algorithm successfully clusters EST sequences. Successful clustering in this case implies that better quality clusters in terms of sensitivity and specificity are generated. The research has further investigated the extent to which the triangle inequality can be applied to reduce distance computations and evaluations during clustering in order to improve running time.

Answering the first question involves finding the overall complexity of the algorithm from the combined complexity of all modules of the algorithm. Running time of the algorithm has been compared to that of a quadratic algorithm. Using average running times, the two algorithms have been compared and contrasted. The second question has been answered by comparing quality of clusters generated by this algorithm to those generated by a quadratic algorithm, *wcd* with respect to sensitivity and specificity [Zhu *et al.* 2003]. Parameters that affect performance of the algorithm with respect to running time and quality of clusters have been experimentally deduced. These parameters include threshold value, overlap parameter, the minimum distance between pivots, the nature of data set being clustered and pivot selection techniques.

1.5 Results

Experimental results have shown that the running time of the algorithm is closer to linear than to the quadratic or logarithmic running times. Such a running time is a great achievement in EST clustering because huge data sets can be efficiently clustered in a shorter time. Mathematical analysis provides an upper bound on the running time and has shown that the algorithm has an $O(p \log p)$ complexity. The algorithm achieves 100% specificity but fluctuates in its sensitivity. This has been due to a major limitation arising from the technique employed by the algorithm. Depending on data sets and sensitivity required, the algorithm can be used for direct clustering of data sets or preclustering. Implementation of the triangle inequality increases running time of the algorithm because in experiments it held only about 2% of all distance computations performed by the algorithm.

1.6 Importance of the Research

Clustering separates mixed up, redundant and erroneous EST sequences derived from randomly selected mRNA into different groups representing unique expressed genes. The clustering process speeds up joining of ESTs into consensus sequences, which reveals complete expressed gene sequences. Joining EST sequences from the same cluster in different ways reveal alternative splicing of the same gene. Hence, EST clustering is useful for expressed gene identification, expression studies, SNP identification, micro-arrays design, genetic engineering, disease diagnostics and drug design [Kalyanaraman *et al.* 2003]. Gene identification helps in understanding gene sequences and corresponding functions of proteins produced. Highly expressed genes yield more mRNA from which ESTs are transcribed. Clustering such ESTs result into many clusters. Clustering is, therefore, useful in estimating gene expression levels [Burke *et al.* 1999]. In medicine this information helps in diagnosing diseases and prescribing drugs. Gene expression varies in different parts of the body. EST clustering assists in identifying genes that are being expressed in different parts of the body [Davison 2001]. Clustering helps in designing micro-array chips that are used in detecting gene expression levels [Kalyanaraman *et al.* 2003].

Through the developed algorithm, this research has contributed to the efficiency of EST clustering. Given the biological reasons above, the importance of efficient EST clustering cannot be underestimated. ESTs are transcribed in large numbers. There are about 8 million and 4.8 million ESTs for human and mouse respectively in the NCBI dbEST database [NCBI]. Hence, they need to be clustered as efficiently as possible with the available limited resources. Efficient clustering leads to quick gene identification, speeds up expression studies and SNP identification among many others. This consequently contributes to the efficiency of disease diagnostics, drug manufacturing and genetic engineering in agriculture. In general, efficient EST clustering can speed up several fields that rely on understanding and determining the number, type and locations of expressed genes. This is because information can be obtained within a quick and reasonable time and put into the required use.

1.7 Summary

ESTs are cDNA fragments, which are reverse transcribed from mRNA found in the cell cytoplasm of cellular organisms. EST sequences are mixed up, numerous, erroneous and redundant. This is a bigger challenge in efficiently generating EST consensus sequences to reveal underlying gene sequences. Clustering reduces the computational complexity of generating consensus sequences that represent complete gene sequences. Clustering assists in revealing the number and types of genes being expressed at a time. Knowledge of number and types of genes is useful for disease diagnostics, drug design, genetic engineering, gene expression studies, microarray design and single nucleotide polymorphism studies. Previous approaches are computationally expensive in clustering ESTs. This research aimed at finding an efficient algorithm for clustering ESTs with respect to running time, space utilisation and quality of generated clusters. A tree-structured index algorithm has been developed and implemented. Experimental results have shown that the running time is closer to linear with a 100% specificity but fluctuates in its sensitivity depending on the data set and parameters used. However, this running time is an improvement over quadratic running time.

For the purpose of clarity, it is important to highlight some of the terminologies and expressions that have been used. The tree-structured index algorithm developed in this research is called the *tsi_cluster algorithm*. It is also referred to as *the algorithm* or *the developed algorithm* unless stated otherwise. *Gene sequencing* is interchangeably used with *DNA sequencing*. However, DNA sequencing is much broader than gene sequencing. Assembling ESTs into consensus sequences generally means joining all sequences in a cluster into one complete representative sequence. The terms *genes* and *expressed genes* are interchangeably used, but this research is focused on expressed genes, which are genes that are active.

This dissertation is organised in seven chapters. Chapter one has introduced the research problem, its biological background, importance of the research, the developed algorithm and consequent research questions.

Chapter two discusses the biological background of the research problem, the EST clustering problem, prior work in various aspects of the algorithm and prior algorithms. The biological background explores the role of DNA in living organisms, transcription of DNA, translation of RNA, reverse transcription of mRNA and the production of ESTs. The chapter defines the EST clustering problem and it further explores clustering processes and techniques. Various techniques from prior work that have been employed in the algorithm such as: distance functions, the metric space and tree-structured indices have been explored in the chapter. Prior clustering algorithms such as d2_cluster, *wcd*, CLU and many others have been studied and discussed in the same chapter.

A description of the developed tree-structured index algorithm is given in chapter three. It covers the algorithm design, data structures used and functionality. The chapter further discusses research questions and how they are answered. Chapter four describes experiments that have been conducted to examine the performance of the algorithm with respect to running time, quality of generated clusters, space utilisation and the effect of the triangle inequality. Results of this research are presented in chapter five. Results are presented on running time, memory utilisation, quality of generated clusters and the effect of the triangle inequality. Chapter six discusses the major limitation of the algorithm that affects the quality of generated clusters. Major issues discussed in all the six chapters are summarised in chapter seven.

Chapter 2

Background and Related Research

2.1 Introduction

This chapter builds on the biological background of the research problem briefly explored in the previous chapter. The previous chapter covered the importance, methodology and results of the research. This research aimed at coming up with an EST clustering algorithm that runs in $O(p \log p)$, where n is the total number of sequences, m the average number of windows per sequence and $p = mn$ is the total number of windows in a data set. Such a running time is a great improvement over a quadratic running time of $O(p^2)$. The algorithm reduces running time by building a tree-structured index in a pseudometric space. The tree-structured index is constructed by recursively selecting two far apart sequence windows as pivots and partitioning the remaining sequence windows into two groups based on their proximity to either of the pivots. The recursive process terminates when a defined threshold point is reached or there are no more sequence windows.

This chapter further discusses the biological background of the research problem. It explores the research problem and prior work done in various fields applied in the algorithm. Understanding the biological complexity of the research problem will assist in appreciating implementation of the algorithm.

The biological background is covered in section 2.2 with focus on the role of DNA in living organisms as a blueprint of all their characteristics and functionality. The DNA transcription and RNA translation processes are covered in the same section. These processes are discussed to link the role of DNA to the functionality and structure of organisms. DNA functions are explored when discussing the production of proteins and their consequent roles and functions in various parts of the body. The need to understand the types and locations of DNA coding regions is covered under DNA sequencing. This is followed by a discussion on the generation of cDNA and expressed sequence tags. The section aims at placing the research problem in the biological context where it is applied.

The research problem is further discussed in section 2.3 where the EST clustering problem, processing of ESTs to remove errors incurred during the generation process are discussed. Different aspects of the algorithm such as distance functions, threshold distance value, clustering in metric space, pivot selection and tree-structured indices as applied in prior research are explored. All these components collectively work together to reduce distance computations and evaluations in the algorithm in order to improve running time and quality of generated clusters. Distance functions are used to measure the level of similarity among sequence windows. A threshold value is a cut-off point on the level of similarity among sequence windows. Edit distance and $d2$ are the two distance functions explored with the aim of justifying the distance function that has been finally implemented in the algorithm having examined both their strengths and weaknesses. The two issues are covered in section 2.4.

Sequence windows are modelled as objects distributed in a metric space. The developed algorithm is designed with the assumption that similar objects are distributed in metric space to form clusters. The algorithm finds these clusters by measuring relative distances among the objects to determine how close they are to each other. This subject is discussed in section 2.5. The section also covers metric space properties, importance and queries that determine levels of similarity. Under the same section, pivot selection and its effects on clustering are explored. Application of tree-structured indices with respect to prior work and clustering techniques are respectively covered in sections 2.6 and 2.7. Two examples of tree structures are explored with regard to their implementation and functionality. The tree structures are Sequence Search Tree (SST) as implemented by Giladi *et al.* [2002] and Burkhard-Keller Trees (BKT). Prior clustering algorithms are covered in section 2.7 with the aim of showing how they work, their possible limitations and relationship to the developed algorithm. A summary of the chapter is made in section 2.8.

2.2 Biological Background

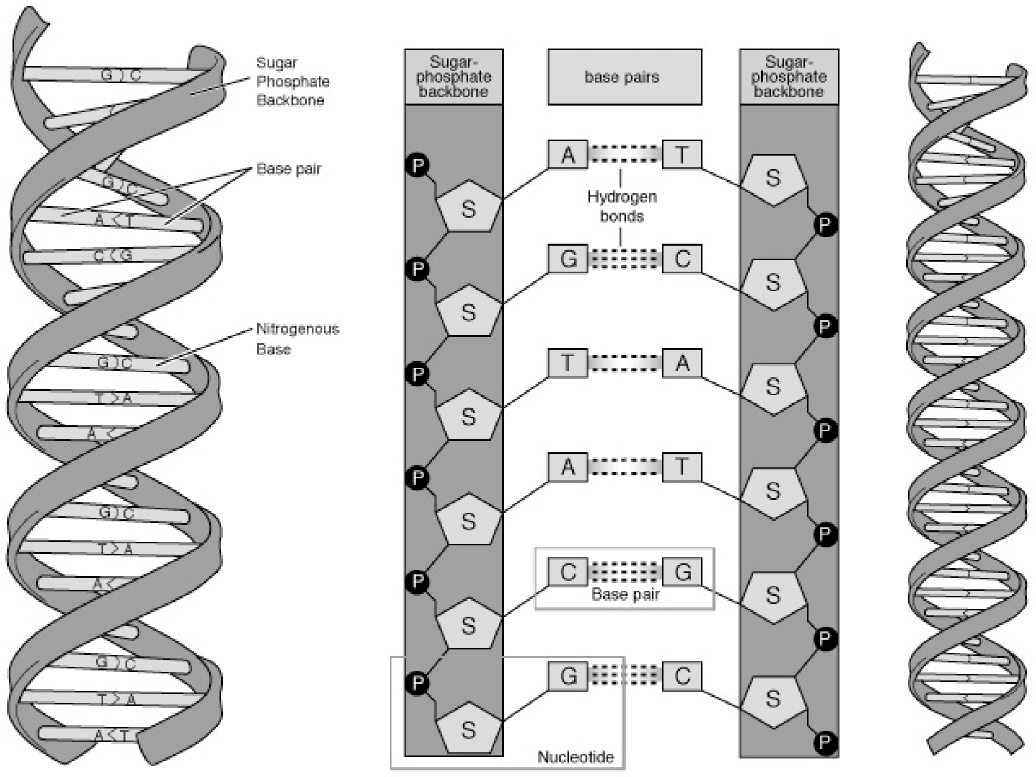
EST clustering is a biological problem but it has been solved by the tsi_cluster algorithm using computational techniques. It is important, therefore, to place the EST clustering research problem in its biological context in order to clarify sources of constraints and complexities of the problem. In order to provide this clarification, this section discusses DNA in general in subsection 2.2.1. Then it explores the relationship among DNA, expressed genes and ESTs by discussing intermediate processes and their resulting products. The transcription process discussed in subsection 2.2.2 shows relationships among DNA, expressed genes, RNA, mRNA, mature mRNA, proteins, ESTs and complementary DNA. The sections begins by provided the biological overview in living things with focus on the role of DNA.

The diversity and complexity of cell structure, composition and functionality of living organisms in both multicellular and unicellular organisms is determined by their DNA. In multicellular organisms, DNA is found in the cell nucleus and special structures in the cell known as mitochondria. In unicellular organisms such as bacteria, DNA is found in the cell cytoplasm [Hunter 1991]. Special locations in DNA known as genes encode useful information about the physical appearance, structure and functionality of an organism. Each gene is responsible for the production of a specific protein in addition to other products. Proteins are used for specific tasks in the body. Although the number, locations and functions of genes remain a mystery in most organisms, it is important and worth uncovering. This is useful for reasons outlined such as disease diagnostics, drugs design and genetic engineering. In general, if a gene responsible for a specific disease is known, the disease may be cured by suppressing expression of the responsible gene, thereby controlling its effects. It should be noted however, that some diseases are genetically caused by other reasons such as: damaged genes, expression failure or over-expression. The following subsection briefly describes DNA with respect to its structure, composition and functionality.

2.2.1 DNA

DNA is the root source of ESTs. ESTs are generated from cDNA, which is reverse transcribed from mature mRNA, a direct expressed gene transcript encoded in a DNA strand. Hence, understanding DNA composition, structure and functionality assist in comprehending the complexity of the EST clustering problem. DNA is found in the cell nucleus and mitochondria of multicellular organisms. In unicellular organisms, it is found in the cell cytoplasm. DNA comprise four nucleic acid molecules known as nucleotides, which are adenine (A), guanine (G), cytosine (C) and thymine (T). These nucleotides are joined together to form a sequence [Cohen 2004]. For example; ACCCCGGTAGCGCTTTTAAAA is a sequence. Nucleotides are also known as bases. A sequence consists of coding and non-coding regions respectively known as exons and introns. Adjacent nucleotides in a DNA strand are joined by a sugar phosphate backbone.

DNA structure can be described as a twisted ladder where the rungs can be seen as joint complements of the nucleotides (base pairs). Exclusive bonds are made between adenine and



(a) DNA (b) Detailed DNA

Figure 2.1: DNA at different levels of detail (figure from [National Health Museum 2006])

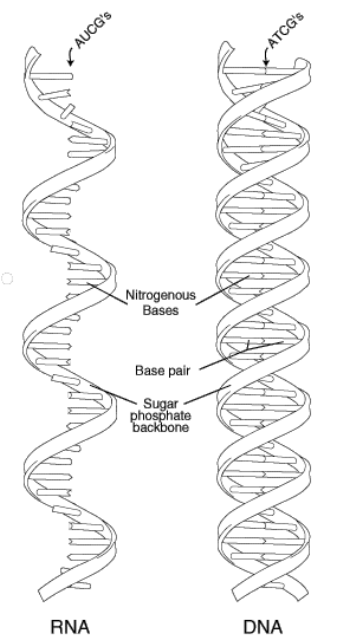
thymine, and between guanine and cytosine. DNA sequences are measured in thousands of bases (kb). It is double stranded with one strand running in one direction (from $5'$ to the $3'$ DNA end) and another complementary strand running in a reverse direction (from the $3'$ to $5'$ end). For example, if the first strand running in one direction is *AGGTCAC*, its complementary strand would be *TCCAGTC*. Figures 2.1(a) and 2.1(b) show DNA at two different levels of detail. DNA is transcribed into RNA, which is then translated into proteins. The process is covered in the subsection 2.2.2

2.2.2 DNA Transcription

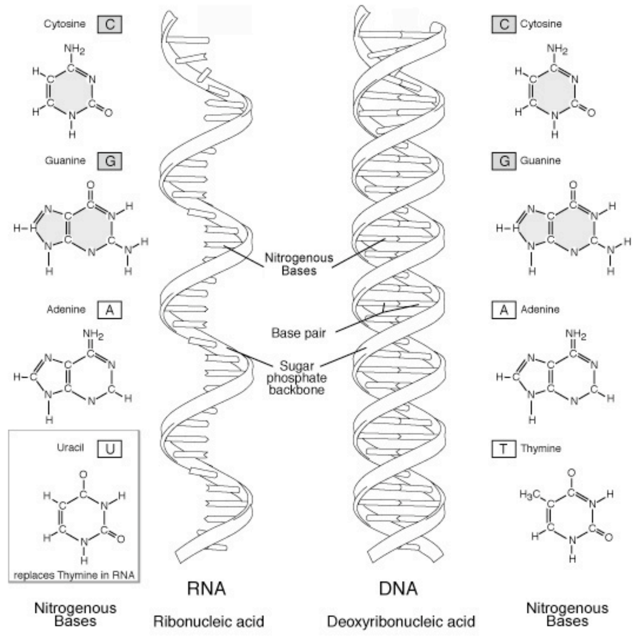
The transcription process is discussed to show the relationship that exists between expressed genes and mature mRNA, from which cDNA is reverse transcribed. The research problem is to group these cDNA fragments (ESTs) to reveal expressed gene sequences.

The transcription process begins at the promoter region in DNA and continues through the coding region ending at the terminator region. Transcription factors bind around promoter locations on the $5'$ side of DNA and RNA polymerase binds to these transcription factors, thereby opening the DNA double strand. The RNA polymerase continue the process while generating a single stranded premature mRNA (pre-mRNA) moving from $3'$ DNA side to $5'$. This is accomplished by using basic nucleotide pairing rules. For example, a *C* nucleotide encounter on a *DNA* strand results in a *G* insert on a pre-mRNA strand and an *A* encounter results in a *T* insert. Pre-mRNA is single stranded and is composed of exons and introns. Introns are spliced out from pre-mRNA to form mature RNA [Kalyanaraman *et al.* 2003]. This process takes place in the cell nucleus from which mRNA is transported outside the nucleus into cytoplasm.

When DNA is transcribed into mRNA, thymine (T) is replaced by uracil (U). For example, *AGGTCAC* transcribes into *UCCAGUG*. Non-overlapping triplets of these nucleotides form codons, which correspond to a particular amino acid. Figures 2.2(a) and 2.2(b) show the structure and composition of RNA. The next step is to translate mature mRNA in the cell cytoplasm into proteins and the process is discussed in subsection 2.2.3.



(a) RNA



(b) Detailed RNA

Figure 2.2: RNA at different levels of detail (figure from [National Health Museum 2006])

2.2.3 Mature mRNA Translation

Mature mRNA either is translated into proteins or reverse transcribed into complementary DNA. The translation process results in proteins through which genes carry out different functions they encode. This section focuses on the translation process. Both the process and the resulting protein structures are so complex such that it is easier to study and understand genes at an earlier stage of cDNA than at the protein level.

Mature mRNA is transferred outside the cell nucleus into the cell cytoplasm where proteins are produced in cell structures known as ribosomes. Mature mRNA codons are mapped into corresponding amino acids by transfer RNA (tRNA) forming peptide bonds. Translation of Mature mRNA to form proteins is achieved through peptide bonding of mature mRNA and amino acids using tRNA. Proteins are formed from twenty different amino acids and they fold into a three-dimensional structure as they come out of ribosomes. Their unique three-dimensional structure determines their specific biological role and the way they interact with chemical compounds. Chemical groups bind to different locations of a protein through a post-translational modification process. Binding of these chemical compounds further affect the behaviour and functions of proteins [Yona 1999].

Proteins provide structural support, act as catalysts for chemical reactions, regulate genes, provide mechanisms for acquiring and transferring energy and distinguishes body from what is foreign in the immune system [Hunter 1991]. Figure 2.3 illustrates processes of DNA transcription, exon splicing and mRNA translation. Since DNA is a blueprint for protein production in the body, it is important to know and understand gene sequences in order to identify proteins that are produced. Attempts to know nucleotide order in DNA, have led to DNA sequencing discussed in subsection 2.2.4.

2.2.4 DNA Sequencing

Gene sequences is very important in predicting the type and function of the resulting proteins. Techniques used in DNA sequencing broadly apply in cDNA sequencing as well which results in generation of EST. This subsection explores DNA sequencing and problems that are encountered in the process. Understanding DNA sequencing is important in order to appreciate sources of errors and redundancy in ESTs, which contributes to the complexity of the research problem.

DNA sequencing refers to the process by which the precise order of nucleotides in a strand is known. Available techniques are limited to sequence smaller pieces of DNA. To determine a complete DNA or gene strand, several overlapping pieces of DNA are sequenced. To accomplish

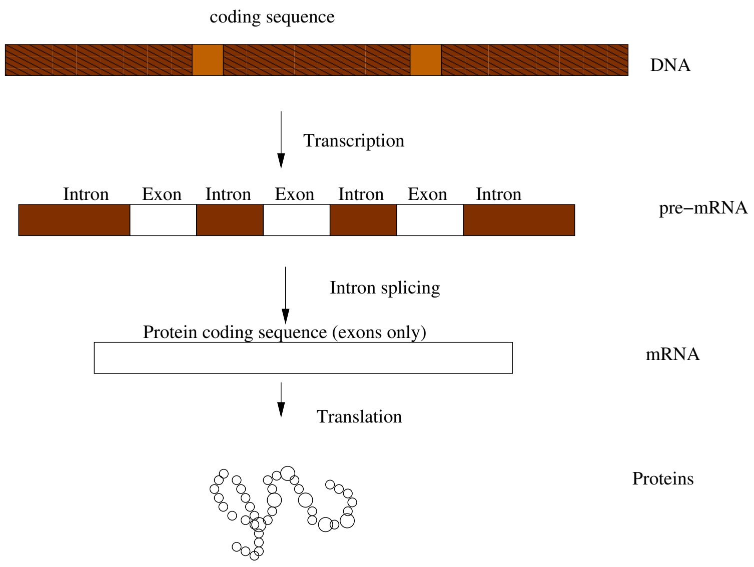


Figure 2.3: DNA transcription, splicing and translation process

this, a long strand of DNA is divided into smaller manageable pieces whose order is known. These pieces are further divided into much smaller overlapping pieces, which are then replicated.

There are several DNA replication techniques. One of the techniques uses vectors such as cosmids. A cosmid is a type of bacteriophage, which is a virus that affects bacteria. It is able to accept DNA base pair inserts. As the cosmid replicates, it duplicates the inserts DNA [Hunter 1991]. This is done to maintain and reproduce several DNA copies in order to have several pieces to work with. Another technique for replicating DNA is Polymerase Chain Reaction (PCR). In this technique, primers bind to a DNA sequence and they sequentially generate a DNA transcript, which is complementary to the original DNA strand.

Gel electrophoresis is a technique used to determine nucleotides order in the transcript. DNA transcripts resulting from cosmid replication or PCR are placed on a gel. Different sizes of nucleotides affect the distance that they move under the gel when subjected to an electric charge. When the gel is scanned, labels of the nucleotide movement and size are identified forming a DNA sequence [Malde 2005].

It is difficult to directly identify a complete gene sequence from short pieces of DNA because DNA sequences comprises both exons and introns. However, cDNA, which is mature mRNA reverse transcript consists of exons only. cDNA, therefore, provides the required complete gene sequence. Section 2.2.5 explores the reverse transcription process through which cDNA is generated.

2.2.5 Generating cDNA and Expressed Sequence Tags

One important feature of mature mRNA is that it comprises coding regions only (exons). Hence, it codes for complete expressed gene sequences. Identification of mature mRNA in the cell cytoplasm is, therefore, important in determining the number and types of expressed genes. The problem with sequencing mature mRNA is that it is single stranded, which causes it to be unstable for experimental purposes. The problem as reviewed by Hide *et al.* [1999] is solved through reverse transcription of mature mRNA to generate cDNA. cDNA is double stranded and therefore, stable for experimental purposes. Generating a full-length cDNA sequence at once is time consuming, expensive and technically difficult because of the limited capability of available techniques. The reverse transcription process starts and ends at random positions in the mature mRNA strand. This results in overlapping pieces of cDNA. The cDNA is cloned and placed in libraries from which ESTs are sequenced by single-pass reads. The length of ESTs range from 300 to 500 base pairs. When the pieces are assembled and joined together into consensus sequences, they reveal the underlying complete gene sequence.

ESTs are erroneous because they are generated from unedited, single pass reads of selected cDNA. Some of the errors incurred during the generation process cause compression and base scaling in the frame sheets. This causes errors in clone orientation, associated clone ID and missing $3^{'}$ and $5^{'}$ [Ptitsyn 2000]. In spite of these errors, EST data remains a rich and readily available information source for identifying complete expressed gene sequences. Hence, given efficient techniques and appropriate tools, ESTs can be efficiently used for identifying expressed genes. EST clustering is useful for gene prediction, gene discovery and genomic mapping [Hide *et al.* 1999; Adams *et al.* 1991]. When clustered, ESTs provide useful information on particular gene. The number of clusters generated represents the number and type of genes found from the given EST data set [Davison 2001]. Clusters are assembled and processed to form consensus sequences, which represent complete expressed gene sequences and their possible alternative splices.

2.3 EST Clustering

A detailed biological background of the research problem has been provided in the previous subsection. The relationship that exists between ESTs and respectively DNA, RNA, mRNA, mature mRNA, cDNA and proteins has been explored. Biological processes that take place have been discussed in order to show how they affect ESTs and eventually the EST clustering problem. Having laid the biological foundation, this section proceeds to define the EST clustering problem.

ESTs are partial sequences of cDNA, which are experimentally derived from mature mRNA in the cell cytoplasm of an organism. Mature mRNA in the cell cytoplasm represents genes expressed at a particular point in time in an organism. Hence, EST clustering is important not only because it results in discovery of complete gene sequences but it also reveals genes that are active at a specific point in time [Malde 2005]. Such information is useful in determining types of disease causing genes. Knowledge of such genes is used in designing and manufacturing drugs. ESTs, therefore, encapsulate very useful information that needs to be extracted efficiently by clustering and generating consensus sequences from which the number of expressed genes, their type and complete sequences are deduced. EST clustering is also important in understanding alternative splicing, a process in which a single gene transcribes into two or more mature mRNA. The different mature mRNA translate into different proteins, which carry out different functions in the body. By understanding alternative splicing these proteins and their respective functions are mapped to the responsible expressed gene. ESTs generated from alternatively spliced mature mRNA are grouped into the same cluster. They are joined together in different ways representing alternative splicing of the same gene.

2.3.1 What is EST Clustering?

EST clustering is the grouping of ESTs using overlaps that exist among them such that those that are from the same expressed gene are placed in one distinct cluster [Kalyanaraman *et al.* 2005]. The main challenge, however, is that ESTs are numerous, erroneous and redundant, which causes the clustering problem to be more difficult and computationally expensive. The problem is abstracted as a graph problem where vertices represent EST sequences and edges are distances among them [Ranchod and Hazelhust 2005]. The EST clustering problem is, therefore, to find components of a connected graph. Overlaps among sequences are determined by a distance function that measures the level of similarity. Two sequences overlap if the distance between them is less or equal to a defined distance threshold.

2.3.2 Processing ESTs

Since ESTs are generated by single-pass reads, they are processed before clustering to reduce errors. Pre-processing involves screening out lower quality regions, contaminations, vector sequences, repeats and lower complexity sequences. This reduces overall noise and improves efficacy [Carpenter *et al.* 2002]. It minimises chances of clustering unrelated sequences. Vector fragments and contaminations remaining in sequences negatively affect clustering results. These are, therefore, removed to improve clustering quality. Contaminations are external sources of DNA nucleotides that get attached to sequences during cDNA cloning.

A bigger part of multicellular DNA comprise repeats and only a small proportion encodes for genes [Adams *et al.* 1991]. Low complexity areas such as *Poly A tracks* and *AT repeats* are masked out because they cause errors during clustering. Masking involves replacing the repeats with special characters that are ignored during alignment. Softwares such as: RepeatMasker, VecScreen and Lucy are used in this stage [Malde 2005]. A quality threshold and minimum length is defined for sequences to be acceptable.

2.4 Distance Functions

Distance functions are used to determine the level of similarity among sequences based on which sequences are clustered. This section explores the functionality and implementation of the edit distance and d2 distance functions. The nature of the problem being solved determines the type of distance to be used. Focus has been placed on the edit distance and d2 distance functions because they have been implemented in the algorithm. However, d2 has been found to be more suitable for the algorithm and is implemented in the final version. The discussion begins by exploring in general the relevance of applying distance functions on ESTs.

ESTs are represented as strings over an alphabet $\Sigma = \{A, C, G, T\}$ as shown in Figure 2.4. Finding the level of similarity among EST sequences is, therefore, a string matching problem. A string S is an ordered list of characters written contiguously from left to right. For any string S, $S[i \dots j]$ is a substring of S starting at position i and ending at j, where $i < j$. $S[1 \dots i]$ is a prefix ending at i and $S[i \dots |S|]$ is a suffix and $|S|$ is the length of S. The words *string* and *sequence* are synonymously used in this research. The string matching problem is also known as the pattern matching problem and there are two types. These are exact and approximate string matching. Exact string matching finds a string pattern within another string or body of text. Formally, given a pattern string $P \in \Sigma^*$, with length $|P| = m$ and text string $T \in \Sigma^*$, with length $|T| = n$, where $0 < m \leq n$ and Σ^* is the unordered concatenation of the alphabet Σ. In exact string matching, character mismatches are not allowed in the string while in approximate string matching they are allowed. These mismatches could be due to misspellings in real world errors. In ESTs, they are caused by either errors during EST generation or may be due to real biological causes such as mutation. The type of errors that are of concern in this research are those that occur during EST generation and sequencing as covered in section 2.3.2. Since ESTs are erroneous, approximate string matching is more ideal in finding the level of similarity as opposed to exact string matching [Hazelhurst 2004]. The minimum number of changes made on two unequal strings to make them similar defines approximate string matching. The number of changes is known as edit distance and it is discussed in detail in section 2.4.1.

```
>T27784 g609882 | T27784 CLONE_LIB: Human Endothelial cells.
 LEN:337 b.p. FILE gbest3_seq 5-PRIME DEFN:EST16067
 Homo sapiens cDNA 5′ end
 AAGACCCCCGTCTCTTTAAAATATATATTTTAAATATACTTAAATATATATTT
 CTAATATCTTTAAATATATATATATATTTNAAAGACCAATTTATGGGAGANTTGC
 ACACAGATGTGAAATGAATGTAATCTAATAGANGCCTAATCAGCCCACCATGTTC
 TCCACTGAAAAATCCTCTTTCTTTGGGGTTTTTCTTCTTTCTTTTTGATTTTGCAC
 TGGACGGTGACGTCAGCCATGTACAGGATCCACAGGGGTGGTGTCAAATGCTATT
 GAAATTNTGTTGAATTGTATACTTTTTCACTTTTTGATAATTAACCATGTAAAAAA
 TG
```

Figure 2.4: An EST sequence in Fasta format

Two strings are compared at a time by pairwise alignment. The strings are placed one above the other such that for every character there is an opposite unique character or a gap. There are two types of alignment: global and local alignment. In global alignment, the whole string is aligned and compared against the other. Local alignment, however, aims at finding regions or substrings that are approximately similar. The length and number of similar local regions or substrings found between the two strings determine the similarity score.

Distance functions and threshold values are required in clustering. A distance function determines the level of similarity among sequences. A threshold value defines a distance cut-off point below which two EST sequences are considered similar. Distance computation is done on overlapping windows and not on the entire sequence. The aim is to capture regions of local similarity between two sequences, which indicate that the sequences overlap and are, therefore, from the same gene.

2.4.1 Edit Distance

Edit distance is the minimum total number of edit operations made on one string to transform it into another string [Gusfield 1997]. Edit operations performed on a string are deletions, insertions and substitutions. In general, edit operations are made on both strings since an operation on one string translates into another operation on the second string. For example, an insertion on one string is considered as a deletion on the other. Edit distance is, therefore, generally defined as the minimum total number of edit operations made on two strings to make them similar. It is a direct product of global alignment as previously discussed. The two strings are aligned one above the other with spaces inserted such that every character or space in both strings is opposite with a character or space in the other string. For example, given two strings S_1 and S_2, where $S_1[1...i]$ = ACGTTGCTTAAA and $S_2[1...j]$ = AGTTTTGACAT. They could be aligned as follows:

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

The edit distance D to character i in nonempty substring $S_1[1..i]$ and character j of nonempty substring $S_2[1..j]$ is denoted $D(i, j)$. Dynamic programming is a widely known technique for computing edit distance [Gusfield 1997]. It has three main components in its computation, which are the recurrence relation, the tabular computation and the trace back operation.

Recurrence Relation

The recurrence relation has two base conditions, which apply when either of the strings are empty.

Base operations

- $D(i, 0) = i$, when string j is empty
- $D(0, j) = j$, when string i is empty

The recurrence relation

- $D(i, j) = min[D(i-1, j) + 1, D(i, j-1) + 1, D(i-1, j-1) + t(t, j)]$, where $t(i, j)$ has a value of 1 when there is a mismatch and 0 otherwise.

Proof of correctness of this equation is beyond the scope of this discussion. However, the recurrence equations cover all possible edit operations in the two strings as follows:

- An edit operation can be a deletion in either i or j, which is represented as $D(i-1, j) + 1$ and $D(i, j-1) + 1$ respectively. A deletion on one string can be considered as an insertion in another string. Assuming that the last symbol to be edited (inserted) is $S_2(j)$ in order to transform $S_2[1..j]$ into $S_1[1..i]$, then symbols before $S_2(j)$ must specify the minimum number of edit operations to transform $S_1(1...i)$ to $S_2(1..j-1)$. This takes $D(i, j-1)$ edit operations plus the last one for $S_2(j)$, which is $D(i, j-1) + 1$.

- Assuming that the last symbol to be edited (deleted) is $S_1(i)$ in order to transform $S_2[1..j]$ into $S_1[1..i]$, then symbols before $S_1(i)$ specify the minimum number of edit operations to transform $S_2(1...j)$ to $S_1(1..i-1)$. This takes $D(i-1,j)$ edit operations plus the last one for $S_1(i)$, which is $D(i-1,j)+1$.

- A match or mismatch is represented by $D(i-1,j-1)$ plus a penalty of $t(i,j)$. If there is a match then character $S_1(i)$ will have to be replaced by character $S_2(j)$ and the symbols to the left of i and j will specify the minimum number of edit operations to be done to transform $S_1[1..i-1]$ to $S_2[1..j-1]$. Hence this yields $D(i-1,j-1)$ edit operations plus one last edit operation, which becomes $D(i,j) = D(i-1,j-1)+1$.

- When there is a match, $S_1(i) = S_2(j)$, the cost on the last edit operation is zero. Hence, a variable $t(i,j)$ is introduced for both the match and mismatch. When there is a match $t(i,j) = 0$ and $t(i,j) = 1$ when there is a mismatch. Hence, the two cases are combined into one giving $D(i,j) = D(i-1,j-1) + t(t,j)$

Relationship between a pair of strings can be established by using alignment. Strings, which are closer to each other, yield a smaller number of edit operations as compared to those, which are far apart. For mathematical precision, costs are associated with the edit operations and the total number of edit operations defines the distance between a pair of strings.

Tabular Computation and the Trace Back Operation

The recurrence relations and base conditions are easily coded as a recursive program using any programming language. A procedure is called with m, n as input to give the correct answer. Although this top down recursive approach is simple to implement, it is highly inefficient for larger values of n and m because of numerous redundant recursive calls, which are made and grow exponentially.

Hence, the tabular computation uses the recurrence relation to efficiently compute edit distance. An $n \times m$ matrix is built with the two strings, one running horizontally and the other vertically. Characters in the horizontal and vertical strings represent rows and columns respectively. By using a bottom-up approach, the matrix is filled with edit values in all position (i,j) in a most efficient way [Gusfield 1997]. As the matrix is being computed, pointers that link cells with a minimum number of edit operations are established. The minimal path of edit operations is established by tracing back from cell (n,m) to cell $(0,0)$. Computing values in the matrix takes a time complexity of $O(nm)$ because each cell takes a constant amount of time and there are nm cells in total. The trace back operation takes a time complexity of $O(n+m)$. The overall complexity of the algorithm in the worst case, therefore, is $O(mn)$.

The discussion on distance functions proceeds in the sections that follow with focus on the d^2 distance function. This will help in comparing and contrasting the two distance functions so that the distance function that is finally implemented in the algorithm can be justified.

2.4.2 The d^2 Distance Function

The d^2 is a distance function that determines the level of similarity between two strings by searching for a specified number of similar words over a given window size [Burke *et al.* 1999]. It is pronounced and referred to as d2 distance function. It is applied in many clustering algorithms such as d2_Cluster, CLU algorithms, *wcd* and many others. The distance is found by computing the sum of the squares of the differences of word frequencies over a given window. The following notation is used to formally define the d2 distance function [Hazelhurst 2003].

$$d_k^2 = \sum_{|w|=k}(c_{s_x}(w) - c_{s_y}(w))^2$$

Where s_x and s_y are sequences, w is a word, k is the length of w, $c_{s_x}(w)$ and $c_{s_y}(w)$ are the number of times w appears in s_x and s_y respectively. The difference in the number of times a word, w appears in s_x and s_y is squared. The squared values are summed for all occurring words and this is the d2 distance value between the two strings. Comparison between two sequences is done over a window of specified length. The idea is to find existence of highly similar windows between two sequences. Hence, the d2 distance between two sequences is the minimum score among all pairs of windows. Generally, a threshold is defined and all sequences below the given threshold are considered to be in the same cluster.

In previous work, windows of size 100 and words of length 6 have been used and the same sizes have been used in the developed algorithm. The d2 distance function has been successfully applied in d2_cluster and *wcd* algorithms by Burke *et al.* [1999] and Hazelhurst [2004] respectively. The size of the window, the length of the word and threshold value determines the performance of algorithms. The best performance of such algorithm could be found by incrementing the values until the optimal point is reached.

There are two main problems in implementing the d2 distance function in the algorithm. The distance function unfortunately does not satisfy all metric space properties and it is expensive to compute because it has a quadratic complexity. However, the square root of d2 obeys all metric space properties except the strict positiveness property discussed in section 2.5. The square root of d2, therefore, defines a pseudometric space and it is implemented in the algorithm. The d2 distance function has a quadratic complexity when it searches for a pair of windows with a minimum score between two sequences being compared. However, computing d2 on a pair of windows globally takes linear time, which solves the second limitation of using the distance function.

Edit distance also obeys metric space properties, which makes it more suitable for implementation in the algorithm. However, it has a quadratic complexity, which creates a bottleneck in the running time of the algorithm. As a result, it was found reasonable to implement d2 distance function in the algorithm. A threshold value is placed on distance as a cut-off point on the level of similarity. This is further explored in the next section.

2.4.3 Threshold Distance Value

A threshold distance value determines a point from which two sequences S_x and S_y are considered similar to each other [Zimmermann 2003]. The value also defines a bound on the level and depth of clustering that is achieved. The quality of clusters generated are greatly affected by the threshold value. Hence, problems arise when a threshold value is not properly defined. The threshold value requires to be carefully chosen through experimentation to establish an appropriate and reasonable value that does not lead to underclustering or overclustering. Prior work has shown that bigger threshold values lead to overclustering. In this case, the clustering process stops before fully clustering sequences due to insufficient stringency. A smaller threshold value leads to underclustering where the clustering process continues after allocating sequences into their appropriate clusters because of excessive stringency [Ptitsyn and Hide 2005]. Thresholds are used in similarity functions and clustering algorithms. BLAST uses thresholds in order to determine similarity among sequences [Cameron *et al.* 2004]. The d2_cluster algorithm developed by Davison [2001] and the Sequence Search Tree algorithm developed by Giladi *et al.* [2002] among other examples, use thresholds in determining cut-off points on the level of similarity among sequences.

The choice of threshold value depends on the nature of the problem being solved. For example in exact string matching, the need for a threshold value is irrelevant because the two strings being compared have to completely match. The issue of the threshold value arises in approximate string matching where the level of similarity varies.

The choice of distance functions and their respective thresholds in this research is made with consideration to their complexity and whether they obey metric space properties. The metric space, its properties are discussion in the next section.

2.5 Clustering in Metric Space

Research work done by Chavez *et al.* [2001] and Giladi *et al.* [2002] has shown that it is possible use the metric space for searching. Searching in metric space has been successfully implemented and achieved by Giladi *et al.* [2002] in the Sequence Search Tree algorithm. In his work, Chavez *et al.* [2001] proposed ideas that could successfully implemented for searching. This research, however, implements the metric space for clustering based on ideas from prior work. This section discuses metric space properties and how the tsi_cluster algorithm applies them in solving the EST clustering problem. Clustering in metric space has been implemented to reduce the number of distance computations and evaluations made during clustering and consequently reduce running time from quadratic to logarithmic running time.

A metric space $(\mathbf{X}, d)$ is formally defined as a set of objects $\mathbf{X}$ whose similarity is modelled by a distance function d that satisfies its properties [Chavez *et al.* 2001]. Distances among objects are usually expensive to compute and the metric space approach reduces the number of distance computations and evaluations by constructing a tree-structured index structure. Pivot selection and compact partitioning are the two techniques used in constructing tree-structured indices [Chavez *et al.* 2001]. In the tsi_cluster algorithm, the two techniques work by choosing two sequence windows as pivots in a data set and recursively partitioning the remaining sequence windows into two groups depending on their proximity to either of the selected pivots until a threshold point is reached or there are no more windows. Partitioning enables pivots to be compared to sequence windows within their partition only instead of all windows in a data set, which also reduces distance computations made.

This section formally introduces the metric space, its notations and the four basic properties as outlined by Chavez *et al.* [2001] and Bustos *et al.* [2003]. The universe of valid objects is denoted $\mathbf{X}$, a finite subset of $\mathbf{U} \in \mathbf{X}$ is known as a database or dictionary from which query operations are made. The words *database* and *data set* are interchangeably used in this research. The size of the database is denoted $n = |\mathbf{U}|$. The measure of distance between two objects is denoted $d(s_x, s_y)$, where $s_x, s_y \in \mathbf{U}$ and the following properties apply in metric space.

- positiveness: $\forall s_x, s_y \in \mathbf{X}, d(s_x, s_y) \geq 0$.

  This property enforces that given two windows x and y, the distance between them must be strictly positive or equal to zero if the windows are equal, hence prohibiting any negative values in the distance measure.

- symmetry: $\forall s_x, s_y \in \mathbf{X}, d(s_x, s_y) = d(s_y, s_x)$.

The distance from s_x to s_y and vice versa is equal.

- reflexive: $\forall s_x \in \mathbf{X}, d(s_x, s_x) = 0$.

  This property enforces the fact that the distance from a window to itself has to be zero.

- strict positiveness: $\forall s_x, s_y \in \mathbf{X}, s_x \neq s_y \Rightarrow d(s_x, s_y) > 0$.

  This property ensures the fact that for any two given windows, s_x and s_y, the distance between them should be strictly greater than zero. When this property does not hold, the space is known as a pseudometric space. The square root of d2 distance function does not obey this property and it, therefore, defines a pseudometric space. The developed algorithm consequently works in a pseudometric space because it implements the d2 distance function.

In general the metric space, $(\mathbf{X}, d)$ has to satisfy the triangle inequality property:

- $\forall s_x, s_y, s_z \in \mathbf{X}, d(s_x, s_y) \leq d(s_x, s_z) + d(s_z, s_y)$.

  The triangle inequality property is central in metric space clustering implemented in the algorithm. The property states that the distance between the two given sequence windows (pivots) s_x and s_y is less or equal to the sum of the distances from the two windows s_x and s_y to the third window, s_z.

Metric space properties are useful in handling three types of proximity queries: range queries, nearest-neighbour queries and k-nearest neighbour queries. These queries are used in determining the proximity of sequence windows to pivots in order to partition a data set. Hence, understanding of these queries is necessary in order to comprehend how the developed algorithm works.

Range Queries

The range query $(q, r)_d$ retrieves all elements $u \in \mathbf{U}$ within radius r from the query point q. It retrieves all database elements $u \in \mathbf{U}$ within a radius of r from the query window q.

Near Neighbour Queries

The nearest neighbour query retrieves all elements $u \in \mathbf{U}$ closest to query q. The query is built on top of the range queries but it varies the search radius until the desired number of elements found. There are different variants of the near neighbour query such as: the increasing radius, decreasing radius and priority backtracking [Chavez *et al.* 2001]. The increasing radius searches for elements starting with the initial radius r and then increments it until the required number of elements is found. The decreasing radius does the opposite. It starts from infinity and decreases the radius until the required number of elements is found. In priority backtracking, a lower bound on distance from pivot q to elements is known. When transversing the tree looking for such elements, roots of subtrees are compared with the pivot q and subtrees with distances closer to the lower bound are traversed first. Backtracking is done when subtrees with a greater distance from pivot q is found. Hence, priority is made during transversing the search tree.

k-Nearest Neighbour Queries

The k-nearest neighbour query retrieves k-closest elements $u \in \mathbf{U}$ to a query element q. The query is a variant of the near neighbour query only that it provides a bound on the number of elements that require to be retrieved.

Importance of the Metric Space in the algorithm

The developed algorithm relies on these metric properties and range queries to work. The properties are required to apply on the distance function that defines similarity among sequence windows. This enables meaningful comparison of sequences based on which the space is partitioned. Queries discussed above apply only if the distance function used on sequence windows obeys metric space properties. This is because application of these queries depends on the relative distance between pairs of sequence windows in metric space, which enable identification of positions of sequences with respect to each other. For the purpose of this research, range queries are used firstly, to identify pivots and secondly to find the proximity of sequence windows from the pivots based on which they are partitioned. The resulting clusters comprise sequences windows that lie within a specific radius, which is a predefined threshold value. Proper choice of pivots is crucial in order to come up with meaningful clusters. This is the focus of the next subsection.

2.6 Tree-Structured Indices

The previous section discussed the metric space, its properties and queries that can be applied. This section proceeds to discuss usage of pivots and partitioning techniques in building tree-structured indices. The section explores a number of tree-structured indices with respect to their construction, functionality and possible strengths and weaknesses so that a suitable tree-structure index implemented in the algorithm can be appreciated. Pivoting and compact partitioning are the two main techniques used in building tree-structured indices in a metric space [Chavez *et al.* 2001]. Compact partitioning is the process by which the metric space is divided into smaller regions with the aim of narrowing down the problem space. Pivots are used in the compact partitioning process as decision making points for dividing the metric space. The tree-structured index algorithm employs these two techniques in clustering ESTs.

A tree structure is useful in reducing the number of distance computations and evaluations performed during clustering. Distance computations are reduced by recursively dividing a data set into two partitions, thereby avoiding comparison of each sequence window to the rest of the data set at each step. Instead, a window is only compared to the two pivots chosen in a partition. Hence, at each step the partition reduces by half. This technique is central in improving efficiency of the algorithm. There are different variations of tree structures depending on the number of pivots used in constructing the tree. In the tsi_cluster algorithm, two pivots are used per recursive call. Examples of tree-structures are discussed as applied in various algorithms and presented in prior work. Two types of tree-structured indices are explored. The first group works on discrete distance functions and the second one works on continuous distance functions. The focus in this discussion is placed on discrete distance function, in particular the Sequence Search Tree and Burkard-Keller Tree. Their approach is similar to the one applied in the tsi_cluster algorithm. Pivot selection is explored in the next section before discussing the tree structures so that its implementation can be compared and contrasted in tree structures.

2.6.1 Pivot Selection

The developed algorithm uses pivots to locate windows in a pseudometric space based on which the data set is partitioned. The challenge is to choose pivots that facilitate efficient clustering. Pivot selection heavily affects performance of an algorithm. Properly chosen pivots greatly reduce the search space, which is useful in constructing a tree-structured index [Bustos *et al.* 2003; Chavez *et al.* 2001]. The search space is reduced by recursively dividing it into smaller partitions at each run. This is achieved by grouping sequence windows closer to the first pivot, p_1 together, and those close to the second pivot, p_2. The process recursively continues until a threshold point is reached or there are no more windows to partition. Depending on how pivots are selected, the algorithm produces a balanced or unbalanced tree. The desirable result is to have a balanced tree, which is more likely achieved when far apart pivots are selected.

In general, there are two factors that affect performance of an algorithm because pivot selection and these are: pivot selection criteria and the number of pivots used [Bustos *et al.* 2003]. The larger the number of pivots, the faster the clustering process. This is because the search space becomes much smaller at each step as the number of pivots increase. The developed algorithm uses two far apart pivots in an attempt to generate a balanced tree, which is crucial for achieving efficient running time.

There are a number of heuristics that are used to choose pivots. Heuristics are explored in subsections that follow in attempt to gain more understanding on pivot selecting techniques from which improvements or modifications are applied in the algorithm. Most heuristics work in metric space and they select far apart pivots [Bustos *et al.* 2003; Chavez *et al.* 2001]. Bustos *et al.* [2003] developed an efficient searching algorithm based on pivots and three pivot selection heuristics employed are explored with the aim of understanding their strengths and weaknesses that are taken into consideration when developing a pivot selection technique in the tree-structured index algorithm.

N-Random Groups Selection

The first pivot selection heuristic is the N-random groups selection. In this technique, N groups of k pivots are randomly selected from database objects, **U**. Then, a mean μ_D of distance distribution in D is calculated for each group and a group with maximum μ_D is selected. An estimation of a value of μ_D is obtained as follows. Pairs of objects are randomly chosen from **U**, $\{(a_1, a_1^{'}), (a_2, a_2^{'}), ..., (a_A, a_A^{'})\}$ result into a set of distances $D_1, D_2, ..., D_A$, where A is the total number of pairs in a group. Hence, an estimated value of $\mu_D = \frac{1}{A}\sum_{1\leq i\leq A} D_i$. This is an average distance of sum of distances between pairs divided by the number of pairs, A. Hence, $2k$ distance computations are made when selecting k pivots. The computational costs becomes $2kAN$ since the value of μ_D is estimated N times for N groups of pivots and there are A pairs of objects from **U** database objects chosen at random.

Incremental Selection

The second heuristic for choosing pivots as outlined by Bustos *et al.* [2003] is incremental selection. In this heuristic, a pivot p_1 with maximum μ_D value is selected from N objects of **U**.

Then, the second pivot p_2 is chosen from another sample of N objects of **U** such that $\{p_1, p_2\}$ has the maximum μ_D. The third pivot p_3 is also chosen from another sample of N objects of **U** such that p_1, p_2 and p_3 have a maximum μ_D, where p_1 and p_2 are fixed. The process continues for all k pivots. Distance computations for previous pivots are kept in the array of size A in order to save computational time when finding subsequent pivots. Hence, $2NA$ distance computations are required to estimate μ_D. For k pivots, the total computational time becomes $2kAN$.

Local Optimum Selection

The third heuristic is local optimum selection. In this technique k pivots p_i are randomly chosen from objects $u \in$ **U**, where $1 \leq i \leq k$. Using A pairs of objects, a matrix $M(r, j) = D_{pj}(a_r, a_r^{'}), 1 \leq r \leq A, 1 \leq j \leq k$ is computed. A parameter r in matrix M denotes the maximum distance of pivot j, (p_j) to the other pairing object, $a_r^{'}$. Hence, for every r, it follows that $D(a_r, a_r^{'}) = max_{1 \leq j \leq k} M(r, j)$. And this is used to estimate μ_D. The index in M of maximum value of r, r_{max} and second maximum value, r_{max2} are kept. Another pivot parameter *contributor contr* = $M(r, r_{max}) - M(r, r_{max2})$, if $j = r_{max}$ is computed. The parameter determines how much a pivot contributes to the increment of a value $D(a_r, a_r^{'})$ of a row. A pivot with minimal contribution (victim) is replaced by a better pivot selected from remaining database objects denoted X. This computation is repeated for $N^{'}$ times. The matrix construction cost is $2Ak$ distance computations. The cost for searching for a victim is zero since there are no extra distance computations. Finding better pivots costs $2AX$ distance computations done $N^{'}$ times giving a total costs of $2A(k + N^{'}X)$. Since $kN = k + N^{'}X$ the optimisation cost is $2AkN$ distance computations. Values of $N^{'}$ and X can be exchanged during cost optimisation.

Results of comparison of these pivot selection revealed that incremental selection performs relatively better as compared to the other two heuristics. The reason is that it takes advantage of previous work done and more pivots are added without increasing the complexity of the optimisation process. A pivot selection technique implemented in the tsi_cluster algorithm is similar to the incremental selection. It is discussed in detail in the next chapter. Pivots are useful in building tree-structured indices as previously discussed. All other database elements are easily identified and clustered with their reference. Their usage in compact partitioning in order to building tree-structured indices is discussed in sections that follow.

2.6.2 Sequence Search Tree

Giladi *et al.* [2002] developed a Sequence Search Tree (SST) algorithm for searching in metric space. This algorithm is of special interest in this research because its approach is similar to the implementation of the developed algorithm. This discussion begins by showing how windows are used to partition a data set and how they are mapped into a vector space. Finally, constructing the sequence search tree using the windows is shown.

The Sequence Search Tree algorithm implements k-tuple encodings in order to define similarity between pairs of windows in order to map them into a vector space. Giladi *et al.* [2002] apply the method, as follows. Sequences in the database are partitioned into overlapping windows of a fixed length (number of nucleotides) W. A parameter $\triangle$ in range $5 \leq \triangle \leq W/2$, determines the size of the overlap among windows. Windows begin at position $j \times \triangle$ and end at position $j \times \triangle + W$. The value of j = $0, 1, 2...$, $(W/\triangle) - 1$. A query involves finding database windows similar to a query sequence. In order to achieve this, each window is mapped into a vector space. Hence, a vector is created for each sequence for the occurrence of each k-tuple. The size of k is chosen and usually ranges from 2 to 10. Assuming that $k = 2$, then a vector of size $4^k = 16$ is created, where 4 is the number of DNA nucleotides. An example of such tuple combination could be *AA,AC,AG,AT....TG,TT*. An integer I is associated with each of the tuples of the nucleotides and the following formula is used.

$$I(a_1, a_2 \ldots a_k) = \sum_{l=0}^{k-1} 4^l M(a_l)$$

$$M(a_l) = \begin{cases} 0 & \text{if} \quad a_l = A \\ 1 & \text{if} \quad a_l = C \\ 2 & \text{if} \quad a_l = G \\ 3 & \text{if} \quad a_l = T \end{cases}$$

Invalid symbols resulting from repeat masking such as X are ignored. All database windows are mapped into a vector R^{4^k} where the I^{th} entry represents the occurrence of the I^{th} tuple in that window. For example given that $k = 2$, $a_i \in \{$*A,C,G,T*$\}$ and a window *AACCGG*. The following mapping is achieved (1100011000100000) showing that there is an occurrence of *AA, AC, CG, ...* but not an *TG* and *TT* in the window. Since each window has a fixed number of k-tuples, the number of k-tuples being shared by windows from the two sequences is deduced. It is a lower bound on the edit distance between them. The number of k-tuples that occur in one window and not the other, defines the Manhattan (L_1) distance between the two windows. The Manhattan distance is used together with the edit distance as a distance similarity measure on query window sequences in a metric space. The sequence search tree is built as follows:

- Two windows, w_A, w_B are chosen as pivots from the database U. It is assumed that the pivots are chosen randomly because the technique does not specify any criteria for pivot selection.
- Then for every object w_y in the database, an L_1 distance function is computed to the pivots, w_A, w_B.
- A window w_y is assigned to a set A if the distance $d(w_A, w_y) < d(w_B, w_y)$ and to set B otherwise.
- New pivots are chosen from each set and windows are assigned to a set of pivots as described above.
- The process is repeated until there are no more windows to partition or a threshold is reached.

In a case where the resulting tree is balanced, $O(n \log n)$ amount of work is required to construct the tree, where n is the total number of sequences. The number of windows is not considered in the complexity because it is proportional to the number of sequences. Other computational costs require to be reduced. Disk access for example, is reduced by performing a tree construction on the subtree small enough to fit into a block of memory so that disc access occurs once for each subtree. Tree construction is sped up by computing pivots from the same partition. A sample estimate of 1000 windows was used during tests and proved to speed up the running time.

The tsi_cluster algorithm builds up from this design with several modifications because the goals of the two algorithms are different. SST was developed for searching purposes while the tsi_cluster algorithm is for clustering. In the tsi_cluster algorithm, internal nodes of the tree are not created and only resulting leaves are maintained in an array data structure of records to save space.

2.6.3 Burkhard-Keller Trees (BKT)

The second tree-structured index is the Burkhard-Keller Tree (BKT) [Chavez *et al.* 2001]. This tree structure works with discrete distance functions and range queries. It is possibly the first tree to be conceived for searching in metric space and it is constructed as follows.

- an element in a database $p \in \mathbf{U}$ is chosen arbitrarily and acts as a root of the tree.

- $\mathbf{U}_i$ is a set of elements $u \in U$ at distance $i > 0$ from the root p and is denoted $\mathbf{U}_i = u \in \mathbf{U}$, where the distance from the root p to an element u is i and is denoted, $d(u, p) = i$. This is the set of all elements at distance i from the root.

- a BKT is recursively built at the child of p for any nonempty $u \in \mathbf{U}_i$ until there are no more elements or a threshold point is reached as shown in Figure 2.5. The first part of the figure shows the first step of the tree construction and the objects' locations are shown in the second part of the figure. Object u_{11} is the root and all objects at distance $i_1, i_2, ...i_n > 0$ are grouped together in one node. In each group, an object is randomly chosen as a child of the root at that node from which the same process is repeated on the remaining objects at that level. This is done for each level until there are no more objects left.

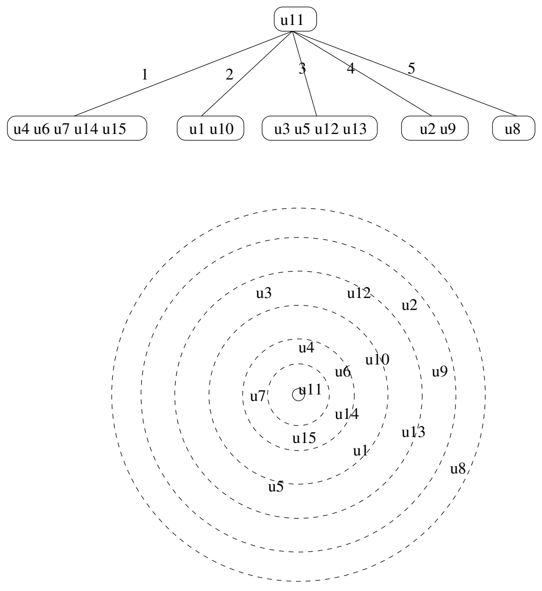


Figure 2.5: An example of a Burkhard-Keller Tree

The Burkhard-Keller Trees differs with the approach taken in the algorithm in several ways. Firstly, the BKT uses one pivot per level while the algorithm uses two pivots. Secondly, the BKT groups elements depending on their ranges from the root. This results in a tree, which is not binary because it has more than two children per node. The BKT is applicable for searching purposes but not clustering because elements that are grouped in nodes do not form clusters. However, use of pivots and range queries provide a good example of how they are applied in metric space for clustering purposes.

2.7 Related Work

Various aspects of the tsi_cluster algorithm have been discussed. The aspects include: distance functions, threshold distance value, pivot selection, clustering in metric space and tree-structured indices. The section continues to explore some of the EST clustering algorithms that have previously been successfully implemented. The section begins with an exploration on techniques employed by clustering algorithms. This section studies the weaknesses and strengths of these algorithms with respect to their computational complexity, which is a major motivation for developing the tsi_cluster algorithm. Strengths and weaknesses of these algorithms have been used in evaluating the performance of tsi_cluster algorithm. The following algorithms are explored: d2_cluster, *wcd*, CLU, UIcluster, suffix array and UniGene. The last discussion is on parallel algorithms implemented to improve efficiency of algorithms designed for stand-alone computers. Development of parallel EST clustering algorithms shows the extent to which efforts are being made to improve efficiency of EST clustering.

2.7.1 Clustering Techniques

Different approaches are taken by algorithms to solve the clustering problem in general. These techniques are discussed in general before exploring specific EST clustering algorithm. The algorithms fall under different categories of clustering techniques.

ESTs can be clustered in a loose or stringent way [Hide *et al.* 1999]. Stringent clustering uses strict rules. It results in shorter consensus sequences and lower coverage of expressed gene. Loose clustering, however, is less strict and results in longer consensus sequences with greater coverage of expressed gene.

Both loose and strict clustering can be done in a supervised or unsupervised way. In the supervised way, clustering is conducted with respect to known reference sequences. In an unsupervised way, however, clustering is done without any prior knowledge [Hide *et al.* 1999]. UniGene performs supervised clustering because it uses known mRNA sequences to perform clustering while algorithms such as d2_cluster and *wcd* perform unsupervised clustering.

Clustering techniques are categorised into two main groups: hierarchical and partitional [Jain *et al.* 1999]. Hierarchical clustering provides a representation of nested grouping of patterns (sequences) and defines a similarity level at which the grouping changes. Single linkage and complete linkage are the main variants of hierarchical algorithms. Single linkage uses the minimum of distances between pairs of patterns in the two clusters. In complete linkage however, the distance between two clusters is the maximum distance of all pairwise distances between patterns in the two clusters. Partitional clustering, however, produces a single set of clusters from a data set and does not allow further changing of clusters. The technique is less computationally expensive and ideal for clustering bigger data sets. The problem with the technique is that it requires knowing the number of clusters before hand. Partitional clustering techniques are not discussed in detail because they do not directly relate to the approach in the algorithm.

The different techniques above are generally affected by some crosscutting issues and some of the issues relevant in this research are:

- whether the clustering is agglomerative or divisive. In agglomerative each pattern begins in a distinct cluster and are merged until a stopping criteria is reached. In divisive however, all patterns begin in one cluster and are split until a threshold is reached.
- whether the clustering is polythetic or monothetic. Distances between pairs of patterns are computed based on which decisions are made. In monothetic, one pattern is used to divide a given collection of patterns.
- whether the clustering is incremental or non-incremental. These issues arise where a data set is very big and there are constraints on memory and execution time.

2.7.2 The d2_cluster Algorithm

The d2_cluster algorithm is discussed as presented by Burke *et al.* [1999]. In the d2_cluster algorithm, every sequence S_i begins in its own cluster C_i. Clustering is based on transitive closure, which requires that two sequences be placed in the same cluster if they share a level of similarity through another sequence. Two sequences S_A and S_B are placed in the same cluster if there exists another sequence S_Y, which satisfies a similarity measure to both S_A and S_B. In order to determine similarity among sequences, distance is computed using d2 distance function and a threshold is defined as a cut-off point on the level of similarity. Details of d2 distance function have been explored in section 2.4.2. The first sequence, S_0 in the data set is selected as a query and the remaining sequences in the data set denoted S_i are targets. Cluster C_i is merged with cluster C_0 if the d2 distance, $d2(S_0, S_i)$ is less than the threshold and this is done for all the remaining sequences in the data set. The process continues with subsequent sequences as queries if they were not merged into other clusters already. Clustering is completed after n iterations, where n is total number of sequences in the database. During each iteration, all target sequences are compared with the query sequence if they are not already clustered. In the worst case, the running time for the algorithm is $O(n^2)$, since for every sequence S_i in the database, there are

n iterations. The tsi_cluster algorithm attempts to improve on this quadratic running time. The outline of the algorithm is as follows:

```
1. Given that the total number of sequences is n
2. Assign every sequence Si to its own cluster Ci
3. for i = 1 to n
4.   for j = i to n
5.     if sequence Si and Sj do not belong to the same cluster then
6.       if d2 score between sequence Si and Sj < threshold then
7.          merge cluster Ci and cluster Cj
8.       end if
9.     end if
10.  end for
11. end for
12. output
```

2.7.3 The wcd Algorithm

The second algorithm to be explored is *wcd* and it was developed by Hazelhurst [2003]. It uses d2 distance function and works like d2_cluster. The type of clustering performed by *wcd* is single linkage clustering where two sequences are put in the same cluster if they are close to each other. Transitive relationship among sequences is taken into consideration, such that if there exists a sequence S_Y close to both sequences S_A and S_B, then S_A, S_B and S_Y are placed in the same cluster. In order to improve efficiency of running time, the algorithm employs a heuristic before the d2 distance function is computed between pairs of sequences. This is based on the assumption that for the d2 score between two sequences to be below a threshold, they must share a specific number of similar words between them [Hazelhurst 2003]. The heuristic avoids d2 distance computation if the number of similar words shared between two sequences is below the specified value. The algorithm saves memory by compressing sequences. However, the memory space saved is relevant only for more than 100k sequences. The heuristic and the saving of memory differentiate the algorithm from the d2_cluster, however both algorithms run in quadratic time in the worst case.

The quadratic running time achieved by *wcd* is still expensive for EST clustering. The need remains, therefore, for more efficient algorithms in terms of running time and memory usage. The *wcd* algorithm has improved memory utilisation and running time as compared to d2_cluster algorithm.

2.7.4 CLU

CLU is an EST clustering algorithm developed by Ptitsyn and Hide [2005]. The algorithm works for both EST and protein sequences although it is not sensitive enough to low similarities. The algorithm works like the d2_cluster and *wcd* algorithm with some variations in the implementation. Similarity among sequences is found using a match detection algorithm, which works as follows:

- Build a hash table H of words from the query sequence S_q.
- Slide a short frame by one word from sequence S_b and for each frame count the number of similar words between the hash table H and the frame. A local similarity function, $F(W_i)$ is used to calculate the similarity function values and is defined as follows:

  $F(W_i) = \sum_{i=0,w}^{w} H_{wi}$. Where w is the width of frame W. Words are represented as numbers in the hash table and have different weightings depending on whether the words are repeats or appear rarely. The technique saves space by compressing sequences like *wcd*.

- Similarity score is found by vector multiplication of $F(W)$ by the precalculated weight factors $\overrightarrow{b}$. The weight index factors are linear coefficient in the equation of the line connecting centres of two contrasting sets of pairs of sequences with and without local similarity. A projection of the categorised $F(W_i)$ distribution $(\overrightarrow{a})$ to a line stretched between pairs of centroids of simulated classes is considered as a similarity index. Hence, a threshold, $|x|$ separating similar and dissimilar sequences is chosen such that:

  $|x| = (\overrightarrow{a}) \times (\overrightarrow{b}); |x| > x_{threshold}$.

- The resulting score is compared to the predefined threshold, $x_{threshold}$.

The algorithm implements a single linkage agglomerative clustering. In this type of clustering, initial clusters have one sequence and are compared to each other. In the algorithm, when a match is found between a pair of clusters, the two clusters are merged, their members are concatenated and a consensus sequence is generated by pairwise alignment as a representation of all the members. As the clusters get merged, the number of clusters reduces, which consequently speeds up the clustering process. The outline of the algorithm is as follows:

```
1. Let the total number of sequences be n
2. Assign every sequence Si to its own cluster Ci
3. for i = 1 to n
4.     for j = i to n
5.       if sequence Si and sequence Sj match then
6.          align consensi of sequence Si and sequence Sj
7.             if alignment confirms a match then
8.                 merge cluster Ci and cluster Cj
9.             end if
10.      end if
11.   end for
12. end for
13. output
```

Although CLU is similar to d2_cluster algorithm, it differs in several aspects. Firstly, CLU does not use the d2 distance function to determine similarity among sequences. Instead it computes that similarity score between pairs of sequences using the $(\overrightarrow{a})$ and $(\overrightarrow{b})$ as previously discussed. Secondly, CLU generates consensus sequences within clusters and uses them for inter-cluster comparison instead of individual sequences in the clusters. However, both algorithms perform single linkage agglomerative clustering as and their running time is quadratic in the worst case, $O(n^2)$.

2.7.5 UIcluster

UIcluster is another EST clustering algorithm developed by Trivedi *et al.* [2002]. It uses edit distance as a measure of similarity among sequences. The measurement is made on words of length M where N bases are required to be similar. Hence, the value of the error tolerance is given by $(M - N)$. The algorithm generally works as follows:

- It reads sequences from an input file one at a time.
- It compares each sequence against every existing cluster
- If a similarity measure is met, a sequence is assigned to an existing cluster or it becomes the first member of a new cluster.
- The process is repeated until all sequences are examined and clustered.

There are three versions of the algorithm. The first version stores clusters in a two-dimensional linked list. Sequences are read from the input file and compared against a representative sequence in the cluster, which is the longest sequence. A hash table is used to keep words (M) of the representative sequence to efficiently evaluate the similarity criteria against a query sequence. The second version of the algorithm uses a global hashing table. Hashes are generated for the new sequence being clustered. These hashes are then indexed into the global hashing table where each element points to a linked list of clusters that contains one occurrence of the hash equal to the index. A detailed comparison is then made between an input sequences and the candidate clusters. The third version is parallelised, such that the computational and memory required are divided across several computers. In this version, each cluster is stored on one computing node. When an input sequence is read, it is distributed to the computing node and a search is made. A sequence is assigned to a cluster with the best match otherwise, it is assigned to its own cluster and to one of the computing nodes in a round robin manner.

All the three versions of the UIcluster algorithm run in quadratic time, $O(n^2)$, in the worst case. Its implementation is similar to d2_cluster algorithm since sequences begin in their individual clusters and a single linkage agglomerative clustering technique is applied. UIcluster differs from d2_cluster and *wcd* because it does not use d2 distance function to measure similarity among sequences.

The four algorithms discussed above (d2_cluster, *wcd*, CLU and UIcluster) run in quadratic time. The quadratic running time is what the tsi_cluster algorithm attempts to avoid. The d2_cluster algorithm and *wcd* use d2 distance function and CLU uses a hash table of words. This is a major similarity between each of these algorithms and the tsi_cluster algorithm.

2.7.6 Suffix Array Algorithm

Malde *et al.* [2003] developed the suffix array algorithm for EST clustering that runs in sub-quadratic time complexity. A suffix array is a lexicographically ordered array of all suffixes of a set of strings. The algorithm determines the level of similarity between pairs of sequences by finding a matching block, which is a contiguous segment that matches exactly between the two sequences. A parameter k specifies the length of the shortest matching block. A k-clique is a set of all sequences with a specific matching block of length k. A similarity score between a pair of sequences is the sum of matching block lengths that are non-overlapping and appear in the same order. A minimal score is defined to be used as clustering threshold.

The algorithm works in three parts. The first part of the algorithm identifies pairs of sequences with matching blocks. The second part computes scores between the sequences and lastly, hierarchical clusters are built. Details of the algorithm are as follows:

1. The first part identifies matching blocks of length k.
    - From the data, construct suffixes and sort them into a suffix array.
    - Suffixes that share a prefix of length k are grouped into cliques.
    - Maximum matching blocks for each clique is generated between each pair of suffixes in the clique.
2. The second part computes scores between pairs of sequences.
    - Matching blocks are collected between pairs of sequences that share at least one matching block.
    - The largest contiguous set of matching block is computed for each pair and their corresponding scores.
3. The third part of the algorithm does the clustering.
    - Clusters are built hierarchically starting with the highest scoring pair.
    - Clusters are split using the threshold. The result is a binary tree whose leaves are the clustered sequences and the root is a cluster of a highest scoring pair.

The algorithm runs in sub-quadratic time. The suffix array is generated in linear time using suffix tree and the sorting is done in $O(n^2)$ in the worst case. Generating pairs of a clique is done in quadratic time in the size of the clique. The trade off between sensitivity and performance is due to the size of the clique. Score calculation between pairs is done in linear time to the number of pairs, p. To cluster an input sequence, the cost is logarithmic to the number of sequences stored in the tree, such that for p pairs and c clusters, the clustering cost is $O(pc \log p)$. Sensitivity of the suffix array algorithm using SANBI benchmarks data set was 94.7%, 85.4%, 56.0% and 50.3% when its clusters were compared to d2_cluster, BLAST, UICluster and UniGene. In spite of this higher sensitivity, the algorithm has relatively lower specificity, which needs to be improved. The tsi_cluster algorithm attempts to improve on specificity, which is one of the weaknesses of the suffix array algorithm.

Although sorting and clique generation have not been factored into the overall complexity, they form a bottleneck in the running time of the algorithm. In bigger data sets, they can dominate the complexity and substantially compromise the running time. Although the suffix array algorithm runs relatively faster, further improvement on the running time is required to avoid the bottleneck created by sorting and generation of cliques. Further work is required on the algorithm to improve its specificity.

2.7.7 UniGene

UniGene is a program for partitioning sequences such as ESTs from GenBank into gene-oriented clusters [Wheeler *et al.* 2003]. It performs supervised clustering using previously grouped ESTs and mRNAs to generate clusters. It starts by building mRNA clusters, which work as reference for EST clustering. ESTs are compared to the existing mRNA clusters using BLAST to generate their own clusters [Schuler 1997]. Clone identifiers are used to assign non-overlapping $5'$ and $3'$ ESTs that originate from the same gene to the one cluster. An authentic $3'$ mRNA end recognisable by the algorithm, is used to anchor the clusters to avoid having multiple dis-joined clusters for the same gene. Hence, clusters generated are known as *anchored clusters*. Clusters without the $3'$ mRNA terminals are discarded and their sequences are rechecked for clustering.

The use of mRNA and clone identifiers to generate anchored clusters distinguishes UniGene from other clustering algorithms discussed before including the algorithm developed in this research. An anchored cluster represents a fully transcribed gene with a $3'$ end. Results from other techniques represent portions of complete gene sequences only.

2.7.8 Parallel EST Clustering

In an attempt to improve efficiency, prior work has been done on clustering EST on parallel computers. Findings show that running time generally improves from quadratic to linear. The approach has an advantage of sharing resources such as memory and processor time from different computers, which are the major constraints on stand alone computers.

Ranchod [2005] successfully parallelised the *wcd* algorithm on 101 processors and yielded a linear running time. The approach utilised the master-slave paradigm. All sequences are sent to slave computers, which perform comparison among sequences and notifies the master if two sequences are close to each other. The master merges sequences that satisfy the similarity measure and maintains load balancing among the slaves by only assigning sequences to slaves with smaller workloads.

PaCE (Parallel Clustering of EST) is another parallel EST clustering algorithm developed by Kalyanaraman *et al.* [2003]. The algorithm uses a generalised suffix tree data structure, which is built in parallel. It also employs the master-slave paradigm where the master is responsible for maintaining and updating all the clusters. It receives pairs of sequences from slaves and determines pairs to be subjected for further alignment. It dispatches pairs of sequences to slaves in batches for alignment, gets back the results, and determines if clusters for the corresponding sequences should be merged. The slaves generate potential pairs for pairwise alignment and also perform alignment as assigned by the master.

Carpenter *et al.* [2002] parallelised d2_cluster algorithm using shared memory. Shared memory was chosen in the implementation because of the significant amount of shared data in the algorithm. The d2_cluster main loop was split into pieces that perform a specific function and distributed among processors. Merging of clusters was not parallelised because of its sequential execution nature. When 15876 sequences were clustered on a SGI 2000 multiprocessor computer, running time dropped from 52 minutes achieved on a single processor computer to 2 minutes.

Another application of parallel clustering using *UIcluster* algorithm has been discussed in section 2.7.5. In general, parallel clustering is implemented in order to improve running time by sharing resources such as memory and processor time from different computers.

The parallel approaches are ideal when computer resources are available and can be easily shared. However, there is still need to come up with an algorithm that efficiently runs on a single processor computer with limited memory and processor speed. The developed algorithm in this research is an example of such efforts.

2.8 Summary

The chapter has provided a detailed biological background of the EST clustering problem. It has further explored aspects of the developed algorithm with respect to prior work with the aim of exposing strengths and weakness of previous algorithms in order to make improvements in implementing the tsi_cluster algorithm. Aspects of the clustering algorithms discussed include: distance functions for measuring similarity among sequences, use of threshold values in determining level of similarity cut-off points, clustering in metric space, pivot selection and tree-structured indices.

Section 2.2 explored the biological background of the EST clustering problem. It specifically discussed the structure, function and composition of DNA, RNA and proteins in organisms. The section covered the transcription of DNA into RNA, translation of RNA into proteins, DNA sequencing and generation of ESTs. ESTs are partial sequences of cDNA experimentally derived from mRNA, which is found in the cell cytoplasm. ESTs are of interest because they represent expressed genes. mRNA is transcribed from DNA in the cell nucleus of multicellular organisms and it is transported into the cell cytoplasm. It represents expressed genes. This is useful for understanding expressed genes and their alternative splicing. However, mRNA is single stranded and, therefore, unstable for experimental purposes. It is reverse transcribed into cDNA, which is double stranded. Sequencing complete cDNA is not possible due to the limited capability of available techniques. Hence, bits and pieces of cDNA are generated starting and ending at random positions in mRNA resulting in several redundant copies known as ESTs.

Section 2.3 explored the EST clustering problem. It further described the process and types of clustering. The EST clustering problem is to group ESTs from the same gene into one cluster. The only available information used to determine whether ESTs originated from the same gene are overlaps that exist among them. The goal of EST clustering is to identify the number and types of expressed genes and their alternative splicing. This is achieved by constructing consensus sequences for each cluster, which represents a complete gene sequence. Computationally, the EST clustering problem is modelled as a graph problem and clusters are found by finding components of the connected graph.

The level of similarity among sequences is determined by distance functions discussed in section 2.4. Shorter distances between pairs of sequences imply a higher level of similarity while longer distances imply lower levels of similarity. Two distance functions were discussed and these were edit distance and d2 distance function. Edit distance is the minimum total number of insertions, deletions and substitutions made to make two different sequences similar. It is mathematically modelled by the recurrence relation, which is computed by either a dynamic programming technique or tabular computation. The tabular computation is more efficient and is preferred over the dynamic programming. The d2 distance function is found by computing difference of squares of the number of similar words found between a pair sequence windows. Words of a given length are compared over a specified window that slides over the two sequences by a specified number of characters.

Quadratic algorithms are inefficient because they perform prohibitively a large number of distance computations and evaluations. The tsi_cluster algorithm reduces the number of distance computations and evaluations by clustering sequence windows in a pseudometric space. Similarity measure among sequence windows is modelled by a distance function that satisfies pseudometric space properties. Clustering in a pseudometric space involves choosing pivots and recursively partitioning sequence windows into two groups based on their proximity to either of the two pivots. The section further discusses metric space properties such as: positiveness, symmetry, reflexive, strict positiveness and the triangular inequality. Choice of pivots is crucial on efficiency of clustering. A general principle of pivot selection heuristics is to select pivots that are far apart. Three pivot selection heuristics discussed in the section were N-Random groups selection, incremental selection and local optimum selection.

Section 2.6 explored two tree-structured indices. The two tree structures discussed were Sequence Search Tree and Burkhard-Keller tree. The section aimed at exploring strengths and weaknesses of construction techniques so that an efficient technique is implemented in the tsi_cluster algorithm. A tree structure is useful in reducing the number of distance computations and evaluations made during clustering. The algorithm recursively selects two pivots and partitions an EST data set into two groups at each level.

Section 2.7 explored prior work done on EST clustering algorithms. The goal of the section was to study EST clustering algorithms with reference to their strengths, which could be applied in the algorithm and weaknesses, which required to be avoided or strengthened. The section started by exploring different clustering techniques and approaches employed by clustering algorithms. The following algorithms were explored: d2_cluster, *wcd*, CLU, UIcluster, suffix array and UniGene. All algorithms explored run in quadratic time except for the suffix array algorithm, which runs in sub-quadratic time.

The overall aim of the research is to efficiently cluster EST in a pseudometric space using a tree-structured index. The clustering technique reduces the number of distance computations and evaluations made to cluster ESTs in $O(p \log p)$ running time.

Chapter 3

A Tree-Structured Index Algorithm

3.1 Introduction

The previous chapter discussed the biological background of the research problem and various areas relating to the tree-structured index algorithm developed in this research. Areas previously discussed include: the EST clustering problem, distance functions for measuring the level of similarity among sequences, use of the metric space in clustering, examples of tree-structured indices and EST clustering algorithms previously developed. The exploration of these areas laid a foundation for this chapter based on which various decisions and choices have been made in designing and implementing the tsi_cluster algorithm.

This chapter provides a detailed overview of a tree-structured index algorithm, its related research questions, their motivation and how these questions have been answered. An overview of the algorithm covers choices and justifications of data structures used and the overall implementation of the algorithm. Discussion on overall implementation involves input data handling, space utilisation, distance functions, pivot selection techniques and the overall clustering process. The overall implementation of the algorithm aims at achieving $O(p \log p)$ running time, where $p = mn$ is the total number of windows in a data set of n sequences and m is the average number of windows per sequence.

This chapter is divided into six sections. Section 3.2 discusses motivations for the development of the tsi_cluster algorithms. The major motivation for developing this algorithm is the prohibitive clustering running time achieved by quadratic algorithm. The specific technique employed by the tsi_cluster algorithm is largely motivated by the work of Giladi *et al.* [2002] and Chavez *et al.* [2001]. An overview of the tsi_cluster algorithm is given in section 3.3. The overview covers data structures used, design and implementation of the algorithm. A discussion on research questions and how they are answered is made in section 3.4 and 3.5 respectively. There are two main research questions. The first question asks whether the algorithm achieves the $O(p \log p)$ running time. The second question examines the quality of clusters generated to ensure that the results are biologically meaningful. Major issues discussed in chapter are summarised in section 3.6.

3.2 Motivation

This research was largely motivated by a Sequence Search Tree (SST) algorithm developed by Giladi *et al.* [2002] for the purpose of searching in metric space. The success of the SST algorithm discussed in section 2.6.2 initiated the idea that ESTs can also be clustered in metric space by construction of a tree structure. Although both algorithms work in metric space in general, the goals and consequently implementation are very different. The SST algorithm use edit distance and k-tuple encoding to measure similarity between pairs of sequence windows. It was implemented to work in metric space. This algorithm however, implements d2 distance function to measure similarity among windows and it works in a pseudometric space. The work done by Chavez *et al.* [2001] further motivated the idea of constructing a tree structure in metric space for the purpose of clustering. Their work, however, was focused on searching. This research built on the same principles to perform clustering.

The other major motivation for the development of this algorithm is the prohibitive running time achieved by quadratic algorithms. Attempts such as parallelisation of quadratic algorithms have being made in order to improve their running time. However, there is still need for clustering algorithms that efficiently run on stand-alone computers. This need has led to the development of the tsi_cluster algorithm. Quadratic algorithms generally perform clustering by doing pairwise comparison of sequences. A sequence is divided into overlapping windows. Distance is measured among the windows from two different sequences to determine their level of similarity. The windows are compared a pair at a time, until a similar pair is found or there are no more pairs. Similarity between a pair of windows implies that the two sequences from which the windows originate are similar. Hence, the two sequences originate from the same gene. Pairs of sequences, which have windows with distances below a specified threshold value, are placed in the same cluster. Given a data set of n sequences, each sequence is indexed into m windows on average, the entire data set generates a total of $p = mn$ windows. Quadratic algorithms on such data sets run in $O(p^2)$ complexity.

It was conceived in this research that ESTs could be clustered in a pseudometric space by constructing a tree-structured index. A tree-structured index is constructed by selecting two windows as pivots from a data set and partitioning the subsequent windows into two groups based on their proximity to the selected pivots. Relative distances from the two pivots to all the windows are computed consequently mapping the windows into a pseudometric space. Construction of a tree-structured index in such a manner costs p amount of work per level and if the resulting tree is balanced, it yields a depth of $\log p$ and an overall computational complexity of $O(p \log p)$. Final partitions (leaves) in the tree are then merged to generate clusters. Figure 3.2 illustrates this process. The tsi_cluster algorithm is formally outlined in section 3.3 and diagrammatically shown in Figure 3.1.

3.3 Overview of the Algorithm

This section explores components and details of the algorithm design and implementation as shown in Figure 3.1.

- Given a data set of n sequences, the algorithm indexes overlapping windows $w_{i,i'}$ of size *l* equal to 100 in sequences. The number of windows generated per sequence, m depends on the length of the sequence and size of the overlap, $\triangle$. Step *A* in Figure 3.1, represents all windows, $p = mn$ generated. Windows are denoted r_i, where $0 \leq i \leq p$ is a position in a partition.

- From a data set, two windows, p_1 and p_2 at $pDistance \geq 2\theta$ apart are chosen as pivots. $pDistance$ is the minimum distance between a pair of selected pivots and θ is the threshold value. In cases where pivots at $pDistance$ apart are not found, a window with the longest distance from the first window (first pivot) is selected as a second pivot. Pivot selection is discussed in detail later in section 3.3.3. This is step *B* in Figure 3.1 of the algorithm.

- Each window in the partition is compared to the two pivots in step *C* and is partitioned with the pivot to which it is closer. The threshold point is reached when no more pivots can be found at a distance greater than the threshold value. The process recursively continues and terminates when there are no more windows left or a threshold point is reached.

- At step *E*, final partitions are generated and merged in step *F* and stored in step *G*. These final partitions contain windows from different sequences such that merging the windows consequently merges sequences to generate clusters.

The pseudocode for the algorithm is as follows:

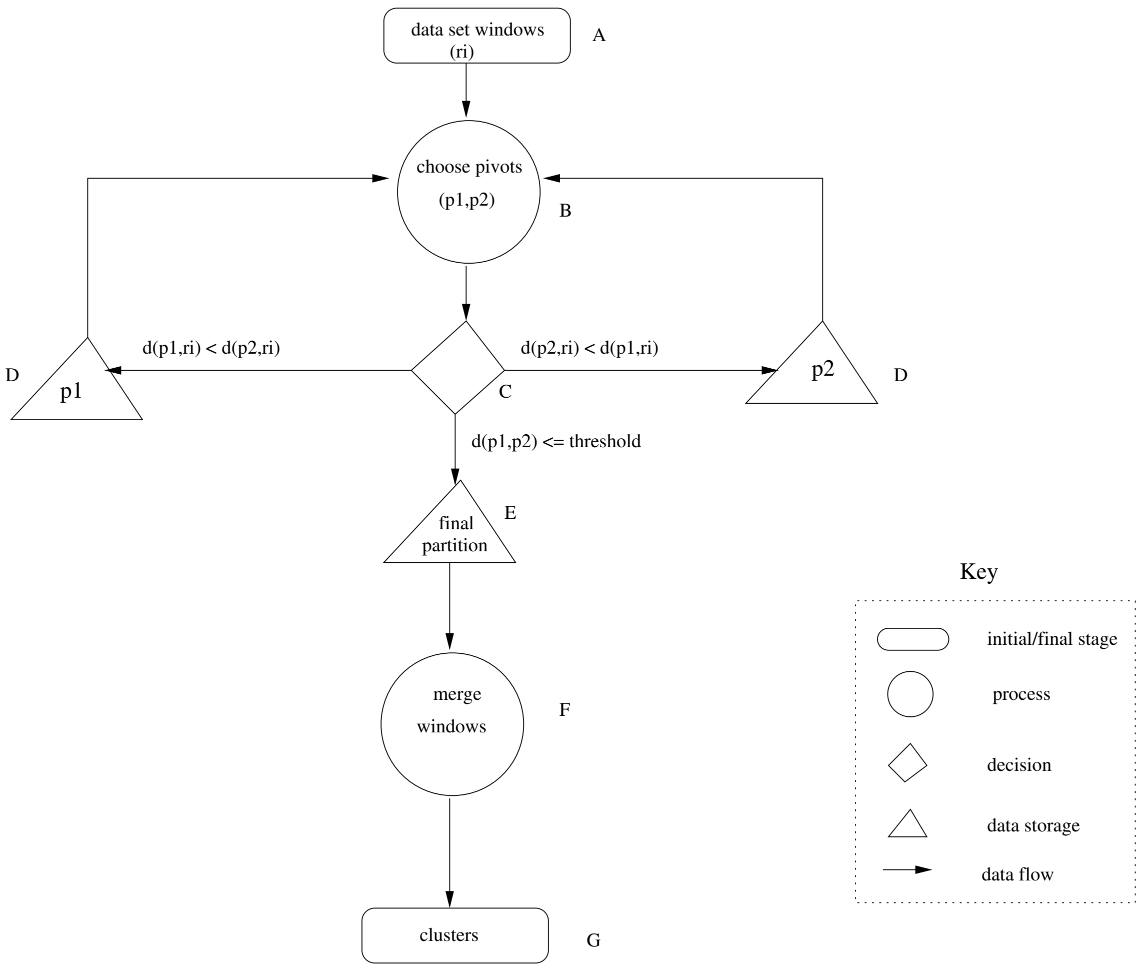


Figure 3.1: Construction of a tree structure

1. Let $(\mathbf{X}, d)$ be a pseudometric space with a distance function d that determines the level of similarity between pairs of sequences and $\mathbf{X}$ a set of valid objects (a data set of sequences).
2. Let θ be a distance threshold value that defines a cut-off point on the level of similarity between pairs of sequences.
3. Given n sequences S_i, where $i = 0, 1, ..n$.
4. Read the sequences into an input array, A.
5. Partition the sequences into windows $w \sqsubseteq S_i \in \mathbf{X}$, which overlap with a parameter $\triangle$; where $S_i \in \mathbf{X}$ means that S_i is in the set $\mathbf{X}$ and $w \sqsubseteq S_i$ mean that w is a subsequence of sequence S_i.

6. Index the windows with respect to their respective sequence position i in the array A, and their starting positions i', in the sequence. Hence the windows are denoted $w_{i,i'}$.

7. Choose initial pivot windows, $(w_{x,x'} \in \mathbf{X}$ and $w_{y,y'} \in \mathbf{X}) \in P_j$ such that $d(w_{x,x'}, w_{y,y'}) \geq 2\theta$. P_j is a partition and $1 \leq j \leq mn$.

8. While there are still more windows, partition window $w_{i,i'} \in \mathbf{X}$ in the following way:
   - If $d(w_{x,x'}, w_{i,i'}) < d(w_{y,y'}, w_{i,i'})$ then
   - Assign $w_{i,i'}$ to the partition of pivot $w_{x,x'}$ or to the partition of $w_{y,y'}$ otherwise.

9. Recursively do steps 7 and 8 on resulting partitions, P_j until there are no more windows to partition or a threshold value θ is reached.

10. For all the final partitions (P_j), generated in step 9, if they have windows from different sequences, S_{i_x}, S_{i_y}, place the sequences (S_{i_x}, S_{i_y}) into the same cluster C_x.

11. end.

3.3.1 The Clustering Process

The clustering process in the algorithm works as follows. Input sequences in a data set are counted and sequentially read into a sequence array, A. Each sequence, S_i is identified by an index position in the array, i. Windows are indexed in each sequence and their identification values are kept in an array of records, r. A record consists of three fields: an index position of a sequence, i a starting position of a window in a sequence, i' and the partition number, h. A window is generally denoted $w_{i,i'}$, where i is the index position of a sequence and i' is the starting position in the sequence. The notation r_j refers to the record that identifies a window $(w_{i,i'})$ at position j in the record array, r. The total number of windows in a data set is given by $\frac{n(m-l)}{\triangle} + 1$, where n is the total number of sequences in the data set, m is the average length of sequences, l is the size of a window and $\triangle$ is the overlap. Size of a window, l is usually equal to 100 nucleotides and overlaps range from 5 to 25 nucleotides.

The clustering process involves indexing overlapping windows in sequences, selecting pivots and recursively partitioning the data set into two. The following example is used to illustrate the process and Figure 3.2 aids the illustration. Consider a data set of 100 sequences, with each

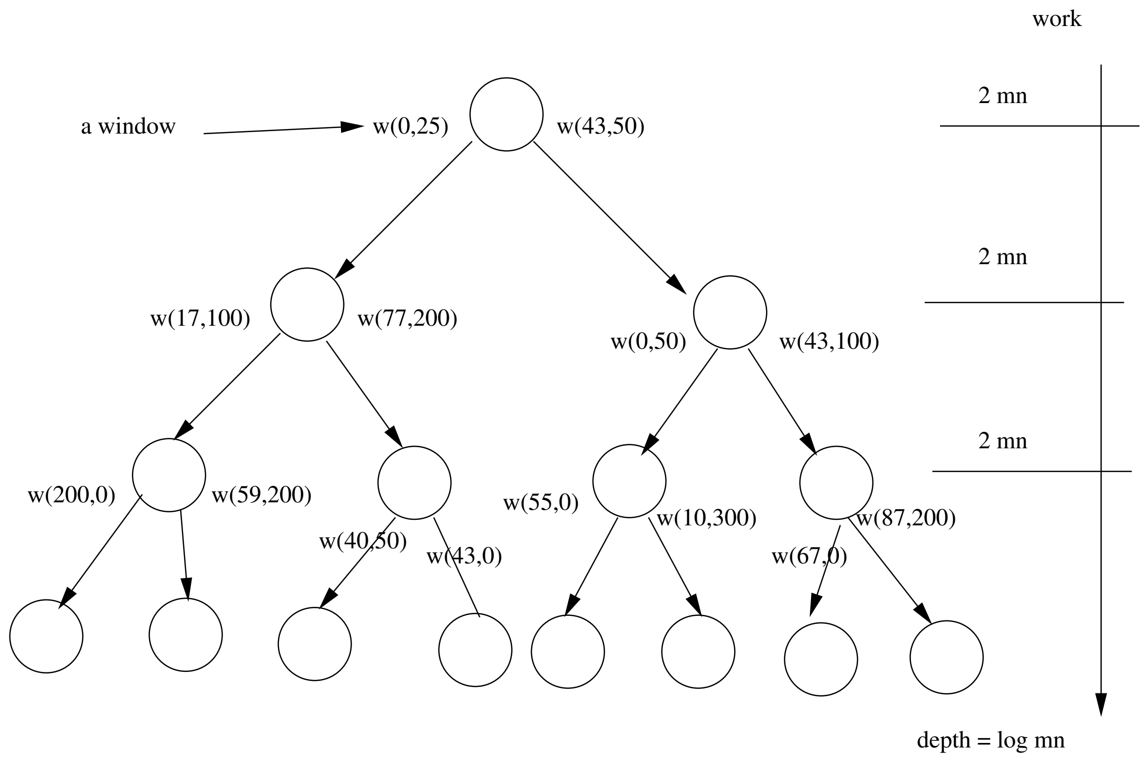


Figure 3.2: Construction of a tree-structured index

sequence having an average length of 300 nucleotides. The sequences are read and stored in a data structure where there are indexed from position 0 to 99. In each sequence, there are $\frac{300-100}{25} + 1 = 9$ windows, where 25 is the overlap, 100 is the size of the window and 300 is the length of a sequence. The total number of windows generated is $9 * 100 = 900$. A sequence indexed at 0 has the following windows: $w_{0,0}, w_{0,25}, w_{0,50}, w_{0,75}, w_{0,100}, w_{,125}, w_{0,150}, w_{0,175}, w_{0,200}$. Windows from sequences 1 to 99 have the same starting positions but they have a different sequence index number. Consider that in the first partition windows $w_{0,25}$ and $w_{43,50}$ are chosen as pivots. This implies that the two windows are at a distance greater than twice the threshold value apart. It further implies that $w_{43,50}$ is the first window to be found at the specified pivot distance. The remaining windows are compared to the two pivots. They are partitioned with the pivot to which they are closer. Two partitions are created (partition 1 and partition 2) at the stage on which the clustering process proceeds. In partition 1, windows $w_{17,100}$ and $w_{77,200}$ are selected as pivots while in partition 2, windows $w_{0,50}$ and $w_{43,100}$ are selected as pivots. The rest of the windows are compared to the pivots and partitioned accordingly in their respective partitions. This process recursively continues until there are no more windows to partition or a threshold point is reached.

This process constructs a tree structure as shown in Figure 3.2. When the process termines, the final partitions contain mixed up windows from different sequences. Sequences with windows in the same final partition are merged forming final clusters.

Records that store indices of windows are kept in an array data structure denoted r. The notation r_j refers to the window record at poisition j, where $1 \leq j \leq mn$ is a position in the array of window records. The record further stores a field, h that identifies a partition to which a window belongs at any particular point in the recursive process. The relationship between the representation of windows and records is shown in Figure 3.3.

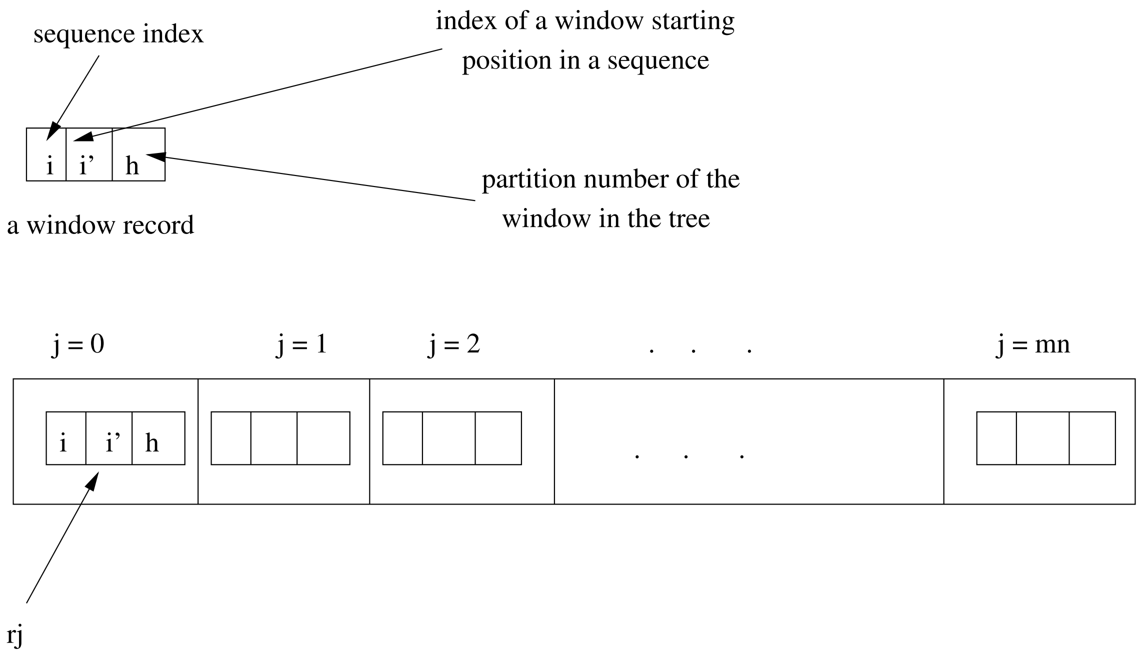

sequence index
index of a window starting position in a sequence
i
i'
h
a window record
partition number of the window in the tree
j = 0
j = 1
j = 2
. . .
j = mn
i
i'
h
. . .
rj

Figure 3.3: Representation of windows in a partition

When the two pivots are chosen in a partition, windows are labelled with the partition number of the pivot to which they are closer. Partition numbers increment by one per recursive call. Windows are sorted based on their partition numbers, hence, it is important that there should be no redundancy in partition numbers. Figure 3.4 illustrates the partitioning process.

- The first grey rectangular box in the figure labelled 0, represents the first partition in the clustering process. It contains all windows in a data set, which belong to the same partition. The value of h for all windows is 0 at this point. The algorithm reuses the same space during the partitioning process. Grey rectangles represent windows in a data set that are not yet partitioned.

- Two pivots are selected and the value of h in windows closer to pivot is assigned to 1. In windows closer to pivot two, the value of h is assigned to 2. Windows with h equal to 1 are moved and placed at the beginning of the partition and those closer to pivot two $(h = 2)$ are placed at the end of the partition.

- The next recursive call is made on the first half (1). The algorithm randomly selects any of the two partitions in the next recursive call. Windows in this partion are labelled with partition numbers (h) 3 and 4 depending on the pivot to which they are closer and are sorted accordingly.

- The next recursive call is made on partition 5 and windows in the two resulting partitions are labelled 7 and 8. At this, point a threshold point is reached and no further recursion takes place on partitions 7 and 8.

- Partition 6 divides into partitions 9 and 10 where a threshold point is also reached. The same scenario occurs on partitions 11 and 12 and then the algorithm moves to partition 2.

- A similar process takes place on partition 2 until a threshold point is also reached or there are no more windows left. As shown in the figure, memory utilisation is minimised by reusing the same memory space throughout the execution of the algorithm. A tree is built by shifting around windows with respect to pivots to which they are closer. Windows in the final partitions are merged and the process is efficiently achieved by using a union-find data structure discussed in section 3.3.4.

3.3.2 Distance Function

A distance function is used to determine the level of similarity among sequence windows. There is an inverse relationship between the level of similarity and the distance among windows. A bigger distance implies less similarity while a smaller distance implies bigger similarity. The initial plan was to use edit distance but because of its quadratic complexity, the d2 distance function has been implemented in the algorithm. The d2 distance function has been implemented to compare sequence windows in linear time. This has been an advantage in improving the running time of the algorithm. A distance threshold value θ has been defined as a cut-off point on the level of similarity among windows. The level of similarity among sequence windows is determined as follows:-

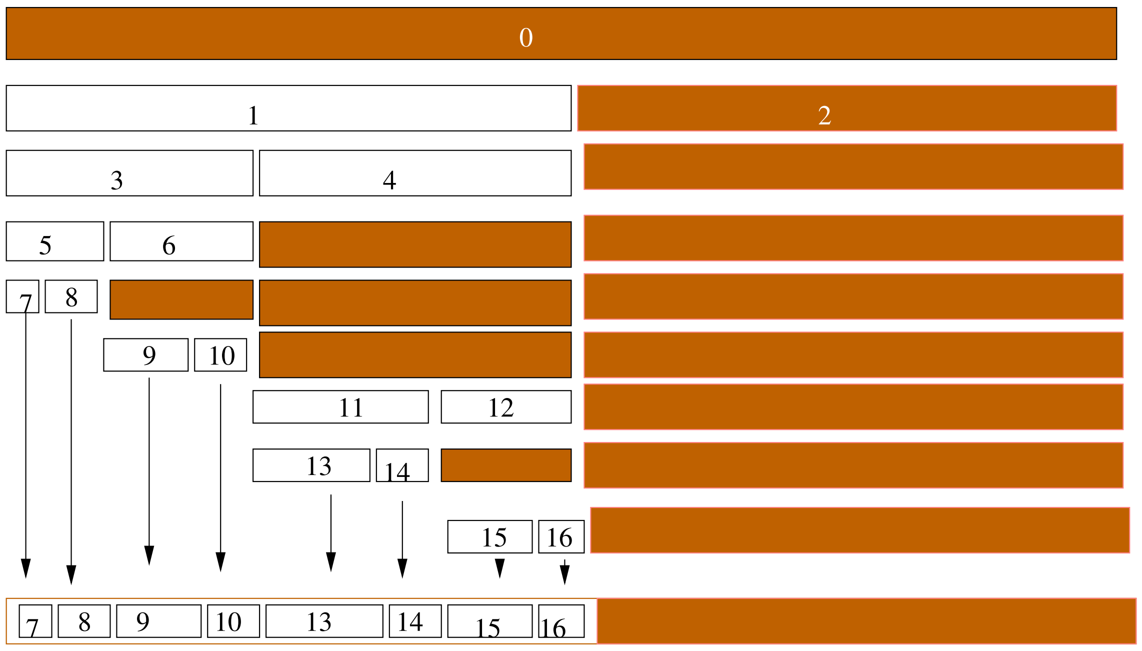


Figure 3.4: Partitioning process

Two windows $w_{x,x'}$ and $w_{y,y'}$, where $w_{x,x'}$ is a window in sequence S_x with length m and $w_{y,y'}$ is a window in sequence S_y also with length m, are regarded similar if $d(w_{x,x'}, w_{y,y'}) \leq \theta$.

3.3.3 Pivot Selection

Pivots are very critical in the construction of a well-balanced tree-structured index of the algorithm. They affect the structure and skewness of the resulting tree and consequently the running time of the algorithm. Pivots that are far apart are expected to yield a well-balanced tree structure as opposed to those that are closer to each other. Selection of pivots that are far apart however, is a big challenge. Random selection of pivots is easy to implement but does not guarantee choosing pivots that are far apart. Far apart pivots are expensive to choose when they are positioned at the far ends of a partition. To avoid this cost, the *"far apart"* in the algorithm is defined by approximating a parameter $pDistance$. A minimum distance between two pivots $pDistance \geq 2\theta$ is approximated.

In general, pivot selection takes a linear time of $O(p)$. Experiments presented in chapter 4 have been conducted to determine an efficient pivot selection technique, which contributes to the efficient running time of the algorithm and quality clusters. Several techniques were implemented and tested and they generally had the following format:-

1. Select the first window r_i, where $0 \leq i \leq p$, in a partition of size p.
2. Sequentially search for another window $r_{i'}$ such that $d(r_i, r_{i'}) \geq pDistance$, where $pDistance$ is the minimum distance between the two pivots.
3. If such a window is not found, store a window, r_{max} with maximum distance from the initial window r_i and continue searching.
4. If there is no window found at a distance $d(r_i, r_{i'}) \geq pDistance$ from r_i, choose r_{max} as the second pivot. If $d(r_i, r_{max}) \leq \theta$, the algorithm terminates.

3.3.4 The Union-Find Data Structure

The union-find data structure maintains and manages a collection of disjointed sets. The data structure stores elements by using trees where each object points to another object except for the root, which points to itself. It supports set operations such as creation, finding elements and merging. In this research, it is used to merge final partitions in order to generate clusters. The union operation merges final partitions using windows and consequently places sequences together in clusters as shown in Figure 3.2. This process continues until there are no more partitions to merge. The operation yields an amortised $O(\log^* p)$ running time [Quinn 1987]. This running time has made the union-find data structure a favourable choice for the clustering process because it does not increase the complexity of the algorithm.

3.4 Research Questions

The ultimate goal of this research is to design and implement an algorithm that clusters expressed sequenced tags in $O(p \log p)$ running time in order to make an improvement over quadratic running time of $O(p^2)$. This reduced running time is achieved by reducing the number of pairwise distance computations and evaluations made among sequence windows during the partitioning process. Instead of comparing all windows to each other, windows are compared to two pivots only per recursive call. Two research questions arise from this implementation:-

1. Can the tree-structured index algorithm cluster expressed sequence tags in $O(p \log p)$ running time?

2. Can the tree-structured index algorithm successfully cluster expressed sequence tags in pseudometric space?

The first research question is more concerned with whether the mathematical model is correctly translated into the algorithm and that it runs accordingly. Hence, the complexity of the algorithm is analysed both mathematically and empirically. Running time is very critical in EST clustering because bigger data sets are required to achieve meaningful results. These bigger data sets require to be clustered in a period that is as short as possible to enhance efficiency in all other fields that rely upon EST clustering. The first research question, therefore, examines whether a the tsi_cluster algorithm clusters ESTs in $O(p \log p)$ running time. The algorithm has a number of functions responsible for different computations. Some of the functions are: reading input, generating windows, distance computation, pivot selection, partitioning windows and merging final partitions. All these functions are analysed and examined whether their collective complexity yield $O(p \log p)$.

It should be noted that a better complexity does not guarantee efficient running time in all data set cases. Specifically, the $O(p \log p)$ complexity does not guarantee a faster execution time as compared to the $O(p^2)$ in all data sets. In most cases, quadratic algorithms run faster on smaller data sets as compared to algorithm with linear or logarithmic complexity. Their running time become prohibitively slow where size of the data set gets larger. Answering the first research question, therefore, requires determining constant factors that largely contributes to the running time and complexity of the algorithm. In the presence of larger constant factors, the positive impact of the logarithmic complexity is hindered. However, with smaller constant factors the impact is remarkable in the efficiency of the algorithm.

The triangle inequality has been implemented and the extent to which it contributes to increasing the efficiency of the algorithm is examined under the first research question. Tests were conducted to examine the frequency of the occurrence of the triangle inequality among sequence windows and the extent to which it reduces running time and improve the quality of resulting clusters.

The second research question examines whether the algorithm solves the biological problem correctly. This implies that the algorithm generates clusters that are biologically meaningful and useful. It questions whether the tsi_cluster algorithm successfully clusters EST sequences. This is defined by sensitivity and specificity of generated clusters as discussed in section 3.5.3. It is not useful to have a faster running time with wrong results. It is therefore important that correct clusters that are biologically useful be achieved in the $O(p \log p)$ running time.

3.5 Answering the Research Questions

The research maps a biological problem to a mathematical model, which is translated into the tsi_cluster algorithm. Answering these research questions requires showing that the algorithm correctly maps the mathematical model and that it solves the biological problem in question. Tests have been conducted using different data sets to determine whether the algorithm solves the research problem and consequently answers the research questions.

3.5.1 Parameters of the Algorithm

Before answering the research questions, parameters that enable the algorithm to yield optimal results had to be experimentally determined. The parameters are: size of the window, overlap of the window, threshold value, minimum pivot distance and a pivot section technique. A window size that yield optimal results was deduced experimentally. Threshold values and overlap parameters were adjusted accordingly during experiments to come up with optimal values. The two parameters have a direct impact on running time and quality of generated clusters. Larger values improve running time, but they have a negative impact on the quality of generated clusters. Small threshold and overlap values however, negatively increase running time but improves the quality of resulting clusters. As such, both the threshold value and overlap parameter that balance running time and quality of clusters had to be experimentally determined and their results were used in testing the performance of the algorithms.

Experiments were conducted to deduce a pivot selection technique that yields efficient clustering with respect to running time and quality of clusters. The choice of pivots greatly affects structure of the tree and consequently the performance of the algorithm and, therefore, its choice requires proper experimentation. Poor choice of pivots results in skewed tree, which in the worst case can be constructed in $O(p^2)$ complexity, which intends to be avoided.

3.5.2 The First Research Question

Both analytical and empirical tests were employed in answering the first research question. The analytical approach involved analysing and showing that the overall running time of the algorithm yields $O(p \log p)$ running time. Analysis was done on all functions of the algorithm in order to determine whether they yield the predicted running time. The expected running time is achieved when the resulting tree is balanced. This is highly affected firstly by the pivot selection technique used and secondly, by the nature of a data set being clustered. Getting a balanced tree requires special data sets, which may be difficult to obtain or simulate. In the worst case scenario, the tree grows on one side only and at every level, there is one window up to the depth p. This case implies an occurrence of a very unlikely situation where every window in the data set is unique.

Empirical tests have been performed to supplement answering the first research question. As discussed in section 3.4, the logarithmic complexity deduced mathematically does not guarantee that it runs relatively faster than quadratic algorithms because there are other constant factors that may have significant impact on the running time. The mathematically deduced complexity further relies on generation of a balanced tree. It is, therefore, important to show that empirical results agree or disagree with analytical results. In cases of disagreement, investigate and understand the possible reasons. Hence, running time of this algorithm was compared to that of *wcd*, a quadratic algorithm developed by Hazelhurst [2003]. The two algorithms were run and timed on the same computer and data sets. Their running times were compared and contrasted to determine whether the tsi_cluster algorithm runs relatively faster. Such tests do not provide conclusive results, but do provide a real time indication of the difference in running time of the two algorithms. Average running times were used to compare performance of the two algorithms. A similar testing approach was done by Kalyanaraman *et al.* [2003] in which the running times of PaCE algorithm was compared to CAP3.

3.5.3 The Second Research Question

The second research question has been answered by experimentally comparing resulting clusters of the algorithm to those of *wcd*, a quadratic clustering algorithm developed by Hazelhurst [2003]. The two algorithms were run on similar data sets and the number, size and contents of the clusters were compared and contrasted. Comparing and contrasting clusters to the results of other methods has been previously used by Ptitsyn and Hide [2005]; Burke *et al.* [1999]; Malde *et al.* [2003] to determine the level of correctness. Ptitsyn and Hide [2005] compared results of CLU algorithm with those of d2_cluster with respect to the number of generated clusters, their size and contents. The d2_cluster results were also compared to those of UniGene in attempt to establish the correctness of d2_cluster [Burke *et al.* 1999]. A similar approach has been adopted in this research, however caution was taken in comparing and contrasting the results. The same distance function and parameters values such as window size and threshold value were used in both algorithms in order to meaningfully compare the results.

Clusters produced by the algorithm are compared to those generated by *wcd* using the *Jaccard index*, which is an approach that measures quality with respect to sensitivity and selectivity of generated clusters [Kalyanaraman *et al.* 2003; Malde *et al.* 2003; Trivedi *et al.* 2003]. To compute the Jaccard index, the following parameter are used: true positives (TP) are pairs found in both clusters, false positives (FP) are pairs found in the first cluster but not in the second one and false negatives (FN) are pairs found in the second cluster but not in the first one. Using these parameters, the following measurements are defined according to Kalyanaraman *et al.* [2003]:

- The Jaccard index also known as overlap quality (OQ), $OQ = \frac{TP}{TP+FP+FN}$.
- Specificity (SP), $SP = \frac{TP}{TP+FP}$. This is the fraction of correctly predicted pairs with respect to total pairs.
- Sensitivity (SE), $SE = \frac{TP}{TP+FN}$. This is the fraction of correctly predicted pairs.

It should be noted that results have been presented with respect to specificity and sensitivity only. This was done to simplify the presentation and to avoid confusion in the interpretation of the results. In most cases, however, the Jaccard index is always equal to sensitivity because the value of false positives is usually negligible.

3.5.4 Memory Utilisation

The amount of memory required by the algorithm is very crucial because it affects the usability of the algorithm. Larger memory requirements can make an algorithm unusable despite being efficient in its running time and quality of clusters. It was previously argued that the tsi_cluster algorithm is able to cluster huge data sets efficiently with less amount of memory. This was tested with the aim of investigating the extent to which the algorithm efficiently utilises memory with respect to data set sizes.

The algorithm was run and tested under memory constraints because both the tree-structured index and the data set had to fit in memory during execution. Memory usage was experimentally tested under varied parameter values. This was done to study and deduce the memory usage behaviour of the algorithm. Size of a window and overlap are the two main parameters that affect the number of windows generated and consequently memory utilised by the algorithm. When size of a window is fixed, the overlap parameter is the only major parameter that dictates the level of memory utilisation of the algorithm.

3.6 Summary

This chapter presented design and implementation details of the tsi_cluster algorithm. Aspects of the algorithm implementation include: getting input data, distance function computation, pivot selection and the clustering process. Input data for the algorithm are EST sequences in Fasta format, which are partitioned into overlapping windows. Windows are identified by sequence index positions in the input array and their starting positions in their respective sequences. Similarity among windows is determined by using a d2 distance function. The d2 distance function was used because of its linear computational complexity, which does not create a bottleneck in the running time of the algorithm. Pivots are chosen by selecting two windows with a minimum distance of 2θ apart. Windows are partitioned into two groups depending on how close they are to the two pivots. The clustering process recursively continues on each generated partition until all windows are partitioned or a threshold point is reached. The resulting final partitions are merged to generate final clusters and the union-find data structure has been used to efficiently merge final partitions into clusters.

The algorithm is examined through two main research questions. The first question mathematically examines whether the overall complexity of the algorithm yields $O(p \log p)$ running time. All the components of the algorithm such as getting input data, distance computation, pivot selection and merging clusters, were analysed and their combined complexity summed up to collectively yield $O(p \log p)$ running time. Empirical tests were conducted to supplement answering the research question. Tests involved comparing average running times of this algorithm to that of *wcd*, another quadratic algorithm in order to determine whether the tsi_cluster algorithm runs relatively faster in real time.

The second question examined whether the algorithm successfully clusters ESTs in a pseudometric space. The question is answered by comparing the quality of clusters generated by the algorithm using *wcd* as a control algorithm. Specificity and sensitivity were used to determine the quality of generated clusters. Memory usage was also tested over varied parameters to determine the memory utilisation levels of the algorithm.

Chapter 4

Experiments

4.1 Introduction

This chapter describes experiments conducted to test the performance of the algorithm. It also describes the experimental environment, setups and data sets used. The experimental environment refers to the hardware specification used and the possible limitations. Hardware specification is discussed in section 4.2. The data sets used are described in section 4.3 with reference to size and other possible properties. Experimental setups are covered in section 4.4. The section describes experiments that have been conducted and their justifications. Experiments were conducted were on the running time, quality of clusters generated, space utilisation and the effect of the triangle inequality. Preliminary experiments were conducted to come up with optimal parameters such as window size, threshold value, overlap parameters and pivot selection technique. The optimal values for parameters were used in the major experiments. Section 4.6 summarises the chapter.

4.2 The Hardware

Experiments to determine the performance of the algorithm with respect to running time and quality of clustering were run on Juggernaut, an Intel Xeon, model 4, stepping 1, with 3 gigahertz processor, 4 gigabytes of random access memory and 1 megabyte on processor cache memory. A processor was always dedicated to an experiment throughout its execution.

Memory utilisation and all preliminary experiments were conducted on a 3 gigahertz processor desktop computer with 1 gigabyte of random access memory. The computer was dedicated to these experiments.

4.3 Data Sets

Two main data sets were used in running time, quality of clustering, memory utilisation and all preliminary experiments. These data sets were the human eye South African National Bioinformatics Institute (SANBI) benchmark data set with ten thousand sequences and the C-series, a simulated data set with 125719 sequences. The C-series was used on the running time experiments while the benchmarks data set was used on quality of clustering, memory utilisation and all preliminary experiments. The SANBI benchmarks data set comprised real 10000 ESTs from cDNA of genes expressed in the human eye and it was obtained South African National Bioinformatics Institute. This is a real data set and it was proper to test the clustering quality using such a data set.

The C-series data set was artificially generated from 2985 full-length cDNA downloaded from the National Institute of Health web site [NIH 2007]. An ESTsim tool, developed by Hazelhurst [2003] is used to simulate generation of ESTs. The tool is able to introduce variuos errors that occur during EST generation into a data set. The tool introduced 1% of evenly distributed single base scaling errors into the C-series data set. The C-series data set was divided into smaller data sets whose sizes increased progressively from 10%, 20% ... 100% with a uniform error distribution. The uniform error distribution in the data set makes it suitable for studying the complexity of the algorithm. For the purpose of studying quality of clusters generated, the SANBI benchmark data set was used.

In addition to these data sets, Drosophila melanogaster and public cotton data sets were used to study the performance of the algorithm on different data types. The public cotton data set comprised 29992 ESTs derived from RNA samples, harvested from 5 to 10 DPA tetraploid wild-type upland cotton ovules [Shi *et al.* 2006]. Sizes, number of sequences and types of all these data sets used in experiments are found in Table 4.1.

4.4 Experimental Setups

4.4.1 Quality Performance of the Algorithm

Overall performance was determined by the running time of the algorithm, quality of generated clusters and memory utilisation. Quality of clusters is defined by the sensitivity and specificity. These are determined by comparing clusters generated by this algorithm to those generated by *wcd* algorithm version 0.2.3.4 Hazelhurst [2003], another clustering algorithm that runs in quadratic time. The *wcd* algorithm has been chosen because it uses a similar distance function

data set	**size (Mb)**	**number of sequences**	**type**
SANBI Benchmarks	5.2	10000	real
C01	5.8	12579	simulated
C02	11.6	25178	simulated
C03	17.4	37790	simulated
C04	23.2	50350	simulated
C05	29.1	62914	simulated
C06	34.8	75467	simulated
C07	40.6	88015	simulated
C08	46.4	100585	simulated
C10	58.0	125719	simulated
Drosophila melanogaster	65.9	83545	real
Public cotton	17.0	29992	real

Table 4.1: Data Sets Used

and generates higher quality clusters. Hence, there can be confidence in tsi_cluster algorithm if it generates clusters that are better than or similar to those of *wcd*. In addition to measuring the quality of clusters, the running times of the two algorithms have also been compared and contrasted with the parameters in both algorithms being adjusted equally to achieve meaningful comparison.

Further experiments were conducted to study effects of data sets used on the quality of clustering and the impact of the triangle inequality on the performance of the algorithm. Both experiments are discussed in this subsection.

- **The Running Time**

  The Running time was experimented on Juggernaut with the C-series data set. System time was used to capture the running time because it covers both program processor execution and input/output time. Experiments were run ten times for each overlap. A graph was plotted using average running time values to determine the complexity of the algorithm. Standard deviation was used to show the deviation of the data from the mean. Graph fitting using linear and multiple regression was used to find the functions that closely define graph lines, consequently the complexity of the algorithm.

- **Quality of Clusters**

Clustering quality of the algorithm was experimented on Juggernaut computer. The SANBI benchmarks data set with 10000 sequences was used on quality of clustering experiments. The algorithm and *wcd* were alternatively run on the same computer with the same threshold, window size settings and the data set. The *wcd* algorithm was used as a control to which the results of the tsi_cluster algorithm were compared. Quality of clusters was deduced with respect to specificity and sensitivity.

Specificity is a fraction of correctly predicted pairs of clustered sequences with respect to the total number of predicted pairs. For example, if the algorithm generates 100 pairs, specificity defines the number of pairs that correctly predicted out of 100. A specificity of 20% implies that both algorithms generated exactly the same 20 pairs out of 100.

Sensitivity on the other hand, is the fraction of correctly predicted pairs by the algorithm out of all pairs generated by the control algorithm. If the control algorithm generates 200 pairs and the algorithm finds only 20 pairs, then the sensitivity of the algorithm is 10%.

- **Memory Utilisation**

  Experiments on memory utilisation were conducted on Juggernaut computer. The algorithm was run with the SANBI benchmarks data set. Memory utilised by the algorithm for each size for different overlap values was recorded. Overlaps ranged from 5 to 25 with all other optimal parameters being kept constant. Windows of size 100 were used with a threshold of size 40 and initial minimum pivot distance of 144. The experiment was only concerned with the size of the data set and the amount of space the algorithm used and not the nature of the data being used. Memory utilisation of this algorithm was compared to *wcd* to deduce the algorithm with lower memory utilisation.

4.4.2 Effects of Data Sets Used

Experiments were conducted to determine whether the algorithm performs equally on different data sets. Three data sets were used, the SANBI benchmarks, Drosophila melanogaster and the public cotton data set. Experiments were run on Juggernaut with optimal parameter values being kept constant.

4.4.3 Triangle Inequality

The triangle inequality was implemented to reduce the number of distance computations and evaluations that are made during clustering. This was one of the major motivations behind the design and implementation of the algorithm. The experiment aimed at determining the extent to which the triangle inequality holds and its effect on the performance of the algorithm. Triangle inequality properties were used to deduce the nearest pivot to a window without the computing distance. The implementation is explained with reference to Figure 4.1.

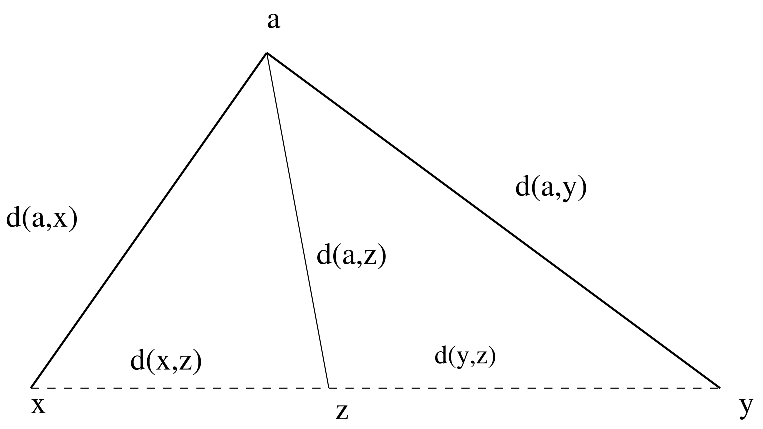


Figure 4.1: Triangle inequality implementation

Let a be the previous pivot to which both current pivots x and y and a window z were already compared with distances $d(a,x)$, $d(a,y)$ and $d(a,z)$ respectively.

The following triangle inequality properties can be applied.

- $|d(a,z) - d(a,x)| \leq d(x,z) \leq d(a,z) + d(a,x)$.
- $|d(a,z) - d(a,y)| \leq d(y,z) \leq d(a,z) + d(a,y)$.

If $d(a,z) + d(a,y) \leq |d(a,z) - d(a,x)|$ holds, then window z is closer to y. If $d(a,z) + d(a,x) \leq |d(a,z) - d(a,y)|$ is true then, z is closer to x.

In general, before computing distance between pairs of windows, the triangle inequality properties were evaluated. If the properties did not hold, then distance was computed, otherwise windows were partitioned based on the properties. Triangle inequality tests were made on Juggernaut computer with the SANBI benchmarks and its subset sub set data set. Parameters used were: windows of size 100, threshold distance of 40 and a minimum pivot distance of 144.

4.5 Preliminary Tests on Parameters Affecting Performance of the Algorithm

Several tests were made prior to final experiments in order to deduce optimal parameter values. Further experiments were conducted to deduce how their different values affect performance of the algorithm. Parameters include: size of the window, window overlap, threshold value, minimum distance between a pair of pivots and the optimal pivot selection technique. Optimal parameter values were points where the algorithm yielded best performance with respect to running time, sensitivity and specificity. All preliminary tests were run on Juggernaut with the SANBI benchmarks data set. Each of the experimental setups is described in detail in this section.

- The SANBI benchmarks data set was used for all windows of different sizes. All the other parameters such as threshold value, minimum distance between two pivots and pivot selection technique were kept constant. The overlap parameter, however, was varied to match size of the window being used. Experiments were conducted to deduce size of a window on which the algorithm functions optimally and how different window sizes affect performance of the algorithm. Sequences were divided into overlapping windows of sizes of 200, 100 and 50 with different overlap sizes depending on the window.

- The optimal window size of 100 was found. From this window the optimal overlap parameters were deduced. Window overlap parameters were tested with the aim of determining a size where the algorithm runs optimally with quality clusters. Overlap parameters ranging from sizes 5 to 25 were used with the SANBI benchmarks data set. Window size, threshold value, minimum pivot distance and pivot selection technique were kept constant in this test at sizes of 100, 40 and 144 respectively. After deducing the optimal window size and overlap parameter, the next step was to find the optimal threshold value that produces both optimal running time and quality of clusters with all other parameters kept constant.

- Tests to determine minimum distance between a selected pair of pivots were made on windows of size 100, with the optimal overlap and threshold value. The minimum pivot distance was varied with all other parameters kept constant. The algorithm was run with the SANBI benchmarks data set on Juggernaut.

- There were several pivot selection techniques proposed and the best technique was deduced experimentally using all optimal values found. Pivot selection techniques were run on the same computer with the same parameter settings at their optimal values. The technique that yielded better performance with respect to running time, sensitivity and specificity was selected to be implemented in the algorithm.

The code for the algorithm was implemented in the C programming language. C is a lower level programming language. It was preferred because of its ability to directly manipulate memory. As a lower level language, it is generally faster and gives complete control to the programmer, which was important in developing the algorithm.

4.6 Summary

Major experiments conducted were on the running time of the algorithm, quality of clusters, space utilisation, effect of different data sets and effect of the triangle inequality. The running time experiments were run on Juggernaut, a 3 gigahertz processor and 4 gigabytes of random access memory with 1 megabyte on processor cache memory using the C-series data set. The C-series is a simulated data set with 1% evenly distributed base scaling errors. Memory utilisation of the algorithm was tested on a 3 gigahertz processor with 1 gigabyte of random access memory using the SANBI benchmarks data set and the C-series. Different computers were used because experiments were independent of each other. Preliminary tests were made to deduce optimal size of the window, overlap parameters, a threshold value, a minimum distance between two pivots and a pivot selection technique. ESTs for Drosophila and public cotton were used to test the effect of different data sets on the performance of the algorithm.

Chapter 5

Results

5.1 Introduction

This chapter presents the results and their interpretation of a series of experiments conducted to determine computational complexity and performance of the algorithm as discussed in the previous chapter. Performance of the algorithm was determined by testing memory utilisation, the running time, sensitivity and specificity of generated clusters. Empirical results have shown that the running time of the algorithm is closer to linear, which is less than the mathematically deduced $O(p \log p)$. This is an improvement over quadratic time, which was one of the major motivations for the development of this algorithm. The algorithm is 100% specific but it fluctuates in its sensitivity due to the limitations discussed in chapter six.

A computational analysis of the algorithm is given in section 5.2. Main functions of the algorithm are analysed from which an overall complexity is deduced. Section 5.3 discusses some of the factors that affect the running time of the algorithm. The factors include window size, threshold value, overlap size, pivot selection techniques and nature of the data set being clustered. Performance of the algorithm with respect to running time, sensitivity and specificity is discussed in section 5.4. One of the aims of this research was to determine the extent to which the triangle inequality reduces distance computations during clustering. Results on the triangle inequality are presented and discussed in section 5.5. A summary of the chapter is made in section 5.6

5.1.1 Distance Computations

The algorithm was initially designed to implement edit distance to measure the level of similarity among sequence windows. However, edit distance, was found to greatly slow down the running time of the algorithm due to its quadratic complexity. Alternatively, d2 distance function was implemented. When d2 is computed over sequences, it runs in quadratic time as windows slide in the two sequences being compared in search for pairs of windows with a minimum d2 score. In such cases d2 performs local alignment, where regions (windows) of higher similarity are searched between the two sequences. Nonetheless, in this algorithm, d2 does not search for pairs of windows with minimum scores between two sequences. It computes distance between pairs of windows globally. This takes a linear time, which improves the running time of the algorithm. In this case, global alignment of the windows is performed. Hence, the final implementation of the algorithm has implemented d2 distance function as a measure of similarity between two sequence windows.

5.2 Computational Complexity

This section provides a brief analysis of the complexity of functions of the algorithm from which an overall complexity has been deduced. The complexity resulting from this mathematical analysis is expected to agree with the complexity obtained empirically.

- **Getting Input**

  Getting input begins by counting the total number of sequences in a data set and this operation takes a linear time of $O(n)$, where n is the total number of sequences. The function counts lines that start with $'>'$ sign, which indicates the beginning of a sequence as previously shown in Figure 2.4. The counting function is called once in the algorithm. The function that gets input sequences is called next and only once in the algorithm. Using the total number of sequences, the algorithm dynamically allocates memory to which sequences are stored in linear time of $O(n)$. Initialising and releasing memory is respectively done once at the beginning and the end of execution in linear time of $O(n)$.

- **Indexing Windows**

  Indexing windows in sequences takes a linear time of $O(p)$, where $p = \frac{n(m-l)}{\triangle} + 1$, l is the size of a window, m is the length of a sequence and n the total number of sequences. Windows are indexed in a sequence from position 0 and after every overlap $\triangle$, up to

$m - l$. Normal size of the window $l = 100$. Since l is very small, the number of windows is computed over a full sequence length of m. With an overlap parameter of size 1, in the worst case scenario, the total number of windows is approximately equal to p. This number reduces as the size of the overlap increases. The function is called once in the algorithm.

- **Distance Computation and Pivot Selection**

  Distance computation performed on a pair of windows is linear $O(l)$ to the size of window, l. However, when reverse complement is computed, the running time is doubled, its complexity remains linear.

  A pivot selection function has $O(p')$ running time, where $1 \leq p' \leq p$. The largest size of a partition is $p = mn$. Searching for the second pivot begins at the first position and ends at the last position in a partition. In the worst case scenario, in a partition of size p, the second pivot is at the last position in the partition, p. However, a partition of size p is encountered at the beginning of the clustering process only and subsequent partitions are of size p'.

- **Partitioning Windows**

  Given a partition of size $0 \leq p' \leq p$, the cost of partitioning windows is the sum of the following costs: pivot selection lp', distance computations lp', comparing windows to find a pivot to which they are closer $2p'$ and sorting windows p' in a partition dividing them into two groups. Windows are sorted in each partition in order to group them with the pivot to which they are closer. Windows closer to pivot one are placed at one end and those closer to pivot two are placed at the other end of the partition. In the worst case scenario, $p' - 1$ windows in a partition belong to one partition, where p' is the total number of windows in a partition and 1 is a pivot of the other partition and this yields a cost of p'. There are several constant operations in the function, which collectively cost c computations. Operations with constant costs include comparison, increment and assignment. Therefore, the partitioning process, costs $c + lp' + lp' + 2p' + p'$. When constants, c, l and 2 are dropped, the total work done on a partition is $4p'$. When a tree has a depth of $\log p$, the algorithm yields an overall partitioning complexity of $O(p \log p)$.

- **Merging Final Partitions**

  In the worst case scenario, there are $p = mn$ total number of leaves. In this case every window is unique and ends up in its own final partition. This case is practically unlikely to occur. In such cases there is no merging work done by the union find data structure. The reason is that merging is done if there are more than one window in a partition. When all windows end up in one leaf, which is another unlikely event, the union-find data structure does $\log p$ amount of work. However, situations that arise during clustering are average cases between the two extreme cases. Analysis of this algorithm is based on the $\log p$ scenario.

- **Overall Time Complexity**

  There are several constant and linear operations in the algorithm. These operations have been respectively discussed under getting input and indexing windows and they are: counting sequences, initialisation of memory and other variables, releasing memory, reading sequences and indexing windows. The total cost of these operations is collectively assigned the value, C. Added to the cost $p \log p$ from partitioning windows and $\log p$ from merging, the overall complexity becomes $C + \log p + p \log p$. Experimental results have shown that the $\log p$ depth is very small and in most cases less than 100. Hence, the dominating term in complexity is p. Since $\log p$ is small, the running time is closer to linear as compared to the expected logarithmic time.

- **Space Complexity**

  Sequences take up an $p = mn$ amount of space, which remains constant throughout the algorithm. When p windows are indexed in the sequences, identification information of the windows is kept in records. Each record has three integer values: sequence index number, starting position of a window and the partition number to which each window belongs during the clustering process. The first two values uniquely identify a window. The amount of space needed to index a windows is $3p$. The total space $(4p)$ required is the sum of space taken up by sequences (p) and space taken up by records, which uniquely identifies windows $(3p)$. When the constant 4 is dropped, it can be deduced that the space complexity is linear in the order $O(p)$

5.3 Parameters Affecting the Performance of the Algorithm

This section discusses how optimal parameter values were experimentally deduced and their effect on the performance of the algorithm. A series of experiments were conducted to determine how these factors affect performance of the algorithm with respect to running time, specificity and sensitivity of generated clusters. Optimum parameter values are points where the algorithm yields best performance. Effects of these parameters on performance are discussed in subsections that follow. The SANBI benchmarks data set was used to illustrate their effect on the performance of the algorithm. This data set was chosen because of its highly skewed nature and which would, therefore, reveal the worst case performance of the algorithm. The size of the data set is larger enough to stress up the algorithm so that both its weaknesses and strengths can be revealed and eventually studied.

Factors affecting the performance of the algorithm were identified and parameter values that enable the algorithm to yield optimal performance were experimentally determined before conducting the final experiments. The parameters identified were window size, overlap size, threshold value, minimum pivot distance between a pair of pivots, pivot selection technique and the nature of data set used. In addition to pivot selection techniques, the distance function that determines the level of similarity among sequence windows was found to affect the performance of the algorithm. As previously discussed in section 5.1.1, the algorithm performs poorly with edit distance as compared to d2 distance function.

5.3.1 The Size of a Window

The aim of this experiment was to deduce the optimal size of a window (w) and investigate the effect of different window sizes (l) on performance of the algorithm. Windows of sizes 200, 100 and 50 were used in both the algorithms and *wcd*, the control algorithm. Threshold values and pivot distances were adjusted with respect to size of windows. For each window size overlaps were varied. Window sizes and threshold values were also adjusted in *wcd* to match those in the algorithm.

Results show that the size of the window directly affects the running time and the sensitivity of the algorithm. Larger window sizes improve both the running time and the sensitivity of the algorithm. Tables 5.1, 5.2 and 5.3 illustrate the effects of size of the window on running time and sensitivity of the algorithm. It can be deduced from these tables that the running time is shorter in windows of size 200 as compared to windows of size 100 and 50. When a window size is larger, a smaller number of windows is generated. Thus the algorithm, does less work and runs relatively faster as compared to when window sizes are smaller. The sensitivity is improved because a larger portion of sequences is compared thereby increasing the chance of finding similarity between two sequences. Both running time and sensitivity are poorer in windows of size 50 as compared to the other sizes. There is, however, a consensus among researchers to use windows of size 100 when measuring similarity among sequences [Burke *et al.* 1999]. Hence, results that follow are presented and discussed based on windows of size 100. It is reasonable to

window size	**overlap**	**pivot distance**	**threshold**	**running time** seconds	**sensitivity** %	**specificity** %
200	100	200	80	37.20	40	100
200	90	200	80	42.60	45	100
200	80	200	80	46.20	47	100
200	70	200	80	50.40	50	100
200	60	200	80	59.40	53	100
200	50	200	80	67.80	61	100
200	40	200	80	85.20	58	100
200	30	200	80	117.60	68	100
200	20	200	80	168.60	74	100
200	10	200	80	357.00	80	100
200	5	200	80	705.60	83	100

Table 5.1: Algorithm performance on a window of size 200

window size	**overlap**	**pivot distance**	**threshold**	**running time** seconds	**sensitivity** %	**specificity** %
100	25	144	40	174.00	66	100
100	20	144	40	213.00	70	100
100	15	144	40	288.00	73	100
100	10	144	40	440.40	76	100
100	5	144	40	912.00	79	100

Table 5.2: Algorithm performance on a window of size 100

window size	**overlap**	**pivot distance**	**threshold**	**running time** seconds	**sensitivity** %	**specificity** %
50	25	50	20	180.00	3	99
50	20	50	20	244.82	4	99
50	15	50	20	316.40	7	99
50	10	50	20	473.34	11	94
50	5	50	20	1003.98	3	89

Table 5.3: Algorithm performance on a window of size 50

use a window size of 100 because some sequences are shorter. Moreover, extending the window size would have resulted in skipping a large number of sequences during the clustering process.

5.3.2 The Overlap Parameter

The aim of this experiment was to deduce the optimal size of the overlap parameter and its effect on performance of the algorithm. Overlap parameters were varied in a 100 size window while pivot distance and threshold values were kept constant. Window size and threshold values were also adjusted to 100 and 40 in *wcd* to make a meaningful comparison.

As shown in Tables 5.1, 5.2 and 5.3 results reveal that overlap parameters have a great effect on the performance of the algorithm with respect to the running time and quality of generated clusters. Memory utilisation of the algorithm is also greatly affected by the size of the overlap. These three tables have shown that as the overlap parameter increases, the running time improves as the sensitivity gets poorer. Larger overlaps result in generation of fewer number of windows, hence less work is done in the clustering process resulting in shorter running times. A smaller number of windows implies that fewer comparisons are made among sequences resulting in poorer sensitivity. Sensitivity also gets poorer because windows of the control algorithm, *wcd* overlap by one. Hence, the algorithm works on less information to measure similarity of windows, compromising the quality of generated clusters.

5.3.3 The Threshold Value

To determine the optimal threshold value and how it affects the performance of the algorithm, its values were varied in windows of size 100. Optimal values of the overlap and minimum pivot distance were kept constant. Window size and threshold values were adjusted in *wcd* at each run to match values in the algorithm.

Experimental results have shown that threshold size directly affects the running time and the sensitivity of the algorithm. It can be deduced that running time of the algorithm generally reduces as the size of threshold increases as shown in Table 5.4. When threshold size is larger, the algorithm stops partitioning windows much earlier resulting in premature clusters. This reduces the amount of work done by the algorithm. However, sensitivity is poorer because the merging of windows is performed on groups of windows that have not been fully compared. A smaller threshold size results in overclustering. Legitimate clusters are divided further into smaller sizes, which lead to the lower sensitivity. The running time increases because the algorithm does more work. It can be deduced from Table 5.4 that both smaller and larger threshold values negatively affect the sensitivity of the algorithm. A proper threshold value is required to efficiently cluster sequences. For purposes of this research, a default threshold value of 40 is used because it is also used as a default value in prior clustering algorithms such as *wcd* and d2_cluster algorithms. The threshold value has also proved to produce consistently better results in this algorithm.

window size	**overlap**	**pivot distance**	**threshold**	**running time** seconds	**sensitivity** %	**specificity** %
100	25	144	5	174.54	39	100
100	25	144	10	174.36	62	100
100	25	144	20	174.18	72	100
100	25	144	30	174.00	69	100
100	25	144	40	174.00	66	100
100	25	144	50	174.34	50	100
100	25	144	60	174.22	18	100
100	25	144	70	174.98	10	100
100	25	144	80	174.92	10	100
100	25	144	90	174.80	3	100

Table 5.4: Effect of the threshold value on performance of the algorithm

5.3.4 Pivot Selection

This experiment aimed at finding an efficient pivot selection technique and the optimal minimum distance between a pair of selected pivots and how the two factors respectively affect the performance of the algorithm. Pivot selection has a significant impact on the running time and the sensitivity of the algorithm. Pivot selection affects performance of the algorithm in two ways: pivot selection criteria and the minimum distance between the first and the second pivot.

Subsections that follow discuss results from the experiments conducted. The proposed pivot selection techniques are described, then results on the performance of the algorithm are presented from which the optimal technique is deduced. This is followed by an investigation on the optimal minimum pivot selection technique and how distance between pairs of pivots affect performance of the algorithm.

Pivot Selection Criteria

The algorithm works by recursively dividing a data set of sequence windows into two partitions. Windows are grouped with the pivot to which they are closer. Results used in this section are generated from a subset data set of the SANBI benchmark sequences. This subset data was found to clearly illustrate the differences in effects of different pivot selection techniques. It was also reasonable to run on a smaller data set because some of the pivot selection techniques run in quadratic time, which takes longer on entire 10000 sequences. Windows size of 100 were used with a threshold size of 40. The overlap parameter values used were 25, 20, 15, 10 and 5. A minimum pivot distance of 144 was used in the experiment. The first pivot selection technique was run and the rest were compared to it. The following are the pivot selection techniques experimented:-

Given X partitions, for every partition P_x of size mn, where $0 \leq x \leq X$, there exists a window $r_i \in P_x$, such that $0 \leq i \leq mn$ and mn, the total number of windows in X. Pivots were chosen as follows:

1. The first pivot selection technique works as follows. Select the first window $(r_i = 0) \in P_x$ as the first pivot p_1. The second pivot p_2 is chosen at an interval $1 \leq i \leq mn$. Pick any window at this interval whose distance $d(p_1, r_i) \geq pDistance$. Pivot distance *(pDistance)* is the minimum value from which the choice of the second pivot begins. If there is no pivot at a distance greater or equal to ***pDistance*** then pick a window with the longest distance in the partition as p_2.

2. In the second technique, select the last window $r_i = mn$ in a partition as the first pivot p_1. The second pivot p_2 is selected as described in the first technique at an interval $0 \leq i \leq mn - 1$ such that $d(p_1, r_i) \geq pDistance$.

3. In the third technique, choose pivots as in the first technique. Repeat the process when resulting partitions are not balanced, for example, when one partition is three times bigger than the other.

4. In the fourth technique, choose $x < mn$ pivots using technique 1. Pick the first window in a partition as p_1, then define a distance $d' = \theta \times f$, where the threshold value θ is multiplied by $1 \leq f \leq 5$. This results in more than two partitions. All windows at distance $\leq d'$ are clustered with p_1. The process is repeated on the remaining unclustered windows until there are no more windows left. The result is that all windows are grouped into their initial clusters. The clustering and pivot selection proceed on the resulting initial clusters using the first technique.

Experimental Results for Pivot Selection Techniques

Experimental results show that the first pivot selection technique (1) yields better results with respect to both the running time and quality of clusters as compared to the second (2), third (3) and fourth (4) techniques. Therefore, the first technique was implemented in the algorithm and all results presented so far are based on it. Results from the other three techniques are presented and contrasted to the first technique in the discussion that follows.

The second pivot selection technique yields poorer performance results as compared to the first technique. The results are shown in Table 5.2. In this technique, both the first and second pivots were randomly chosen. This contributes to the poorer performance achieved when the technique is used. The reason is that when the choice of the first pivot is fixed, clusters build up around the first pivot as the pivot distance reduces between subsequent selected pairs of pivots. This process is scrambled when a different pivot, p_1 is picked randomly. This affects the sensitivity because the clusters that were building up around p_1 break up and some of which may never get merged again.

The third pivot selection technique has several shortcomings. It fails to yield results that are better than those from the first technique. When unbalanced partitions are discovered using the first technique, the process is repeated with the same technique. These trials are made three times and the clustering proceeds regardless of whether the balanced partitions were found or not. The algorithm does more work as it balances partitions, which consequently increases its running time. However, in most cases the algorithm ends up choosing the same pivots picked in the three previous attempts. Eventually, the algorithm reduces to the first technique, yet with increased running time.

Results of the fourth technique are not any better as compared to the first technique. The technique suffers several setbacks. A shorter distance, for example, $d' = \theta \times 2$ between the two initial pivots yields smaller clusters. The sizes of resulting clusters are less than ten on average. In the worst case scenario, where every window is unique, each cluster has a size of one window. For a larger number of windows, the running time of the algorithm turns to quadratic. On the contrary, when d' is large, the technique results in one huge cluster of windows in addition to few others of sizes less than an average of ten. Further clustering continues using the first technique.

Both the running time and the sensitivity of the first and fourth pivot selection techniques were compared and contrasted and results are presented in Tables 5.1(a) and 5.1(b). The label *tech.4a* in Figure 5.1 represents the experiments using the fourth pivot selection technique where initial pivots were selected and windows at a distance equal to or greater than twice the threshold value were clustered. The label *tech.4b* refer to the experiments as described for label *tech.4a* where windows were initially clustered to pivots at a distance three times the threshold value. Since the distance for the initial pivots is larger, one big cluster and other smaller ones were generated. This implementation eventually equates to the first pivot selection technique because the clustering process proceeds on resulting clusters using the first technique. Therefore results of *tech.4b* are similar to those of the first technique. The running time of *tech.4a* is close to quadratic as shown in the graph in Figure 5.1(a) because many smaller clusters are generated on which further clustering is done.

Pivot Distance

Experiments were conducted to determine the effect of the minimum distance between pairs of pivots using the first pivot selection technique. Windows size of 100 were used, overlap parameters took the following values: 25, 20, 15, 10 and 5. A threshold size of 40 was used and the minimum distance at which pivots are chosen was varied for each overlap. Results as indicated in Table 5.5 showed that pivot distance directly affects the running time of the algorithm.

Smaller pivot distances enable the algorithm to choose pivots much quicker as opposed to larger pivot distances. The algorithm does less work and consequently executes faster. In cases where the pivot distance is larger, the worst case scenario in searching for pivots occurs. The algorithm searches for the second pivot up to the last position in the partition. If it does not find any, it selects a window with the largest distance as the second pivot. This is computationally expensive. It was assumed that a good minimum pivot distance should be twice the threshold value ($pDistance = 2\theta$). This assumption was based on the fact that the nearest centre of the next similarly sized cluster should be at a distance, which is at least twice the radius of the cluster. The minimum pivot distance is calculated as follows. A square root of the default threshold value of 40 is approximately 6. This value is multiplied by 2 and squared to get 144. The minimum pivot distance has not been found to affect the sensitivity of the algorithm.

window size	**overlap**	**pivot distance**	**threshold**	**running time** seconds	**sensitivity** %	**specificity** %
100	25	80	40	174.78	68	100
100	25	100	40	175.14	69	100
100	25	120	40	173.64	66	100
100	25	140	40	173.64	66	100
100	25	160	40	175.08	67	100
100	25	180	40	247.44	67	100
100	25	200	40	578.58	68	100

Table 5.5: Effect of pivot distance on the performance of the algorithm

5.3.5 Effects of Data Set Used

Experiments were conducted with the algorithm on three different data sets, the SANBI benchmarks, public cotton and Drosophila melanogaster. The aim of the experiments was to deduce the performance of the algorithm in different data sets. Details of these data sets are found in section 4.3. The type of data set used has a impact on the sensitivity of generated clusters. Similarity among sequences in the data sets may vary from being balanced to highly skewed. Different data sets have different proportions and distribution of different types of errors, which basically define the nature of a data set and also affect performance of the algorithm. Results on the performance of the algorithm with respect to the running time, the sensitivity and specificity on the three different data sets are presented in Tables 5.2 (section 5.3.1, for the SANBI benchmarks data set), 5.6 and 5.7.

window size	**overlap**	**pivot distance**	**threshold**	**running time** seconds	**sensitivity** %	**specificity** %
100	25	144	40	829.20	83	100
100	20	144	40	1043.40	89	100
100	15	144	40	1408.80	83	100
100	10	144	40	2131.80	91	100
100	5	144	40	4525.80	92	100

Table 5.6: Performance of the algorithm on the public cotton data set

window size	**overlap**	**pivot distance**	**threshold**	**running time** seconds	**sensitivity** %	**specificity** %
100	25	144	40	2332.80	40	100
100	20	144	40	3057.60	51	100
100	15	144	40	4108.20	47	100
100	10	144	40	6061.20	63	100
100	5	144	40	12453.60	65	100

Table 5.7: Performance of the algorithm on the Drosophila melanogaster data set

It can be deduced from the results presented in Tables 5.2, 5.6 and 5.7 that the sensitivity of the algorithm differs from one data set to another. The algorithm generated clusters of higher quality on the public cotton data set as compared to the SANBI benchmarks and Drosophila melanogaster data sets. However, the nature and structure of data sets on which the algorithm performs better has not been experimentally determined. Hence, the extent to which the nature of a data set affects the running time remains not clear. Further work is recommended in this area to deduce the nature of a data set on which the algorithm performs optimally with respect to the running time and the sensitivity of generated clusters. Determining such a data set requires simulating different types of errors and distributing them differently in data sets on which the algorithm is tested.

When *wcd* was run on SANBI benchmarks, Drosophila melanogaster and public cotton data sets, it took 237.60, 22220.22 and 3238.8 seconds of the running time respectively. Comparing these running times with a smallest pivot of 5 used in the algorithm, *wcd* is faster on the SANBI benchmarks and public cotton data set but slower on Drosophila melanogaster. On bigger data sets, the algorithm is much faster as compared to *wcd*, which is one of the major achievements of this research.

5.4 Performance of the algorithm

The overall performance was determined by the running time of the algorithm and the quality of generated clusters. Quality of clusters is defined bythe sensitivity and specificity, which are deduced by comparing clusters generated by the tsi_cluster algorithm to those generated by *wcd* version 0.2.3.4 algorithm [Hazelhurst 2003]. In addition to measuring quality of clusters, average running times of the two algorithms were also compared and contrasted with parameters in both algorithms equally adjusted to achieve a meaningful comparison.

5.4.1 Specificity

Experimental results have shown that specificity of the algorithm on windows of size 100 is 100%. This guarantees that the algorithm clusters together sequences that are part of the same expressed gene. In other words, it never clusters sequences together that do not belong to the same cluster. Results have shown that with windows of size 100, changes in sizes of the overlap parameter, threshold value and pivot distance do not negatively affect specificity of the algorithm as long as these parameters are equally adjusted in the control algorithm. Tables 5.1 and 5.2 show a constant specificity of 100% on windows of sizes: 200 and 100, with varied overlap parameter values. A variation is, however, observed in windows of size 50 as shown in Table 5.3. Since the specificity of the algorithm is always 100%, in windows of size 100, focus on performance assessment of the algorithm was placed on the sensitivity and not specificity.

5.4.2 Sensitivity

Experimental results have revealed that the sensitivity of the algorithm fluctuates depending on a data set. This is due to the limitation of the algorithm implementation discussed in section 6.4. The Sensitivity is highly affected by variations in size of the window, pivot distance, threshold value and size of the overlap parameter as shown in Tables 5.1, 5.2, 5.3, 5.4 and 5.5.

5.4.3 The Running Time

The running time results captured in Figure 5.3 have shown that the complexity of tsi_cluster algorithm is closer to linear. The mathematically analysed complexity of $O(p \log p)$ provides an upper bound on the running time. Experimental running time results for all overlap parameters are found in Appendix C. Graph fitting was done using linear and multiple regression to determine the closest function that defines graph lines at different values of overlap parameters. Linear regression is a statistical technique that attempts to model relationships between two variables by fitting a linear equation on the observed data [Sheskin 2004]. Multiple regression is an extention of this technique where more than two variables are involved in defining the relationship line. The regression line represents the best prediction of the observed value (the running time) given the independent value (number of sequences). Standard errors are the standard deviations of error margins between observed values and the regression line. Equations for graph lines for the algorithm were closer to the linear function as compared to logarithmic and quadratic functions. The regression standard error margin of linear equations were much smaller as compared to quadratic and logarithmic equations as shown in Appendix A. The logarithmic function had the second best graph fit and the quadratic function was last. This showed that the running time of the algorithm is closer to linear.

A possible reason for the difference between the mathematical analysis and empirical results is that the algorithm does not get $\log p$ depth. The mathematical analysis assumes that a balanced tree is generated and at most it has a depth of $\log p$. However, reaching this depth means that each leaf contains exactly one window. This may not be the case because for clustering to work, more than one window per leaf is required from different sequences. The windows in the final partitions (leaves) enable the algorithm to successfully cluster sequences. In cases where all sequences are not similar in a data set, there can still be more than one window from the same sequence per leaf. Since there are more than one window in leaves of the tree, the depth of the tree is less than $\log p$. The mathematical analysis, therefore, provides an upper bound on the running time of the algorithm and the two results do not necessarily contradict. Size of the overlap parameter is found to have an effect on the slope of the graph. Smaller overlap parameter sizes results in steeper slope while bigger overlap parameter sizes yield slopes, which are relatively less steep.

5.4.4 Memory Utilisation of the Algorithm

Memory utilisation of the algorithm results as presented in Figure 5.4 shows that bigger overlap parameters yield lower memory utilisation while smaller overlap parameters result in higher memory utilisation. The algorithm generally utilises less amount of memory on overlap parameters greater than 15 as compared to *wcd*. An overlap parameter of 1 is not used in the algorithm because both the memory requirements and the running time are prohibitive. The difference in the sensitivity between overlap parameter sizes of 1 and 5 is very small. With a window overlap parameter size of 1, the algorithm takes a running time of 60.99 minutes, sensitivity of 81% and specificity of 100% to cluster the SANBI benchmarks data set. When an overlap parameter size of 5 is used, the algorithm takes 15.20 minutes, 79% sensitivity and 100% specificity. The increase of 2% in sensitivity is very small considering the difference of 45.79 minutes in the running time. Memory requirements increase at a linear rate in both algorithms as the number of sequences increases. However, the rate is higher in the algorithm as compared to *wcd*.

5.5 The Triangle Inequality

The triangle inequality was implemented in the algorithm to reduce the number of distance computations and evaluations made during the clustering process. Instead of computing distance between the two pivots being compared, distances already computed in the previous recursive call was used to evaluate their level of similarity and partitioned accordingly. This helped to reduce the number of distance computations made and eventually improve the performance of the algorithm. This experiment aimed at investigating the extent to which the triangle inequality holds and whether its application improves performance of the algorithm.

A 1% occurrence of the triangle inequality was found in the SANBI benchmarks, Drosophila melanogaster, public cotton data sets. There was a 2% occurrence on a subset data set of the SANBI benchmarks for all trials with different overlap parameter sizes. The SANBI benchmarks subset data set provided a clear picture of the behaviour of the algorithm with the triangle inequality implementation. Results of the implementation on the subset data set as shown in Figure 5.5 revealed an improvement in the sensitivity of the algorithm but an increase in the running

Overlap	Windows	Trials	Successes	Percentage %
25	5193	239562	4863	2
20	6427	281642	5952	2
15	8493	416923	8004	2
10	12613	658806	12073	2
5	24987	1508501	24192	2
1	124006	6271853	134926	2

Table 5.8: The triangle inequality success proportions

time. A possible explanation for the increase in running time is that the triangle inequality computation is an expensive operation. The number of times that it successfully holds is smaller as compared to the total number of trials made. Hence, the cost of computing and deducing the triangle inequality exceeds its benefits. Increase in sensitivity can be explained by the assumption that the triangle inequality evades occurrence of the algorithm limitation as discussed in chapter six in the few times that it holds. Table 5.8 illustrates the proportions that the triangle inequality held on the subset data set of the SANBI benchmarks.

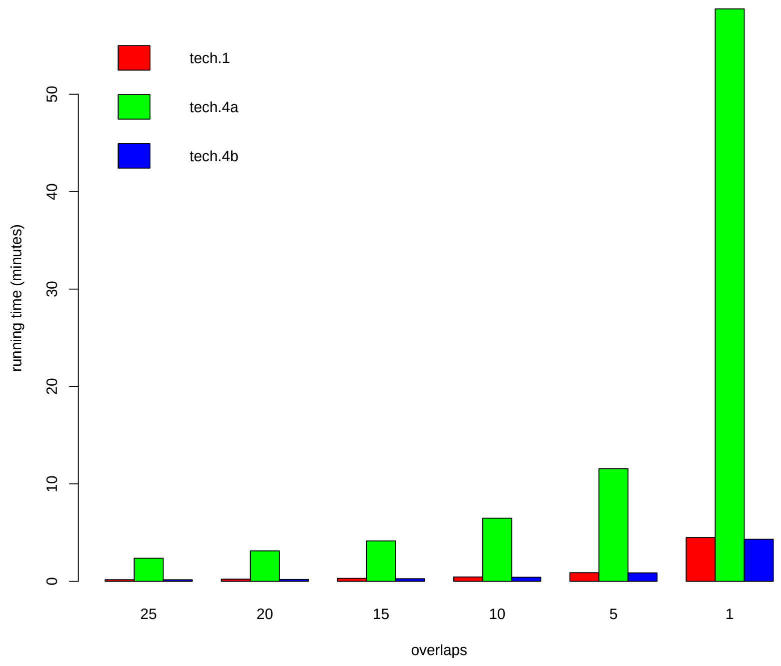
5.6 Summary

This chapter discussed the experimental results. Results on performance of the algorithm with respect to the running time, memory utilisation, the sensitivity and specificity of generated clusters have been presented in the chapter. Prior to major experiments, preliminary experiments were conducted to determine parameters that affect performance of the algorithm.

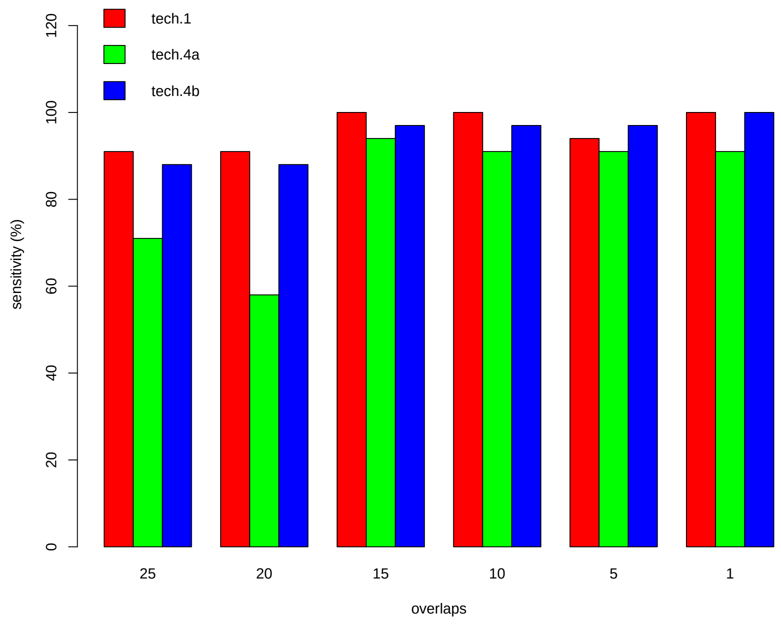
Experimental results showed that the algorithm has a running time closer to linear, which is less than the mathematically analysed $O(p \log p)$ running time. Initially, the algorithm was designed to use edit distance in measuring levels of similarity among sequence windows. But since edit distance has a quadratic complexity, it creates a bottleneck in the running time of the algorithm. As a result, d2 distance function was implemented globally on windows to run in linear time. To deduce quality of clustering, generated clusters were compared to those of *wcd*.

The following parameters were found to affect performance of the algorithm: window size, overlap parameter, threshold value and the type of pivot selection technique used. Larger window sizes were found to improve both the running time and sensitivity of the algorithm. Larger overlap parameters yield shorter running times with poorer sensitivity while smaller overlap parameters results in longer running time but improves sensitivity of the algorithm. Both bigger and smaller threshold values result in shorter running time with poorer sensitivity. The type of pivot selection technique used highly affects running time and quality of clustering of the algorithm. The first pivot selection technique works by selecting the first window in a partition as the first pivot and sequentially searches for the second pivot. The second pivot is a window found at distance greater or equal to the minimum pivot distance from the first pivot. This technique works more efficiently than all the other techniques attempted. The minimum distance at which a second pivot is selected has a tremendous effect on the running time of the algorithm but has been found to have no effect on the sensitivity of the algorithm.

The triangle inequality was tested in order to determine whether it holds and the extent to which it can reduce the total number of distance computations and evaluations made during clustering in order to improve running time of the algorithm. Results have revealed that its success proportion over the total number of trials made is smaller. As such the cost of computation is greater than the intended benefit resulting in increased running times. An improvement in sensitivity was noted with the triangle inequality implementation. Possibly the triangle inequality evades the algorithm limitation as discussed in chapter six.

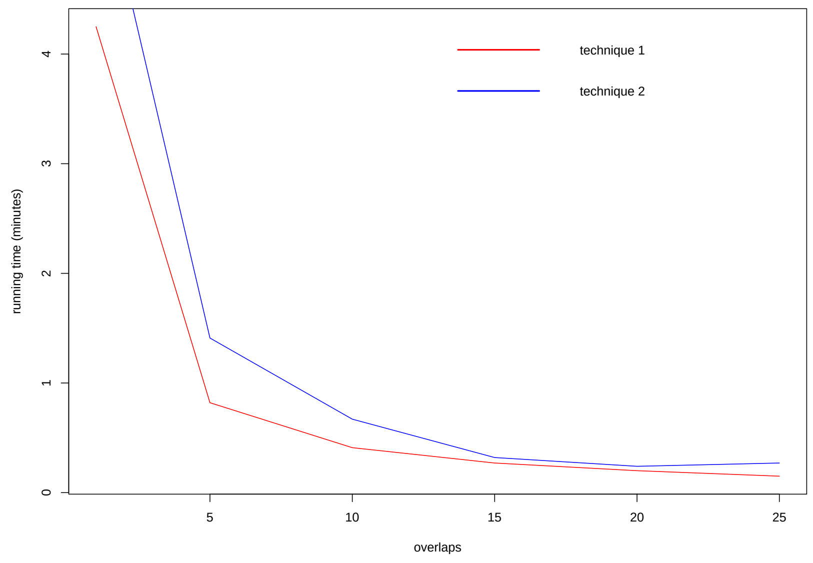


(a) Running Time

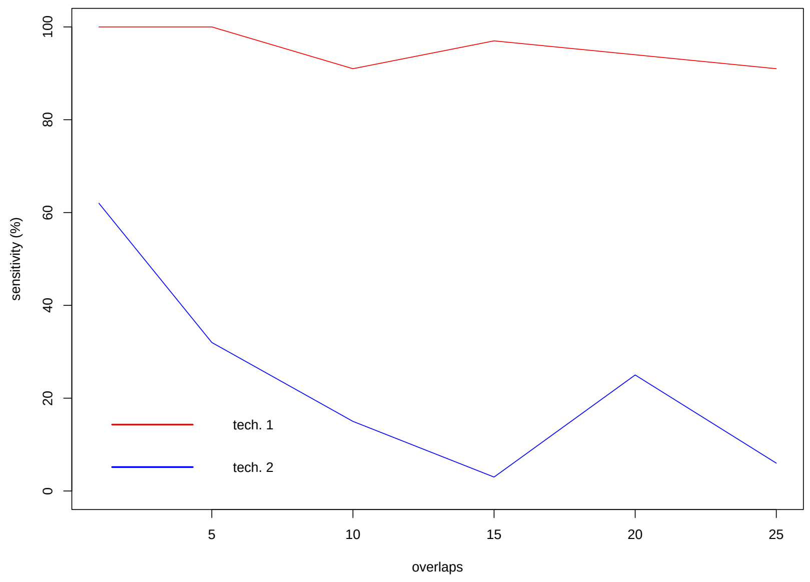


(b) Sensitivity

Figure 5.1: Comparing running time and sensitivity of pivot selection techniques one and four



(a) Running Time



(b) Sensitivity

Figure 5.2: Comparing running time and sensitivity of pivot selection techniques one and two

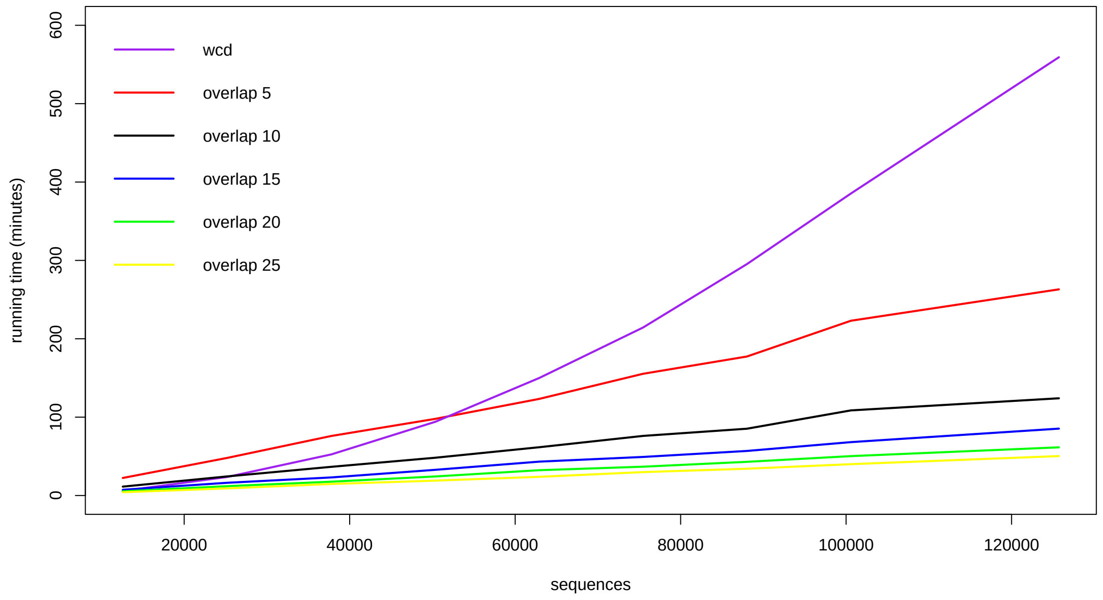


Figure 5.3: An $O(p \log p)$ running time of the algorithm

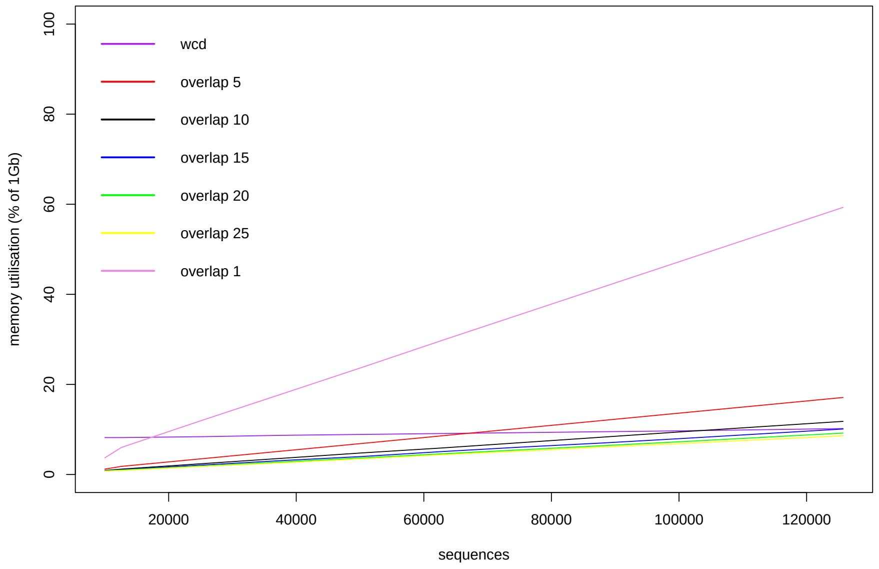
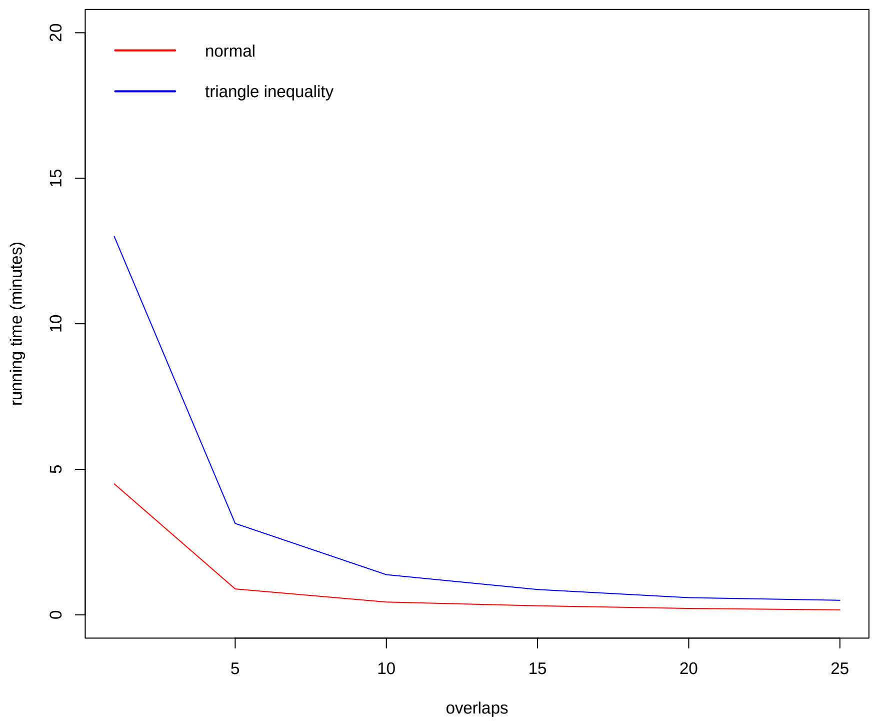
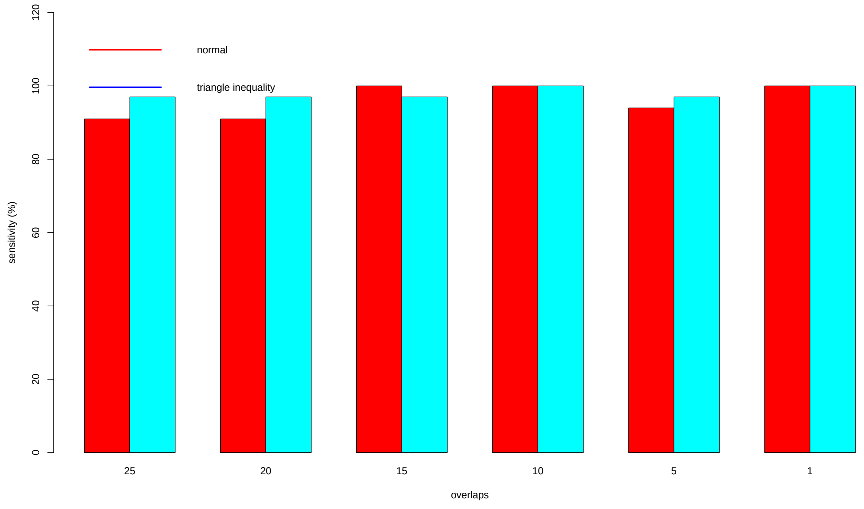


Figure 5.4: Memory utilisation of the algorithm



(a) Running Time



(b) Sensitivity

Figure 5.5: Comparing running time and sensitivity of the triangle inequality implementation

Chapter 6

Discussion

6.1 Introduction

This chapter analyses the overall results with respect to the two research questions. Results show that the algorithm achieves 100% specificity but fluctuates in its sensitivity. This is due to the fundamental limitation in the design of the algorithm. The focus of this chapter is on the fundamental limitation of the algorithm and various attempts made to solve it. It is hoped that based on this discussion, further work can be done to solve the limitation.

In spite this fundamental limitation, the implementation of this algorithm positively contributes to the EST clustering field as discussed in section 6.2. Overall findings with respect to the research questions are presented in section 6.3. The fundamental limitation of the algorithm is explored in section 6.4 and its possible solutions in section 6.5. The chapter is summarised in section 6.6.

6.2 Contribution of the Research

An experimentally deduced running time closer to linear is one of the major contributions of this research to the field of EST clustering. Although this running time does not make much difference in smaller data sets, it does on bigger data sets where efficiency is crucial. Its running time is much faster than what quadratic algorithms can achieve on bigger data sets. This is important considering the increase in the number of ESTs being sequenced and deposited in public databases. Efficient EST clustering of bigger data sets implies efficiency in identification and discovery of expressed genes. Consequently, this will facilitate efficiency in gene expression studies, disease diagnostics, drug discovery, genetic engineering and many other areas that rely on the discovery of expressed genes.

The approach taken by this research in clustering ESTs, where EST sequence windows are mapped into a pseudometric space and clustered by recursively partitioning the space into two has never been applied in any research before. Therefore, this research opens up a new way of clustering from which other techniques can be developed. Specific to clustering in pseudometric space, the triangle inequality has been examined and results reveal that it holds in rare cases. The cost-benefit analysis of implementing the triangle inequality in clustering is very low. This is one of the contributions of this research since no work has been done before to investigate the application of the triangle inequality in clustering.

6.3 Answering the Research Questions

The two research questions that rose from the implementation of the algorithm are:-

1. Can a tree-structured index algorithm cluster ESTs in $O(p \log p)$ running time?
2. Can a tree-structured index algorithm successfully cluster ESTs in a pseudometric space?

In addition to these two research questions, this work was set to investigate the extent to which the triangle inequality holds during clustering. Usage of the triangle inequality properties on sequence distance measures during clustering was expected to assist in reducing distance computations made and hence, speed up the clustering process.

6.3.1 The First Research Question

The need to develop a new algorithm for EST clustering rose because of the prohibitive quadratic running time. The tsi_cluster algorithm was developed to run in $O(p \log p)$. Experimental results have shown that the running time of the algorithm is closer to linear as shown in Figure 5.3. This running time is an achievement in the EST clustering field especially with the increase in the number of ESTs being sequenced and deposited in public databases. Although running time of the algorithm is slower on smaller data sets as compared to *wcd*, there is a considerable difference on bigger data sets for all window overlap parameter sizes. High quality clusters are achieved on bigger data sets within a relatively shorter running time as compared to *wcd* even with the smallest overlap parameter.

6.3.2 The Second Research Question

Results have shown that the tsi_cluster algorithm successfully clusters ESTs in a running time closer linear. To achieve optimal results during clustering, recommended parameter values have to be used. The algorithm runs efficiently with windows of size 100, threshold distance of 40 and minimum pivot distance of 144. Window overlap parameter values can be varied over 25, 20, 15, 10 and 5 depending on the quality of clusters required.

All experiments revealed that the algorithm achieves 100% specificity but fluctuates in sensitivity depending on the data set. This is an assurance that the algorithm always correctly clusters pairs of sequences. However, the fluctuation in sensitivity suggests that the algorithm sometimes fails to put together sequences that belong to the same cluster. This is explained by the major limitation of the algorithm explored in the discussion that follows.

6.4 The Fundamental Limitation of the Algorithm

The principle based on which this algorithm is designed partly contributes to the fundamental limitation of the algorithm. The algorithm divides sequences into overlapping windows. Two pivots are selected from among the windows. All the windows are compared to the two pivots and grouped with either of the pivots depending on their proximity. This process recursively continues until a threshold point is reached or there are no more windows left. Two scenarios are presented to illustrate the occurrence of the limitation during clustering. Consider the following first scenario in Figure 6.1:-

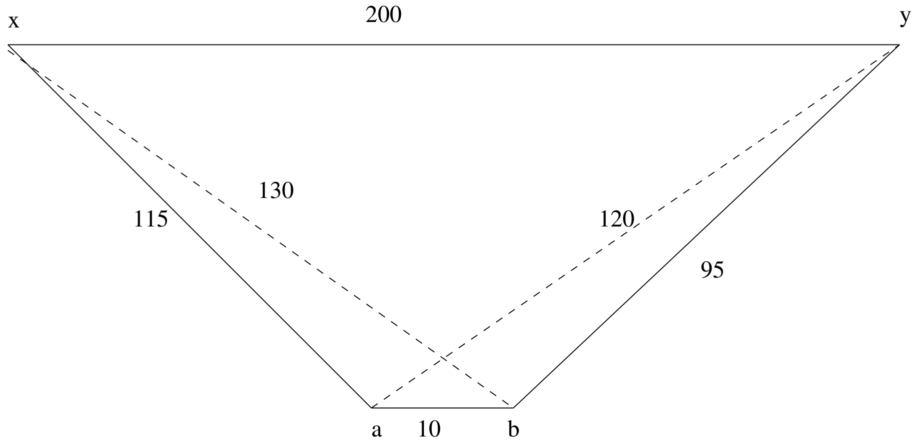


Figure 6.1: First scenario of the limitation of the algorithm

Let x and y be pivots selected in a current partition, P_i. Let a and b, be windows at positions j_1 and j_2 from two different sequences, where $(j_2 - j_1) > 1$. This implies that the two windows are not adjacent to each other in the partition. If the windows have the following distances for example: $d(x, y) = 200$, $d(x, a) = 115$, $d(y, a) = 120$, $d(x, b) = 130$, $d(y, b) = 95$, $d(a, b) = 10$. Let the clustering threshold distance between windows, $\theta = 40$. Notice that all distances are greater than the threshold except for $d(a, b)$.

Windows a and b are required to be partitioned with either pivot x or y. However, the distance between the two windows is less than the threshold value, $d(a, b) < \theta$. This implies that the two windows are similar and require to be partitioned to the same pivot. However, since $d(x, a) < d(y, a)$, window a is partitioned with pivot x. Similarly, since $d(y, b) < d(x, b)$, window b is partitioned with pivot y. The two windows part and never get compared again throughout the partitioning process. They eventually end up in two different clusters after merging if they are the only related windows between the two respective sequences or clusters. The only way that sequences for the two windows can get merged into the same cluster is when there exists another window w in the same leaf with other windows from the two sequences. However, there are situations when such a window does not exist, thereby leaving the two sequences not clustered together. This is a bigger problem when transitivity of other sequences and clusters rely on the relationship of the two windows. It is not only the two sequences that fail to get merged, but many others that rely on the resulting transitivity.

The second scenario of the limitation is shown in Figure 6.2. Let x and y be pivots in the current partition from two different sequences and a be a window to be partitioned with either pivot x or y. Let the distances among the windows for example, be $d(x, y) = 45$, $d(x, a) = 25$ and $d(y, a) = 30$ where the threshold distance $\theta = 40$. It can be observed from this picture that distances $d(x, a)$ and $d(y, a)$ are less than the given threshold θ. Hence, all three windows belong to the same partition. However, since $d(x, a) < d(y, a)$, window a is partitioned with pivot x. Therefore, the two windows part instead of being eventually partitioned together. Other sequences or clusters cannot be merged if their relationship was based on the transitivity generated by the two windows. The scenario in Figure 6.2 occurs when the minimum distance between two pivots in a partition is much closer to the threshold value and in relatively smaller clusters.

With this limitation, a pair of sequences belonging to the same cluster may fail to get clustered together if it encounters such a scenario. When two sequences have a number of similar pairs of windows, this limitation can be avoided because there are several other chances of getting windows from the two sequences compared and partitioned together in the clustering process. However, when there is only one pair of similar windows between the two sequences, once

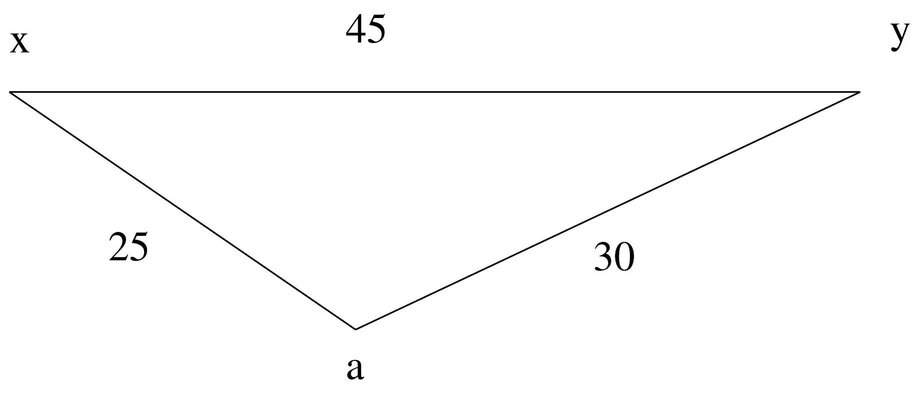


Figure 6.2: Second scenario of the limitation of the algorithm

this scenario occurs, the two sequences do not have another chance of being clustered together. Merging clusters based on transitivity obtained from the relationship of the two sequences fails. This deteriorates the sensitivity of the algorithm as seen in the results. Several attempts have been implemented to resolve this limitation and are discussed in the section that follows.

6.5 Possible Solutions to the Limitation

The first scenario of the limitation, shown in Figure 6.1, is difficult and computationally expensive to detect and resolve. In a partition, one window is compared and grouped to either of the two pivots at a time. Let such a window that is being partitioned be window a. The most feasible way to detect this scenario is to compare all the remaining windows in a partition to window a in order to find other similar windows. Then, compare window a to either of the two pivots and partition all the windows similar to window a to the pivot to which window a is closer. This has to be done for every window being partitioned. Hence, it is very expensive on larger partitions.

A second scenario of the limitation, shown in Figure 6.2, can be handled easily in smaller partitions of sizes less than, or equal to three. The algorithm drops pivots x and y and picks a as the only pivot, when the distance to both pivots x and y from a is less than, or equal to θ. Only one pivot, a is used to avoid this situation reoccurring. This technique was implemented and has several setbacks. The frequency and size of clusters in which this limitation occurs is not yet known. This leads to two major problems when sizes of partitions where this scenario occurs is greater than three. Transitivity is lost between windows that are partitioned with window a and those which are not partitioned with window a. The algorithm is reduced to quadratic when this scenario occurs frequently and in larger partitions, making the running time extremely prohibitive.

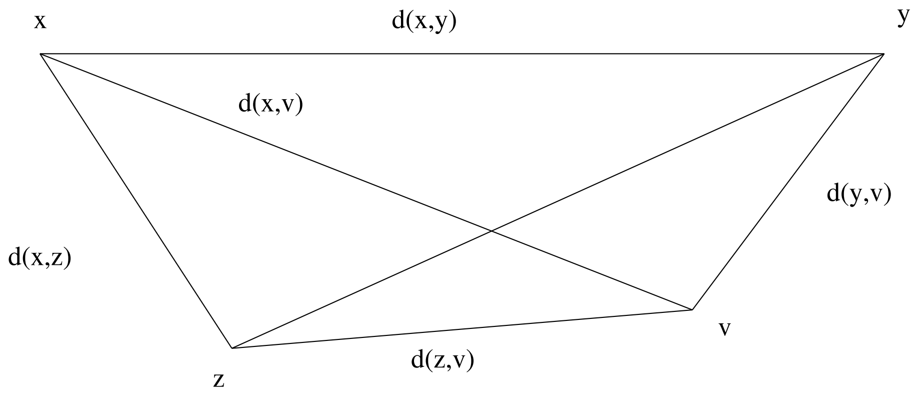


Figure 6.3: First possible solution to the limitation

Two other possible techniques were attempted to resolve this limitation as shown in Figures 6.1 and 6.2. The first approach shown in Figure 6.3 is to choose more than two pivots, x, y, z and v at distances greater than the pivot distance $(pDistance)$, $d(x, y)$, $d(x, z)$, $d(x, v)$, $d(y, z)$, $d(y, v)$ and $d(z, v) \geq pDistance$ and recursively cluster from those pivots. It is feasible to choose two pivots x and y at $d(x, y) \geq pDistance$ apart. However, to get a third pivot z, at $pDistance$ apart from both x and y is computationally expensive. Such a computation gets worse if the number of pivots is greater than three. Choice of pivots and partitioning of windows become computationally expensive because more distance computations are required. The approach of having more than two pivots cannot resolve the limitation as shown in Figure 6.2, because when there are three windows only, the problem returns to its initial state.

The second possible solution shown in Figure 6.4 uses more than one window to represent a pivot. In this case, a pivot becomes an area instead of a single point. As previously discussed, a window is denoted $w_{x,x'}$, where x is an index position for the sequence and x' is the starting position of a window. Indices of the window $w_{x,x'}$ are stored in a record, which is denoted r_j, where j is a position in the record array. During the clustering process, the algorithm sequentially accesses the record from which it indexes the windows.

In this possible solution, more than two windows (in the records array) are selected from the same sequence representing a pivot, $p_1 = \{r_{j_1}, r_{j_2}, r_{j_3}\}$ and the same number of windows from another sequence representing pivot two, $p_2 = \{r_{j_{10}}, r_{j_{11}}, r_{j_{12}}\}$, where $j_1, j_2...$ are positions of window records in a partition. In Figure 6.4, r_{100}, r_{101}, r_{102} and r_{103} are examples of windows from different sequences. Windows in a pivot area are from the same sequence and they are therefore members of the same cluster. A window to be partitioned is compared to all the win-

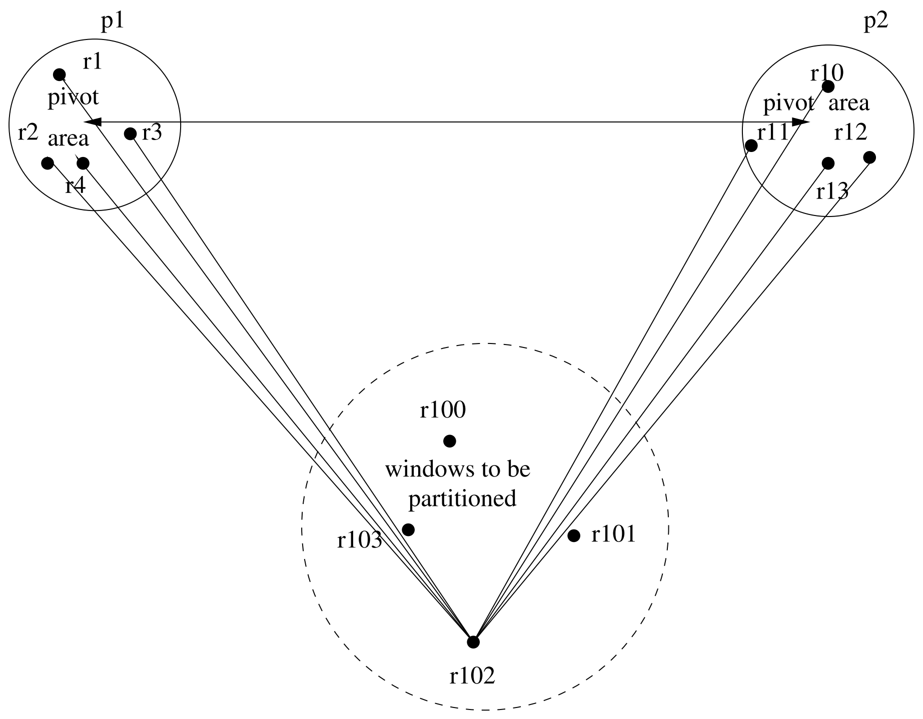


Figure 6.4: Second possible solution to the limitation

dows in p_1 and p_2. For each pivot, a minimum distance is used to determine where to partition a window. For example, window r_{102} is compared to all windows in pivot p_1. The distance between r_{102} and p_1 is the minimum distance between r_{102} and any of the windows representing p_1. The same applies to p_2. The two minimum distances to p_1 and p_2 are compared and window r_{102} is partitioned with the pivot whose minimum distance is smaller.

The approach, however, has setbacks. Partitioning of windows becomes expensive because a window requiring to be partitioned is compared to more windows representing a pivot. In this example, eight distance computations are made to partition r_{102}, which is extremely expensive in bigger data sets. The approach does not resolve the scenario in Figure 6.2 when only three windows remain because the algorithm is forced to define only two pivots to partition the third window.

Therefore, within these limitations, the algorithm successfully clusters expressed sequence tags. The type of user needs and the levels of sensitivity required will dictate when to use this algorithm. Nevertheless, further work is required to resolve this limitation so that the algorithm can be efficiently and fully applied in all types of data sets.

6.6 Summary

This research has come up with a tree-structured index algorithm and it clusters ESTs in a running time closer to linear. The running time has been analysed both mathematically and empirically. The mathematical analysis showed that the algorithm has a logarithmic time complexity, which is an upper bound on the running of the algorithm. The clustering quality of the algorithm is 100% specific but fluctuates in its sensitivity. The poorer sensitivity is explained by the fundamental limitation of the algorithm where the algorithm fails to place similar windows in the same partition. Several attempts have been made to resolve this limitation. The limitation remains unresolved and further work is recommended to resolve it. A running time closer to linear of this algorithm is a remarkable achievement contributed to the EST clustering field. The algorithm can be used for preclustering, or related purposes depending on user requirements. Another important outcome is the finding that the triangle inequality applies during clustering, but only in rare cases. The propotion of its occurrence is very small. This negatively contributes to the running time of the algorithm.

Chapter 7

Conclusion

7.1 Introduction

This chapter summarises the research work and it covers all major issues discussed in previous chapters. Some of the major issues are the research problem, motivation and importance of the research, research questions and how they have been answered, prior work in EST clustering, the tsi_cluster algorithm developed in this research, tests and results found, contribution of this research and further work.

The chapter is divided into six sections. It begins with a summary on the research problem in section 7.2. The same section summarises motivation and importance of the research. The section is followed by a summary on prior work in section 7.3 where an outline of algorithms previously developed is made. The algorithm is summarised in section 7.4 where research questions on the algorithm are also outlined. A summary on findings of this research and limitations of the algorithm are made in section 7.5. Contribution of this research and further work is summarised in the last section.

7.2 Research Problem

The aim of this research was to develop an algorithm that can efficiently cluster EST in $O(p \log p)$ running time. The EST clustering problem is complex because ESTs from different mRNA are mixed up, they are redundant and erroneous. They require to be clustered in huge amounts to achieve meaningful results. The overall goal of solving this problem is to identify expressed gene sequences in an organism. Knowledge of these expressed genes is useful for: disease diagnostics, drug design, microarray design, genetic engineering and many others.

mRNA is a direct gene transcript. It comprises a complete gene coding sequence. It is reverse transcribed into cDNA because cDNA is double stranded and stable for experimental studies. The reverse transcription of mRNA into cDNA results in redundant, erroneous and mixed up bits and pieces known as ESTs. The complete reverse transcribed cDNA represents expressed genes in cells of an organism. The problem, however, is to put these bits and pieces back together in their original order to form a complete cDNA, which eventually reveal expressed gene sequence.

The EST clustering problem is to group these mixed, erroneous and redundant cDNA fragments into groups representing different genes from which they originate. After this step, grouped ESTs are joined back into their original order to form full length cDNA more efficiently. Computationally, EST clustering is a graph problem where ESTs are represented as nodes and the level of similarity between pairs of ESTs are edges. The clustering problem is to find a path in a connected graph that passes through nodes exactly once, where each edge in the graph is less or equal to a defined threshold. Quadratic algorithms achieve this by pairwise distances computations for each node to all the nodes. This is computationally expensive and was one of the major motivations for this research. The idea in this research was to come up with an algorithm that could do the clustering more efficiently in a pseudometric space.

It is important that ESTs are clustered efficiently to speed up gene discovery and all areas that rely on it such as genetic engineering, disease diagnostics, drug discovery and gene expression studies.

7.3 Prior Work

A number of algorithms have been developed for the purpose of EST clustering. Most of the algorithms developed so far run in quadratic time and just a few are outlined. Burke *et al.* [1999] developed d2_cluster algorithm, which runs in quadratic time. Each sequence begins in its own clusters. Then, each sequence is compared to the rest in a pairwise manner and merged if a level of similarity is met. Hazelhurst [2003] developed another quadratic algorithm known as *wcd*. The algorithm works like d2_cluster except that it employs heuristics to reduce distance computations and sequence compression techniques to reduce memory utilisation. The algorithm was parallelised by Ranchod and Hazelhust [2005] to perform clustering on a number of computers simultaneously to improve its running time. A suffix array algorithm for EST clustering that runs in subquadratic time was developed by Malde *et al.* [2003]. All these algorithms were developed in attempt to improve clustering efficiency and this research is part of the same initiative.

7.4 The Algorithm

The prohibitive quadratic clustering running time largely motivated this research and the development of the tsi_cluster algorithm. The algorithm was developed to efficiently cluster ESTs in logarithmic running time in a pseudometric space. It is designed to cluster ESTs by dividing them into overlapping windows. To evade the quadratic complexity, two windows are selected as pivots and the rest of the windows are divided into two groups depending on how close they are to either of the two pivots. This process recursively continues until a threshold point is reached or there are no more windows to partition. Sequences that end up having windows in the same partition are merged into clusters.

The algorithm performs optimally with windows of size 100 nucleotides, a threshold of size 40 and overlap parameter values varying between 5 to 25. A minimum pivot distance of 144 is used which is approximately twice the root of threshold distance squared. The d2 distance function is used to measure the level of similarity among windows. The function was implemented to run in linear time, which successfully contributes to the efficiency of the algorithm. Pivot selection is another crucial factor that affects the performance of the algorithm. Pivots are recursively selected by picking a window at the first position in a partition and the second window at a distance less or equal to the minimum pivot distance from the first pivot. Implementation of the algorithm in such a way initiated research questions, which were answered in determining performance of the algorithm with respect to running time and quality of generated clusters.

The first question was whether the tsi_cluster algorithm clusters ESTs in logarithmic running time. This question was more concerned with time complexity and running time of the algorithm. The question was important because one of the goals of the research was to improve the quadratic complexity in clustering ESTs. The second question asked whether the algorithm successfully clusters ESTs in a pseudometric space. This question focused on quality of generated clusters with respect to sensitivity and specificity. Under this question best parameters that yield optimal performance had to be experimentally defined. The research also investigated whether the triangle inequality can be used to reduce distance computations and evaluations during clustering. Answers to these questions are summarised in section 7.5 based on experiments conducted and analysis of results found.

7.5 Results and Limitations

Experimental results showed that the algorithm has a running time closer to linear. The logarithmic running time, which was deduced through the computational analysis provides an upper bound on this running time. This is because the computational analysis provides an upper bound on the running time. The slope of the running time graph varied depending on the size of the overlap parameter. The running time of this algorithm was compared to *wcd* – a quadratic algorithm and results showed that the quadratic algorithm runs relatively faster on smaller data sets but slower on bigger data sets. The running time of the algorithm was highly affected by parameters such as window size, overlap parameter, threshold value and the minimum pivot distance. The algorithm runs faster with larger window sizes, overlap parameters, threshold sizes and with shorter minimum pivot distances. When window sizes and overlap parameters are larger, few windows are generated which eventually causes the algorithm to do less work. Larger threshold sizes enable the algorithm to run faster because the clustering process stops earlier as compared to smaller threshold sizes. This is because the algorithm does less work and executes quicker. Smaller minimum pivot distance allows the algorithm to choose pivots much quicker causing the algorithm to do less work and execute faster. These parameters have different effects on the quality of clustering, which is the next issue of the summary.

The algorithm achieves a 100% specificity on windows of size 100 for different threshold values, overlap parameters and minimum pivot distance. Sensitivity of the algorithm highly depends on these parameters. In most cases the algorithm fails to get a 100% sensitivity. However, results have shown that sensitivity improves as window size and overlap parameter value decreases. The algorithms performs optimally with windows of size 100, a threshold size of 40 and a minimum pivot distance of 144. Overlaps can be varied over values: 25, 20, 15, 10 and 5. Larger overlap parameters yield lower quality clusters with shorter running times while smaller overlap parameters results in higher quality clusters with longer running time.

The algorithm guarantees that when sequences are clustered together, they indeed originated from the same gene. However, it does not guarantee that two sequences are not similar when they are not clustered together. This is a major limitation of the algorithm. The limitation occurs when a pair of similar windows from two different sequences get partitioned to two different pivots. Consequently transitivity is lost if the pair of windows is the only point of similarity between two sequences, thereby compromising sensitivity of the algorithm.

The proportion of time that the triangle inequality holds in the algorithm is very small. The triangle inequality holds about 1% only of the execution time on the SANBI benchmarks data set. The running time costs of implementing the triangle inequality is very high as compared to the improvement it achieves in the quality of generated clusters.

7.6 Research Contribution and Further Work

This research has successfully developed an EST clustering algorithm that runs in time closer to linear. The algorithm is faster as compared to the quadratic running time on bigger data sets, which was one of the main goals of this work and has been, therefore, successfully achieved. In cases where the algorithm fails to achieve a 100% sensitivity, it is recommended that the algorithm be used for preclustering purposes. This is beneficial on bigger data sets where quadratic algorithms could take prohibitively longer time to cluster. This will eventually improve EST clustering and speed up gene discovery and all areas dependent on it.

Groundbreaking work on the triangle inequality implementation has been done which is one of the major contribution of the research since no studies have been done before in this area. The research has found that implementation of the triangle inequality in EST clustering does not remarkably improve performance of the algorithm.

More work is also recommended to investigate use of the triangle inequality properties in EST clustering. In spite of the fact that the triangle inequality did not yield positive results, this research does not rule out its possible use in clustering. It may work in other implementations.

Further work is recommended to resolve the limitation of the algorithm so that the efficient running time of the algorithm is fully utilised in clustering different types of data sets.

Further work is recommended to determine the nature of data sets on which the algorithm performs better with respect to running time and sensitivity. It has been deduced in this research that the performance of the algorithm differs from one data set to another. Details of the nature of a data set on which the algorithm performs optimally were not explored. It is important to have such details in order to make informed decisions on what type of data sets to use this algorithm in order to achieve quality results. This is also useful on improving the performance of the algorithm on all types of data sets.

References

[Adams *et al.* 1991] M.D. Adams, J.M. Kelley, J.D. Gocayne, M. Dubnick, M.H. Polymeropoulos, H. Xiao, C.R. Merril, A. Wu, B. Olde, R.F. Moreno, A.R. Kerlavage, W.R. McCombie, and J.C. Venter. Complementary DNA Sequencing: Expressed Sequenced Tags and Human Genome Project. *Science*, pages 1651 – 1656, June 1991.

[Burke *et al.* 1999] J. Burke, D. Davison, and W. Hide. d2_cluster: A Validated Method for Clustering EST and Full–Length cDNA Sequences. *Genome Research*, 9(11):1135 – 1142, 1999.

[Bustos *et al.* 2003] B. Bustos, G. Navarro, and E. Chavez. Pivot Selection for Proximity Searching in Metric Spaces. submitted to Elsevier Science (unpublished), 18 March 2003.

[Cameron *et al.* 2004] M. Cameron, H.E. Williams, and A. Cannane. Improved Gapped Alignment in BLAST. *IEEE Transactions on Computational Biology and Bioinformatics*, 1(3):116 – 204, July – September 2004.

[Carpenter *et al.* 2002] J.E. Carpenter, A. Christoffels, Y. Weinbach, and W.A. Hide. Assessment of the Parallelisation approach of d2-cluster for High-Performance Sequence Clustering. *Journal of Computational Chemistry*, 23:1–3, 2002.

[Chavez *et al.* 2001] E. Chavez, G. Navarro, R. Baeza-Yates, and J.L. Marroquin. Searching in Metric Spaces. *ACM Computing Surveys*, 33(3):273 – 321, 2001.

[Cohen 2004] J. Cohen. Bioinformatics – An Introduction for Computer Scientists. *ACM Computing Surveys*, 36(2):122 – 158, June 2004.

[Davison 2001] D.B. Davison. Brute force estimation of the number of human genes using EST clustering as a measure. *IBM Journal*, 45(3), May 2001.

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

[Gusfield 1997] D. Gusfield. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. The Press Syndicate of the University of Cambridge, 1997.

[Hazelhurst 2003] S. Hazelhurst. *The wcd manual: EST clustering using d^2*. The University of the Witwatersrand, July 2003.

[Hazelhurst 2004] S. Hazelhurst. An efficient implementation of the d^2 distance function for EST clustering: preliminary investigations. *Proceedings of SAICSIT 2004*, pages 229 – 233, 2004.

[Hide *et al.* 1999] W. Hide, R. Miller, A. Ptitsyn, J. Kelso, C. Gopallakrishnan, and A. Christoffels. *EST Clustering Tutorial*. South African National Bioinformatics Institute, 1999. Retrieved February 8, 2006, from `http://www.sanbi.ac.za/stack`

[Hunter 1991] L. Hunter. Artificial Intelligence and Molecular Biology. *AI Magazine*, 11(5):27 – 36, January 1991.

[Jain *et al.* 1999] A.K. Jain, M.N. Murty, and P.J. Flynn. Data Clustering: A Review. *ACM Computing Surveys*, 31(3), September 1999.

[Kalyanaraman *et al.* 2003] A. Kalyanaraman, S. Aluru, and V. Bredel. Space and Time Efficiency Parallel Algorithms and Software for EST Clustering. *IEEE Transactions on Parallel and Distributed Systems*, 14(12):1209 – 1221, December 2003.

[Kalyanaraman *et al.* 2005] A. Kalyanaraman, S. Aluru, and S. Kothari. Parallel EST Clustering (unpublished). Computer Science, Iowa State University, 2005.

[Malde *et al.* 2003] K. Malde, E. Coward, and I. Johassen. Fast sequence clustering using a suffix array algorithm. *Bioinformatics*, 19(10):1221 – 1226, January 2003.

[Malde 2005] K. Malde. *Algorithms for the Analysis of Expressed Sequenced Tags*. PhD thesis, University of Bergen, Department of Informatics, 2005.

[Marques *et al.* 2005] A.C. Marques, I. Dupanloup, A. Vinckenbosch, N. Reymond, and H. Kaessmann. Emergence of Young Human Genes after a Burst of Retroposition in Primates. *PLOS Biology*, 3(11):1970–1979, November 2005.

[National Health Museum 2006] National Health Museum. *Graphics Gallery, National Health Museum Resource Center*, April 2006. Retrieved 21 April 2006, from `http://www.accessexcellence.org/RC/VL/GG`

[NCBI] NCBI. *dbEST: database of Expressed Sequenced Tags*. Retrieved 16 Nov, 2007: 9.35 am, from `http://www.ncbi.nlm.nih.gov/dbEST/dbEST_summary.html`

[NIH 2007] NIH. *Mamalian Gene Collection*. National Institute of Health Website, 2007. Retrieved November 5, 2007, from `http://mgc.nci.nih.gov/`

[Ptitsyn and Hide 2005] A. Ptitsyn and W. Hide. CLU: A new algorithm for EST clustering. *BMC Bioinformatics*, 6(2), July 2005.

[Ptitsyn 2000] A. Ptitsyn. *A New Algorithm for EST clustering*. PhD thesis, University of the Western Cape, April 2000.

[Quinn 1987] M.J. Quinn. *Designing Efficient Algorithms for Parallel Processing*, chapter 8, pages 165–170. McGraw-Hill Computer Science Series, 1987.

[Ranchod and Hazelhust 2005] P. Ranchod and S. Hazelhust. *A Distributed System for EST Clustering*. Technical report, School of Computer Science, University of the Witwatersrand, 2005.

[Ranchod 2005] P. Ranchod. *Parallelisation of EST Clustering*. Master's thesis, University of the Witwatersrand, School of Computer Science, 2005.

[Schuler 1997] G.D. Schuler. Pieces of the puzzle: expressed sequence tags and the catalog of human genes. *Journal of Molecular Medicine*, 75:694 – 698, 1997.

[Sheskin 2004] D.J. Sheskin. *Handbook of Parametric and nonparametric statistical procedures*. Chapman and Hall/CRC, third edition, 2004.

[Shi *et al.* 2006] Y. Shi, S. Zhu, X. Mao, J. Feng, Y. Qin, L. Zhang, J. Cheng, Z. Wei, L. Wang, and Y. Zhu. Transcriptome Profiling, Molecular Biological, and Physiological Studies Reveal a Major Role for Ethylene in Cotton Fiber Cell Elongation. *Plant Cell*, 18:661–664, February 2006.

[Trivedi *et al.* 2002] N. Trivedi, J. Bischof, S. Davis, K. Pedretti, T.E. Scheetz, T.A. Braun, C.A. Roberts, N.L. Robinson, V.C. Sheffield, M.B. Soares, and T.L. Casavant. Parallel creation of non-redundant gene indices from partial mRNA transcript. *Future Generation Computer Systems*, 18:863 – 870, 2002.

[Trivedi *et al.* 2003] N. Trivedi, K.T. Pedretti, Braun T.A., T.E. Scheetz, and T.L. Casavant. *Alternative Parallelization Strategies in EST Clustering*, volume 2763/9743, pages pp.384–393. Springer-Verlag Berline Heidelberg, 2003.

[Wheeler *et al.* 2003] D.L. Wheeler, D.M. Church, S. Federhen, A.E. Lash, T.L. Madden, J.U. Pontius, G.D. Schuler, L.M. Schriml, E. Sequeira, T.A. Tatusova, and L. Wagner. Database resource of the National Center for Biotechnology. *Nucleic Acids Research*, 31(1), 2003.

[Yona 1999] G. Yona. *Methods of global organisation of all known protein sequences*. PhD thesis, Computer Science, Hebrew University, May 1999.

[Zhu *et al.* 2003] W. Zhu, S.D. Schueter, and V. Brendel. Refined Annotation of the Arabidopsis Genome by Complete Expressed Tag Mapping. *Plant Physiology*, 132:649 – 484, June 2003.

[Zimmermann 2003] J. Zimmermann. *Suitability Comparison of String Distance Measures for EST Clustering*. Master's thesis, ETH Zurich, Dept. of Computer Science, 2003.

Appendix A

Graph Line Fittings

Overlaps / Equations	5	10	15	20	25
linear error	7.08	2.42	1.12	0.61	0.49
logarithm errors	7.12	2.45	1.13	0.65	0.51
quadratic errors	7.52	2.57	1.08	0.66	0.32

Table A.1: Regression standard errors

Appendix B

Running Time Results

file name	overlap 1	overlap 2	overlap 3	overlap 4	overlap 6
C01	5	10	15	20	25
mean	22.50	11.26	7.20	5.50	4.30
std	0.277	0.147	0.124	.08	0,043
C02					
mean	47.79	24.50	16.55	11.98	9.127
std	0.492	0.232	0.007	0.053	0.101
C03					
mean	76.36	36.685	22.848	17.700	14.77
std	0.511	0.502	0.197	0.0701	0.070
C04					
mean	98.634	48.450	32.658	23.860	19.171
std	0.6837	0.557	0.360	0.268	0.261
C05					
mean	124.51	62.080	42.217	31.977	23.982
std	1.4846	0.6940	0.5572	0.1675	0.2607

Table B.1: Running times mean and standard deviation results

Running Time Results (Continued)

file name	overlap 1	overlap 2	overlap 3	overlap 4	overlap 6
C06					
mean	156.140	74.430	49.561	36.462	29.410
std	1.930	0.9768	0.6361	0.3218	0.1585
C07					
mean	178.243	85.849	57.273	43.113	34.476
std	1.531	0.887	0.5140	0.2461	0.3863
C08					
mean	227.82	106.915	68.833	49.971	39.614
std	9.630	1.5641	0.7166	0.5232	0.4495
C10					
mean	263.679	126.643	85.079	61.931	51.044
std	1.5079	1.7886	0.992	0.5301	0.704

Table B.2: Running times mean and standard deviation results

Appendix C

Running Time Data

These are running time results for ten experiments for different overlap parameter values. Mean and standard deviation (std) are shown for each overlap parameter.

```
Data
set
c01   overlap5  overlap10  overlap15  overlap20  overlap25

      22.7500   11.0600   7.1700    5.5900    4.3100
      22.5100   11.2100   7.0600    5.4600    4.3100
      22.7100     11.2100   7.0600    5.5600    4.2400
      22.1900   11.4700   7.1500    5.3800    4.2900
      22.2400   11.1700   7.1500    5.4500    4.3700
      23.0000   11.2900   7.1800    5.3900    4.2500
      22.8900   11.0300   7.4900    5.4700    4.3100
      22.1800   11.3700   7.1700    5.6000    4.3500
      22.5400   11.4700   7.2200    5.5500    4.2600
      22.4100   11.3600   7.3500    5.3800    4.2400

mean  22.5000   11.2600   7.2000    5.5000    4.3000
std   0.2769    0.1473    0.1239    0.0820    0.0431


c02   47.4700   24.3600   16.2500   12.0600 9.3300
      48.5400   24.3600   16.2500   11.9600   9.0900
      47.2000   24.9400   16.2500   11.9500   9.0700
      48.3400   24.3600   16.2500   11.9500   9.3300
      48.4200   24.6300   15.2500   12.1100   9.0800
      47.2700   24.9600   16.2600   11.9600   9.0700
      47.4700   24.3700   16.2600   11.9600   9.0700
      48.0100   24.3800   16.2600   11.9500   9.0700
      47.2800   24.3900   16.2700   11.9600   9.0800
      47.9400   24.3700   16.2500   11.9600   9.0800
```

```
mean   47.7900    24.5000    16.5500    11.9800    9.1270
std    0.4915     0.2323     0.0067     0.0529     0.1017


c03    76.0300    36.1900    23.0200    17.6800    14.7400
       77.1900    36.1800    22.6700    17.6700    14.7700
       77.1100    36.6300    23.1900    17.6900    14.7500
       77.1200    37.2000    22.6800    17.6800    14.7600
       76.0100    36.7200    22.9600    17.6800    14.9800
       76.0600    36.1800    22.6600    17.9100    14.7600
       76.0400    37.4700    22.8800    17.6700    14.7400
       76.0100    36.1700    22.6600    17.6700    14.7400
       76.0200    37.4700    22.6600    17.6700    14.7400
       76.0100    36.6400    23.1000    17.6700    14.7500

mean   76.3600    36.6850    22.8480    17.6990    14.7730
std    0.5112     0.5024     0.1974     0.0706     0.0697


c04    99.0400    48.4100    32.4100    23.8400    19.0100
       99.3100    49.2900    32.7600    23.9100    19.6900
       99.0100    47.6600    32.4000    23.7700    19.0100
       99.0300    49.3300    32.4400    24.3400    19.2700
       97.8100    48.2100    32.4000    23.8000    19.6500
       97.8100    48.9800    32.4300    23.5600    19.0300
       99.2300    47.6800    32.4000    23.9000    19.0100
       99.4700    48.3900    32.9800    23.5300 19.0100
       97.8000    48.3400    33.5500    23.6100    19.0200
       97.8300    48.2400    32.8100    24.3400    19.0100

mean   98.6340    48.4500    32.6580    23.8600    19.1710
std    0.6837     0.5565     0.3597     0.2682     0.2609


c05    123.3900   61.5700    41.7700    31.9000    24.1200
       123.3400   63.7200    43.2200    31.8900    23.8700
       125.6000   61.5500    41.7500    31.8800    23.8600
       123.5000   61.6300    42.3500    32.3000    23.8900
       127.7200   62.4100    41.7500    31.8800    23.8600
       125.7500   61.6300    41.8100    31.9400    23.8900
       123.4200   61.5500    42.2400    31.8800    23.9000
       123.3600   61.5600    41.7600    31.9000    24.6900
       125.7000   62.5600    42.2700    31.8800    23.8600
       123.3100   61.6400    43.2500    32.3200    23.8800

mean   124.5100   62.0800    42.2170    31.9770    23.9820
std    1.4846     0.6940     0.5572     0.1675     0.2607


c06    155.1200   73.4700    50.5500    36.0800    29.3800
       154.9600   74.9400    49.6600    36.3800    29.3300
```

```
      155.0600  73.4500   50.2200   36.5300   29.4100
      159.2900  74.8700   48.9700   36.1200   29.3600
      160.6000  73.6700   48.8800   36.5300   29.3300
      155.3700  75.9500   49.5400   36.1500   29.3700
      155.3300  73.5200   48.9300   36.1000   29.3500
      155.2000  73.6000   50.5800   36.5200   29.3600
      155.0900  74.7100   49.0400   37.1100   29.3300
      155.3700  76.0800   49.2400   36.8000   29.8800

mean  156.1400  74.4300   49.5610   36.4620   29.4100
std   1.9300    0.9768    0.6361    0.3218    0.1585


c07   177.4600  85.2300   57.6200   43.0200   35.3500
      179.8600  85.2000   56.8400   43.0200   34.2100
      177.4000  86.1200   56.8500   43.0200   34.9700
      182.3200  86.2900   56.9100   43.8400   34.2100
      177.4400  85.2400   57.5200   43.0100   34.6700
      177.7400  85.3000   56.9500   43.0600   34.2100
      177.7100  85.2800   56.9300   42.9900   34.5400
      177.7100  88.1000   58.4200   43.1500   34.2300
      177.4000  86.4700   57.8000   43.0100   34.1700
      177.3900  85.2600   56.8900   43.0100   34.2000

mean  178.2430  85.8490   57.2730   43.1130   34.4760
std   1.5318    0.8870     0.5140   0.2461    0.3863

c08   223.0500  107.1000   70.0300  49.6500   39.8000
      222.9800  104.8900   68.1700  49.6600   39.2100
      222.8900  108.6400   69.0400  49.7300   39.2100
      228.8500  106.0900   68.2300  49.6900   39.2400
      224.2700  104.8700   69.1500  50.3600   39.2200
      224.0200  106.6700   68.3000  51.3400   39.9300
      230.5300  108.6300   68.1700  49.6300   40.5900
      255.6600  108.6100   69.0200  49.7000   39.2300
      222.9200  105.0200   70.0700  49.6700   39.7000
      223.0300  108.6300   68.1500  50.2800   40.0100

mean  227.8200  106.9150   68.8330  49.9710   39.6140
std   9.6300    1.5641     0.7166   0.5232    0.4495

c10   263.5900  124.8300   84.5000  61.4000   50.6100
      263.0700  126.1400   84.4900  61.4100   51.4100
      263.0100  124.8800   84.4700  62.3500   50.3400
      263.4400  128.2300   84.8000  62.3600   51.7500
      263.0200  124.8400   84.8500  62.1500   50.4200
      267.6600  125.2500   87.3800  62.6900   50.6100
      263.0100  128.2900   84.4800  61.3800   51.7200
      263.2700  129.0800   86.0300  62.2600   50.4000
      263.0400  128.2500   84.7100  61.3800   52.1400
```

```
mean  263.6789  126.6433   85.0789  61.9311   51.0444
std   1.5079   1.7886     0.9923   0.5306    0.7042
```