5-5-2017

Novel Algorithms for Big Data Analytics

Subrata Saha
University of Connecticut, subrata.saha@uconn.edu

Novel Algorithms for Big Data Analytics

Subrata Saha, Ph.D.

University of Connecticut, 2017

ABSTRACT

In this dissertation we offer novel algorithms for big data analytics. We live in a period when voluminous datasets get generated in every walk of life. It is essential to develop novel algorithms to analyze these and extract useful information. In this thesis we present generic data analytics algorithms and demonstrate their applications in various domains.

A number of fundamental problems, such as clustering, data reduction, classification, feature selection, closest pair detection, data compression, sequence assembly, error correction, metagenomic phylogenetic clustering, etc. arise in big data analytics. We have worked on some of these fundamental problems and developed algorithms that outperform the best prior algorithms. For example, we have come up with a series of data compression algorithms for biological data that offer better compression ratios while reducing the compression and decompression times drastically. As another example, we have invented an efficient algorithm for the problem of closest pairs. This problem has numerous applications. Our algorithm when applied to solve the two-locus problem in Genome-wide Association Studies performs two orders of magnitude faster than the best-known prior algorithm for solving the two locus problem. As another example, we have proposed a novel deterministic sampling technique that can be used to speed up any clustering algorithm. Empirical results show that this technique results in a speedup of more than an order of magnitude over exact hierarchical clustering algorithms. Also, the accuracy obtained is excellent. In fact, on many datasets, we get an accuracy that is better than that of exact hierarchical clustering algorithms!

Novel Algorithms for Big Data Analytics

Subrata Saha

B.Sc., Bangladesh University of Engineering and Technology, Dhaka, 2009

A Dissertation

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

at the

University of Connecticut

2017

APPROVAL PAGE

Doctor of Philosophy Dissertation

Novel Algorithms for Big Data Analytics

Presented by

Subrata Saha

Major Advisor _____
Sanguthevar Rajasekaran

Associate Advisor _____
Chun-Hsi (Vincent) Huang

Associate Advisor _____
Ion Mandoiu

Associate Advisor _____
Mohammad Maifi Hasan Khan

University of Connecticut

2017

ACKNOWLEDGMENTS

I would like to thank all the people who contributed to the research works portrayed in this thesis. At first I would like to express my sincere gratitude to my academic advisor Professor Sanguthevar Rajasekaran for accepting me into his group and his continuous support during my Ph.D. study. I thank my advisor for giving me intellectual freedom in my research works, and always engaging me in brand new problems and ideas. His supervision, patience, motivation, and enormous knowledge helped me in all the time of my research endeavors and writing of this thesis. Besides my advisor, I would like to thank the rest of my thesis committee: Professor Chun-Hsi (Vincent) Huang, Professor Ion Mandoiu, and Professor Mohammad Maifi Hasan Khan, for their interest, insightful comments, and encouragement.

Finally, I would like to acknowledge friends and family especially my parents and wife for their constant love and support during this long journey.

Contents

Part I

Preface

Chapter 1

Summary of Contributions

The central theme of this thesis is the development of novel data structures and algorithms for big data analytics. We live in a period when voluminous datasets get generated in every walk of life. It is essential to analyze and extract useful information from complex and large datasets (e.g., text, biomedical, and biological data). In this context we have invented generic data analytic algorithms. These algorithms have been applied in varied domains including bioinformatics, biomedical informatics, and materials science. Contents of this thesis have been published in top notch venues including Bioinformatics, BMC Bioinformatics, BMC Genomics, ACM BCB, IEEE ICDM, and ACM CIKM.

Contributions of this thesis are in different areas including applied algorithms, machine learning, data mining, and biological sequence analysis. A brief summary of the thesis follows:

1.1 Sequence Analysis

1.1.1 Metagenomic phylogenetic clusterings

An important problem in analyzing metagenomic shotgun sequencing data is to identify the microbes present in the data and figure out their proportions. Existing taxonomic profiling methods are inefficient especially on large data sets. We have devised a highly accurate and very fast metagenomic phylogenetic clustering algorithm that is based on identifying unique signatures for each of the possible bacterial genomes. Our algorithm can detect genus and species information within the metagenomes with a very high level of confidence. A comparison reveals that our algorithm outperforms the state-of-the-art algorithms with respect to sensitivity, specificity, and runtime.

1.1.2 Spliced junctions discovery

It is a very challenging task to accurately map RNA-seq reads onto the genomic sequence and identify the spliced junctions. There are two shortcomings in some of the best-known algorithms for identifying spliced junctions, namely, the junction boundaries are predicted within a large range, and they take a long time. We have developed a multi-core self-learning algorithm to accurately detect genome-wide splicing events by employing clustering and machine learning techniques. Experimental results show that our algorithm is indeed a more effective and efficient algorithm compared to the other state-of-the-art algorithms in terms of sensitivity, specificity, and runtime. (These results have been presented in *ACM BCB 2016*).

1.1.3 Efficient scaffolding

Due to erroneous base calling, sequence assemblers often fail to sequence an entire DNA molecule and instead output a set of overlapping segments that together represent a consensus region of the DNA. This set of overlapping segments are collectively called *contigs* in the literature. The final step of the sequencing process, called *scaffolding*, is to assemble the contigs into a correct order. In this research we have introduced a series of novel algorithms for scaffolding that exploit optical restriction maps (ORMs). Experimental results show that our algorithms are indeed more reliable, scalable, and efficient compared to the best known algorithms in the literature. (These algorithms have been published in *BMC Genomics 2014*).

1.1.4 Biological sequence compression

We see an explosion of biological sequences in recent times due to the next-generation sequencing technologies. In turn, the cost of storing the sequences in physical memory or transmitting them over the internet is becoming a major bottleneck for research and future medical applications. Data compression techniques are one of the most important remedies in this context. We are in need of suitable data compression algorithms that can exploit the inherent structure of biological sequences. Although standard data compression algorithms are prevalent, they are not suitable for compressing biological sequencing data effectively. Considering this fact we have developed a series of novel compression algorithms to effectively and efficiently compress the biological sequence data (e.g., referential genome compression algorithms and non-referential reads compression algorithms). Experimental evaluations show that our algorithms outperform state-of-the-art algorithms in this domain. (These results can be found in *IEEE BIBM 2014, ISBRA 2015, Bioinformatics 2015, Bioinformatics 2016*).

1.1.5 Error correction

One of the limitations of the modern sequencing technology is the error in the biological sequences (i.e., reads) generated. Any sequence assembler often fails to sequence the entire genome because of the errors in the reads. By identifying and correcting the erroneous bases of the reads, not only can we achieve high quality data but also the computational complexity of many biological applications can be greatly reduced. Considering this fact we have developed k-mer spectrum based error correcting algorithms for Illumina generated short reads to identify and correct substitution errors with a very high level of confidence. They outperform the best-known algorithms in terms of accuracy and run time. (Papers on these algorithms have been published in *BMC Bioinformatics 2015, ACM BCB 2015*).

1.2 Genotype-Phenotype Correlational Analysis

1.2.1 Genotype-phenotype correlation

Single Nucleotide Polymorphisms (SNPs) are sequence variations found in individuals at some specific points in the genomic sequence. As SNPs are highly conserved throughout evolution and within a population, the map of SNPs serves as an excellent genotypic marker. Conventional SNPs analysis mechanisms suffer from large run times, inefficient memory usage, and frequent overestimation. We have employed random projection (RP) and support vector machine (SVM) to efficiently identify the correlation between genotypes (SNPs) and phenotypes (i.e, characteristics such as the presence of a disease). This correlational information and genotypic data of a person can be used to predict if the individual has a specific phenotype or not. Experimental results show that our proposed algorithm is more accurate

than that of multifactor dimensionality reduction (MDR) and principal component analysis (PCA) techniques. (Our algorithms have been published in *BMC Medical Informatics and Decision Making (MIDM) 2013*).

1.2.2 Genome-wide association study

Investigations that try to understand human variability using SNPs fall under genome-wide association study (GWAS). A crucial step in GWAS is the identification of the correlation between genotypes and phenotypes. This step can be modeled as the k-locus problem (where k is any integer). A number of algorithms have been proposed in the literature for this problem. We have developed an algorithm for solving the 2-locus problem based on random sampling and hashing. It is up to two orders of magnitude faster than the previous best known algorithms. For the first time, we have also developed an efficient algorithm for solving the 3-locus problem that is several orders of magnitude faster than the brute force algorithm. (These algorithms have been presented in *IEEE ICDM 2016* and *ACM CIKM 2016*).

1.3 Data Mining and Pattern Recognition

1.3.1 Scalable clustering algorithms

Conventional clustering algorithms suffer from poor scalability, especially when the data dimension is very large. It may take even days to cluster large datasets. For applications such as weather forecasting, time plays a crucial role and such run times are unacceptable. It is perfectly relevant to get even approximate clusters if we can do so within a short period of

time. Considering this fact we have devised and implemented a novel deterministic sampling technique that can be used to speed up any clustering algorithm. Empirical results show that this technique results in a speedup of more than an order of magnitude over exact hierarchical clustering algorithms. Also, the accuracy obtained is excellent. In fact, on many datasets, we get an accuracy that is better than that of exact hierarchical clustering algorithms! (This generic technique has been published in *ADMA 2013*).

1.3.2 Randomized feature selection algorithms

Feature selection is the problem of identifying a subset of the most relevant features in the context of model construction. This problem has been well studied and plays a vital role in machine learning. In this research endeavor we have explored efficient ways of finding the most relevant features from a set of features and invented a series of randomized search methods which are generic in nature and can be applied for any learning algorithm. (These algorithms appear in *DMIN 2013* and *IJFCS 2015*).

1.3.3 Closest pair identification problem (CPP)

CPP is one of the well-studied and fundamental problems in computing. Given a set of points in a metric space, the problem is to identify the pair of closest points. There are numerous applications where this problem finds a place. Examples include computational biology, computational finance, share market analysis, weather prediction, entomology, electro cardiograph, N-body simulations, molecular simulations, etc. As a result, any improvements made in solving CPP will have immediate implications for the solution of numerous problems in these domains. A naive deterministic algorithm can solve CPP in quadratic time. Quadratic time may be too much given that we live in an era of big data. Speeding up

data processing algorithms is thus much more essential now than ever before and we have developed algorithms for CPP that improve (in theory and/or practice) the best-known algorithms reported in the literature for CPP.

1.4 Parts Summary

The rest of the thesis is organized as follows. In Chapter 2 of Part I we provide a list of publications that have resulted from our research. Part II focuses on our contributions to Biological Sequence Analysis. Specifically, in this part we present our referential genome compression algorithms (Chapters 1), non-referential reads compression algorithm (Chapter 2), hybrid error correction algorithm (Chapter 3), genome-wide splicing events detection algorithm (Chapter 4), and scaffolding algorithms (Chapter 5). In Part III we present our genotype-phenotype correlation algorithms. Part IV relates to data mining and pattern recognition. In particular, we describe our deterministic sampling based clustering algorithms in Chapter 1. In Chapter 2 our feature selection techniques are described. Chapter 3 is devoted for the discovery of time series motifs.

Chapter 2

Publications

2.1 Journal Publications

1. S. Saha and S. Rajasekaran: **NRGC: a novel referential genome compression algorithm**, *Bioinformatics*, 2016.

2. S. Saha and S. Rajasekaran: **EC: an efficient error correction algorithm for short reads**, *BMC Bioinformatics*, 2015.

3. S. Saha and S. Rajasekaran: **ERGC: an efficient referential genome compression algorithm**, *Bioinformatics*, 2015.

4. S. Saha and S. Rajasekaran, and R. Ramprasad: **Novel Randomized Feature Selection Algorithms**, *International Journal of Foundations of Computer Science* (IJFCS), 2015.

5. S. Saha and S. Rajasekaran: **Efficient and scalable scaffolding using optical**

restriction maps, *BMC Genomics*, 2014.

6. S. Saha and S. Rajasekaran: **A Unified Bug Testing Software for Biological and Other Systems**, *Journal of Applied Bioinformatics & Computational Biology* (JABCB), 2014.

7. S. Saha, S. Rajasekaran, J. Bi, and S. Pathak: **Efficient techniques for genotype-phenotype correlational analysis**, *BMC Medical Informatics and Decision Making* (BMC MIDM), 2013.

8. S.R. Hussain, S. Saha, and A. Rahman: **SAAMAN: Scalable Address Autoconfiguration in Mobile Ad Hoc Networks**, *Journal of Network and Systems Management* (JNSM), 2011.

2.2 Conference Publications

1. S. Rajasekaran and S. Saha: **Efficient Algorithms for the Three Locus Problem in Genome-wide Association Study**, *IEEE International Conference on Data Mining* (ICDM), 2016.

2. S. Rajasekaran and S. Saha: **Efficient Algorithms for the Two Locus Problem in Genome-wide Association Study**, 25^{th} *ACM International Conference on Information and Knowledge Management* (CIKM), 2016.

3. S. Saha and S. Rajasekaran: **POMP: a powerful splice mapper for RNA-seq reads**, 7^{th} *ACM Conference on Bioinformatics, Computational Biology and Health Informatics* (ACM BCB), 2016.

4. S. Saha and S. Rajasekaran: **REFECT: a novel paradigm for correcting short reads**, 6^{th} *ACM Conference on Bioinformatics, Computational Biology and Health Informatics* (ACM BCB), 2015.

5. S. Saha and S. Rajasekaran: **NRRC: A Non-Referential Reads Compression Algorithm**, 11^{th} *International Symposium on Bioinformatics Research and Applications* (ISBRA), 2015.

6. S. Saha, S. Rajasekaran, and R. Ramprasad: **A Novel Randomized Feature Selection Algorithm**, 9^{th} *International Conference on Data Mining* (DMIN), 2013.

7. S. Rajasekaran and S. Saha: **A Novel Deterministic Sampling Technique to Speedup Clustering Algorithms**, 9^{th} *International Conference on Advanced Data Mining and Applications* (ADMA), 2013.

8. S. Saha, S.R. Hussain, and A. Rahman: **RBP: Reliable Broadcasting Protocol in Large Scale Mobile Ad Hoc Networks**, 24^{th} *IEEE International Conference on Advanced Information Networking and Applications* (AINA), 2010.

9. S.R. Hussain, S. Saha, and A. Rahman: **An Efficient and Scalable Address Autoconfiguration in Mobile Ad Hoc Networks**, 8^{th} *International Conference on Ad-Hoc, Mobile and Wireless Networks* (ADHOC-NOW), 2009.

Part II

Biological Sequence Analysis

Chapter 1

A Referential Genome Compression Algorithm

Next-generation sequencing (NGS) techniques produce millions to billions of short reads. The procedure is not only very cost effective but also can be done in laboratory environment. The state-of-the-art sequence assemblers then construct the whole genomic sequence from these reads. Current cutting edge computing technology makes it possible to build genomic sequences from the billions of reads within a minimal cost and time. As a consequence, we see an explosion of biological sequences in recent times. In turn, the cost of storing the sequences in physical memory or transmitting them over the internet is becoming a major bottleneck for research and future medical applications. Data compression techniques are one of the most important remedies in this context. We are in need of suitable data compression algorithms that can exploit the inherent structure of biological sequences. Although standard data compression algorithms are prevalent, they are not suitable to compress biological sequencing data effectively. In this research work we propose a novel referential genome compression algorithm (NRGC) to effectively and efficiently compress the genomic

sequences. We have done rigorous experiments to evaluate NRGC by taking a set of real human genomes. The simulation results show that our algorithm is indeed an effective genome compression algorithm that performs better than the best-known algorithms in most of the cases. Compression and decompression times are also very impressive.

1.1 Introduction

Next-generation sequencing (NGS) techniques reflect a major breakthrough in the domain of sequence analysis. Some of the sequencing technologies available today are Massively parallel signature sequencing (MPSS), 454 pyrosequencing, Illumina (Solexa) sequencing, SOLiD sequencing, Ion semiconductor sequencing, etc. Any NGS technique produces abundant overlapping reads from a DNA molecule ranging from tiny bacterium to human species. Modern sequence assemblers construct the whole genome by exploiting overlap information among the reads. Since the procedure is very cheap and can be done in standard laboratory environments, we see an explosion of biological sequences that have to be analysed. But before analysis the most important prerequisite is storing the data in a permanent memory. As a consequence, we need to increase physical memory to cope up with this increasing amount of data. By 2025, between 100 million and 2 billion human genomes are expected to have been sequenced, according to [20]. The storage requirement for this data alone could be as much as 2-40 exabytes (one exabyte being 10^{18} bytes). Although the recent engineering innovation has sharply decelerated the cost to produce physical memory, the abundance of data has already outpaced it. Besides this the most reliable mechanism to send data instantly around the globe is using the Internet. If the size of the data is huge, it will certainly create a burden over the Internet. Network congestion and higher transmission

costs are some of the side-effects. Data compression techniques could help alleviate these problems. A number of techniques can be found in the literature for compressing general data. They are not suitable for special purpose data like biological sequencing data. As a result, the standard compression tools often fail to effectively compress biological data. In this context we need specialized algorithms for compressing biological sequencing data. In this research work we offer a novel algorithm to compress genomic sequences effectively and efficiently. Our algorithm achieves compression ratios that are better than the currently best performing algorithms in this domain. By compression ratio we mean the ratio of the uncompressed data size to the compressed data size.

The following two versions of the genome compression problem have been identified in the literature: 1) *Referential Genome Compression*. The idea is to utilize the fact that genomic sequences from the same species exhibit a very high level of similarity. Recording variations with respect to a reference genome greatly reduces the disk space needed for storing any particular genomic sequence. The computation complexity is also improved quite a bit. So, the goal of this problem is to compress all the sequences from the same (or related) species using one of them as the reference. The reference is then compressed using either a general purpose compression algorithm or a reference-free genome compression algorithm. 2) *Reference-free Genome Compression*. This is the same as problem 1 stated above, except that there is no reference sequence. Each sequence has to be compressed independently. In this research work we focus on Problem 1. We propose an algorithm called NRGC (Novel Referential Genome Compressor) based on a user defined reference genome. It is based on a novel placement scheme. We divide the entire target genome into some non-overlapping segments. Each segment is then placed onto a reference genome to find the best placement. After computing the best possible placements each segment is then compressed using the corresponding segment of the reference. Simulation results show that NRGC is indeed an

effective compression tool.

1.2 A Survey of Compression Algorithms

We now briefly introduce some of the algorithms that have been proposed to compress genomic sequences using a reference from the same species. In referential genome compression the goal is to compress a large set S of similar sequences potentially coming from similar species. The basic idea of referential genome compression can be defined as follows. We first choose the reference sequence R. The selection of R can be purely random or it can be chosen algorithmically. All the other sequences $s \in S - R$ are compressed with respect to R. The target T (i.e., the current sequence to be compressed) is first aligned onto the reference R. Then mismatches between the target and the reference are identified and encoded. Each record of a mismatch may consist of the position with respect to the reference, the type (e.g., insertion, deletion, or substitution) of mismatch, value and the matching length.

[3] have used various coding techniques such as Golomb [9], Elias [16], and Huffman [10] to encode the mismatches. [21] have presented a compression program, GRS, which obtains variant information by using a modified Unix diff program. The algorithm GReEn [17] employs a probabilistic copy model that calculates target base probabilities based on the reference. Given the base probabilities as input, an arithmetic coder was then employed to encode the target. Recently an algorithm called ERGC (Efficient Referential Genome Compressor) [18] has been introduced which is based on a reference genome. It employs a divide and conquer strategy. Another algorithm, namely, iDoComp [14] has been proposed recently which outperforms some of the previously best-known algorithms like GRS, GReEn, and GDC. GDC [7] is an LZ77-style compression scheme for relative compression of multiple

genomes of the same species. In contrast to the algorithms mentioned above, [5] have proposed the DNAzip algorithm. It exploits the human population variation database, where a variant can be a single-nucleotide polymorphism (SNP) or an indel (an insertion and/or a deletion of multiple bases). Some other notable algorithms that employ VCF (Variant Call Format) files to compress genomes have been given by [6] and [15]. Next we provide a brief outline some of the best-known algorithms in the domain of referential genome compression. An elaborate summary can be found in [18].

GRS at first finds longest common subsequences between the reference and the target genomes. It then employs the Unix *diff* program to calculate a similarity score between the two sequences. Based on the similarity score it either encodes the variations between the reference and target genomic sequences using Huffman encoding or the reference and target sequences are divided into smaller blocks. In the later case, the computation is then restarted on each pair of blocks. The performance of GRS degrades sharply if the variation is high between the reference and target genomes. GDC can be categorized as a LZ77-style [22] compression algorithm. It is mostly a variant of RLZopt [19]. It finds the matching subsequences between the reference and the target by employing hashing where RLZopt employs suffix array. GDC is referred to as a multi-genome compression algorithm. From a set of genomes, it cleverly detects one (or more) suitable genome(s) as reference and compresses the rest based on the reference. An arithmetic encoding scheme is introduced in GReEn. At the beginning it computes statistics using the reference and an arithmetic encoder is then used to compress the target by employing the statistics. GReEn uses a copy expert model which is largely based on the non-referential compression algorithm XM [4].

iDoComp is based on suffix array construction and Entropy encoder. Through suffix array it parses the target into the reference and Entropy encoder is used to compress the variations. The most recent algorithm ERGC divides both the target and the reference sequences into

parts of equal size and finds one-to-one maps of similar regions from each part. It then outputs identical maps along with dissimilar regions of the target sequence. Delta encoding and PPMD lossless compression algorithm are used to compress the variations between the reference and the target genomes. If the variations between the reference and the target are small, it outperforms all the best-known algorithms. But its performance degrades when the variations are high.

Since referential genome compression is based on finding similar subsequences between the reference and the target genomes, some existing algorithms like MUMmer [11] or BLAST [2] can be used to find the maximal matching substrings. The acronym "MUMmer" comes from "Maximal Unique Matches", or MUMs. MUMmer is based on the suffix tree data structure designed to find maximal exact matches between two input sequences. After finding all the maximal matching substrings, an approximate string aligner can be used to detect the variations.

1.3 Methods

We can find all the variations between the reference and the target genomic sequences by employing any exact global alignment algorithm. Since the time complexity of such an algorithm is typically quadratic, it is not computationally feasible. So, every referential genome compression algorithm employs an approximate string matcher which is greedy in nature. Although genomic sequences of two individuals from the same species are very similar, there may be high variations in some regions of genomes. This is due to the large number of insertions and/or deletions in the genomic sequences of interest. In this scenario, greedy algorithms often fail to perform meaningful compressions. Either they can run indefinitely to

search for common substrings of meaningful length or output compressed data of very large size. Taking all of these facts into consideration, in this research work we propose a novel referential genome compression algorithm which is based on greedy placement schemes. Our algorithm overcomes the disadvantages of the existing algorithms effectively.

There are three phases in our algorithm. In the first phase we divide the target genome T into a set of non-overlapping segments $t_1, t_2, t_3 \ldots, t_n$ of length L each (for some suitable value of L). We then compute a score for each segments t_i corresponding to each possible placement of t_i onto reference genome R employing our novel scoring algorithm. The scores computed in the first phase are then used to find a non-overlapping placement of each t_i onto R in the second phase. This task is achieved using a placement algorithm that we introduce. Finally in the third phase we record the variation between each segment t_i and the reference genome R by employing our segment compression algorithm. More details of our algorithm are provided next.

Algorithm 1.1: Scoring Algorithm (SA)

Input: Ordered fragment lengths of reference R, Ordered fragment lengths of segments $t_1, t_2, t_3, \ldots, t_n$ from target T, Penalty P
Output: Scores S of segments $t_1, t_2, t_3, \ldots, t_n$ from target T
begin

```
1  for i := 1 to n do
2      q := number of ordered fragments in t_i;
3      for j := 1 to m − q + 1 do
4          Align t_1^i, t_2^i, t_3^i, ..., t_q^i onto
           r_j, r_{j+1}, r_{j+2}, r_{j+3}, ..., r_{j+q};
5          Find the number of missed fingerprint sites MFS;
6          Compute score of t_i using Equation 3.3.1;
7          Add the score in S[t_i];
8  Return S;
```

1.3.1 Computing scores

At the beginning, the target genome T is divided into a set of non-overlapping segments $t_1, t_2, t_3 \ldots, t_n$ each of a fixed length L where L is user defined. Since the genomic sequence can be composed of millions to billions of base-pairs and can contain large insertions and/or deletions along with mutations, finding the best possible placement of t_i onto R is not trivial. In fact an exact algorithm will have a quadratic time complexity to compute the best possible placements for all the t_is. Let $|R|$ and $|T|$ be the lengths of R and T, respectively. The time complexity of an exact algorithm could be $O(|R||T|)$ which is extremely high. There is a trade off between the time an algorithm takes and the accuracy it achieves. We accomplish a very good balance between these two by carefully formulating the scoring algorithm. This is done by employing fingerprinting and an ordered lengths of the fragments. We randomly generate a small substring F of length l where $4 \leq l \leq 6$ considering only A, C, G and T characters. F serves as a barcode/fingerprint in this context. Each possible occurrence of F is then collected from R. Since we know the position of each occurrence of F at this point, we can build an ordered lengths of the fragments by clipping the sequence at known fingerprint positions. Following the same procedure stated we can compute the ordered lengths of the fragments for each t_i by employing the same F. Suppose there are no errors (either indels or substitutions) in R and T. In this scenario for any given ordered fragment lengths of a segment t_i, in general, there should exist a subset of matching ordered fragment lengths in the reference R. This information helps to place a segment t_i onto R. But in reality errors could occur due to deletions of some fingerprint sites or a change in some fragment lengths (due to insertions). A novel scoring algorithm is thus introduced to quantify the errors.

Let $A = t_1^i, t_2^i, t_3^i, \ldots, t_q^i$ be the ordered fragment lengths of segment t_i from T and $B = r_s, r_{s+1}, r_{s+2}, r_{s+3}, \ldots, r_{m-s+1}$ be the ordered fragment lengths of a particular region of R.

The region is stretched from s^{th} fragment to $(m-s+1)^{th}$ fragment. The score of t_i for this particular region is computed as in Equation 3.3.1.

Algorithm 1.2: Placement Algorithm (PA)

Input: Segments $t_1, t_2, t_3, \ldots, t_n$ from target T with associated scores S
Output: Placements P of segments $t_1, t_2, t_3, \ldots, t_n$ from target T
begin
1 **for** $i := 1$ **to** n **do**
2 $q :=$ number of ordered fragments in t_i;
3 Sort $m - q + 1$ matching scores of t_i in increasing order;
4 Store the score of t_i and associated start and end index in L; $L[t_i] := \{scores, indices\}$;
5 Store the least score of t_i and associated start and end index in L'; $L'[t_i] := \{leastscore, indices\}$;
6 Sort L' with respect to start index in increasing order;
7 **for** $i := 1$ **to** n **do**
8 Extract information from $L'[t_i]$;
9 **if** (t_i not overlap with already placed segments in P)
10 Place the segment t_i at the end of the list P;
11 **else**
12 Goto line 9 and try to place t_i using top 5 least scores from $L[t_i]$;
13 **if** (t_i could not be placed)
14 Return *failure*;
15 Return P;

$$Score(t_i) = \left| \sum_{j=1}^{q_i} t_j^i - \sum_{j=s}^{m-s+1} r_j \right| + P * MFS \tag{1.3.1}$$

In Equation 3.3.1 P and MFS are the penalty factor and number of missed fingerprint sites, respectively. Penalty term P is user defined and should be very large. Details of our scoring algorithm follow. Let $r_1, r_2, r_3, \ldots, r_m$ be the ordered fragment lengths of the reference R. Let $t_1^i, t_2^i, t_3^i, \ldots, t_q^i$ be the ordered fragment lengths computed from any segment t_i. The

Algorithm 1.3: Variation Detector and Compressor (VDC)

Input: Reference sequence R, target sequence T, Placements associated with segments P from T

Output: Compressed sequence T_C

begin

1. Divide R into m parts ($=|P|$) where the segment r_i corresponds to the segment t_i from T using the placement information. Let these be $r_1, r_2, \ldots, r_m$ and $t_1, t_2, \ldots, t_m$, respectively;
2. **for** $i := 1$ **to** m **do**
3. Divide r_i and t_i into s equal parts. Let these be $r_1^i, r_2^i, \ldots, r_s^i$ and $t_1^i, t_2^i, \ldots, t_s^i$, respectively;
4. **for** $j := 1$ **to** s **do**
5. Hash the k-mers (for some suitable value of k) of r_j^i into a hash table H;
6. Generate one k-mer at a time from t_j^i and hash it into H;
7. If there is no collision try different values of k and repeat lines 5 and 6;
8. If all the different $k-$ mers have been tried with no collision, extend the length of r_j^i and go to line 5;
9. When a collision occurs in H, align r_j^i and t_j^i with this common k-mer as the anchor;
10. Extend the alignment beyond the common k-mer until there is a mismatch;
11. Record the matching length and the starting position of this match in the reference segment r_j^i;
12. Store the raw (unmatched) subsequence of t_j^i;
13. Compress the stored information using delta encoding;
14. Encode the stored information using PPMD encoder;
15. Return the compressed sequence T_C

individual scores are then computed by matching t_1^i with r_1, t_2^i with r_2, t_3^i with r_3, and so on. In other words, we compute a score for t_1^i by matching it with r_j for each possible value of i where $1 \leq j \leq m$. In brief, the inputs of the scoring algorithm are ordered fragment lengths of the reference genome R and ordered fragments lengths of each non-overlapping segment t_i where $1 \leq i \leq n$. Since, m and q are the number of ordered lengths of the reference genome

R and a segment t_i, respectively, there will be $(m-q+1)$-matching scores for each t_i. Each score is calculated by incrementing the position by one until all the $(m-q+1)$-steps are used. In this context position refers to the length of a particular fragment in R. So, the first position refers to the first fragment, the second position refers to the second fragment, etc. After aligning the ordered fragment lengths of a segment t_i to a particular position of the reference R, we greedily detect the number of fragment lengths of t_i that coincide reasonably well with the ordered fragment lengths of R and the number of missed fingerprint sites. We then calculate a matching score of that particular position by employing Equation 3.3.1. We calculate all the $(m-q+1)$ scores of each segment t_i following the same procedure stated above.

A detailed pseudocode is supplied in Algorithm 1.1. The run time of our greedy scoring algorithm is $O(mnq)$, where m is the number of fragments in the reference genome R, n is the number of segments of target genome T and q is the maximum number of fragments in any segment t_i.

1.3.2 Finding placements

Our placement algorithm utilizes the matching scores for each segment t_i to correctly place it onto the reference genome R. The algorithm takes a score list of a particular segment t_i and an ordered fragment lengths of R as input. If m is the number of ordered fragment lengths computed from R and n is the number of non-overlapping segments of target T, then the number of scores associated with each segment t_i will be $m-n+1$. The algorithm proceeds as follows: At first the matching scores associated with t_i are sorted in increasing order. Hence the first position of the sorted list of t_i contains the minimum score among all the scores. As the penalty factor is very large, this matching score is the best score for

placing this particular t_i anywhere in R.

The case stated above outlined an expected ideal case. But sometimes it is not possible to place t_i by considering the least score. If the placements cause to share some regions of R by more than one segment, the placement strategy is not valid at all. To avoid the collision we first try to place t_1; Next we attempt to place t_2 and so on. When we try to place any segment t_i, we check whether the starting and/or ending fragments of segment t_i overlap with any of the already placed segments. If there is such an overlap, we discard this placement and move onto the next segment in the sorted list to correctly place it onto R.

A detailed pseudocode is supplied in Algorithm 1.2. Let m be the number of fragments in the reference genome R, and n be the number of segments from target T. Intuitively the number of matching scores of each segment t_i is at most $O(m)$. Since the matching score is an integer, sorting matching scores of each segment t_i takes at most $O(m)$ time. So, the execution time of lines 1-5 in Algorithm 1.2 is $O(mn)$. Sorting segments with respect to starting position of fragments takes $O(n)$ time (line 6). In the worst case detecting the overlaps (lines 7-14) takes $O(n \log n)$ time. Since $n \ll m$, the run time of Algorithm 1.2 is $O(mn)$.

1.3.3 Recording variations

This is the final stage of our algorithm NRGC. Let $t_1, t_2, t_3, \ldots, t_q$ be the segments of T that are placed onto the segments $r_1, r_2, r_3, \ldots, r_q$ of R, respectively. The algorithm proceeds by taking one segment at a time. Consider the segment t_1. At first t_1 and r_1 are divided into s equal parts, i.e., $t_1 = t_1^1 t_2^1 t_3^1 \ldots t_s^1$ and $r_1 = r_1^1 r_2^1 r_3^1 \ldots r_s^1$, respectively. The variations of t_1^1 with respect to r_1^1 is computed first, variations of t_2^1 with respect to r_2^1 is computed next, and so on. Let (r', t') be processed at some point in time. At first the algorithm decomposes

r' into overlapping substrings of length k (for a suitable value of k). These k-mers are then hashed into a hash table H. It then generates k-mers from t' one at a time and hashes the k-mers into H. This procedure is repeated until one k-mer collides with an entry in H. If a collision occurs we align t' onto r' based on this particular colliding k-mer and extend the alignment until we find any mismatch between r' and t'. We record the matching length, the starting position of this stretch of matching in the reference genome R and the mismatch. If no collision occurs, we decompose r' into overlapping substrings of length k' where $k' < k$ and follow the same procedure stated above. At this point we delete the matching sequences from r and t and align the rest using the same technique as described above. Since there could be large insertions in the target genome T, we record the unmatched sequence of T as a raw sequence. The procedure is repeated until the length of r or t becomes zero or no further alignment is possible.

The information generated to compress the target sequence is stored in an ASCII formatted file. After having processed all the segments of R and the corresponding segments in T, we compress the starting positions and matching length using delta encoding. The resulting file is further compressed using PPMD lossless data compression algorithm. It is a variant of Prediction by partial matching (PPM) algorithm and an adaptive statistical data compression technique based on context modeling and prediction. It predicts the next symbol depending on n previous symbols. This method is also known as prediction by Markov Model of order n. The rationale behind the prediction from n previous symbols is that the presence of any symbol is highly dependent on the previous symbols in any natural language. The Huffman and arithmetic coders are sometimes called the entropy coders using an order-(0) model. On the contrary PPM uses a finite context Order-(k) model. Here k is the maximum context that is specified ahead of execution of the algorithm. The algorithm maintains all the previous occurrences of context at each level of k in a table or trie with

associated probability values for each context. For more details the reader is referred to [13]. Some recent implementations of PPMD are effective in compressing text files containing natural language text. The 7-Zip open-source compression utility provides several compression options including the PPMD algorithm. Details of the algorithm are shown as Algorithm 1.3.

Consider a pair of parts r and t (where r comes from the reference and t comes from the target). Let $|r| = |t| = \ell$. We can generate k-mers from r and hash them in $O(\ell k)$ time. The same amount of time is spent, in the worst case, to generate and hash the k-mers of t. The number of different k-values that we try is a small constant and hence the total time spent in all the hashing that we employ is $O(\ell k)$. If a collision occurs, then the alignment we perform is greedy and takes only $O(\ell)$ time. After the alignment recording the difference and subsequent encoding also takes linear (in ℓ) time. If no collision occurs for any of the k-values tried, t is stored as such and hence the time is linear in ℓ. Put together, the run time for processing r and t is $O(\ell k)$. Extending this analysis to the entire target sequence, we infer that the run time to compress any target sequence T of length n is $O(qk)$ where k is the largest value used in hashing.

1.3.4 Parameters configuration

There are several user defined parameters and these can be found in the code for the proposed algorithm NRGC. Almost all of the experiments were done using default parameters. The most important parameter of the algorithm is the segment size. In the first phase of NRGC, the target genome is decomposed into a set of non-overlapping segments of fixed size L. In our experimental evaluations, we have fixed L as 500K. Users can change this value using an interface provided. A rule of thumb is: if the variations between the reference and the target genomes are small, L can be small otherwise it should be large. The penalty term P was set

to 9999. In the third phase NRGC builds hash buckets by decomposing the sequences into overlapping k-mers. The set of k-values used in the experiment was $K = \{11, 12, 13\}$.

Fingerprint/barcode was set to a default string "ACTAC" throughout the experiments. User can change it to any fingerprint/barcode string using the application interface. It is also permitted that application itself can generate fingerprint of user defined fixed size. In this case NRGC randomly selects alphabets from A, C ,G and T with equal probability and builds a barcode string of user defined length. It then computes the number of times the fingerprint found in the reference genome. This process is repeated several times and the most occurring fingerprint is chosen for ordered fragment length generation.

1.4 Results

TABLE 1.4.1: Datasets used in the experiments.

Dataset	Species	# of Chromosomes	Retrieved From
hg19	Homo sapiens	24	ncbi.nlm.nih.gov
hg18	Homo sapiens	24	ncbi.nlm.nih.gov
KO224	Homo sapiens	24	koreangenome.org
KO131	Homo sapiens	24	koreangenome.org
YH	Homo sapiens	24	yh.genomics.org.cn

TABLE 1.4.2: Performance evaluation of four algorithms using various metrics. Best values are shown in bold face. A.Size and R.Size refer to Actual Size and Reduced Size in MB, respectively. C.Time and D.Time refer to the Compression Time and Decompression Time in minutes, respectively.

				GDC			iDoComp			ERGC			NRGC		
Dataset	Reference	Target	A.Size	R.Size	C.Time	D.Time	R.Size	C.Time	D.Time	R.Size	C.Time	D.Time	R.Size	C.Time	D.Time
D_1	hg19	hg18	2,996	24.42	68.76	**0.54**	**5.15**	20.65	2.55	131.34	**13.93**	2.22	14.85	14.25	2.09
		KO131	2,938	TLE	TLE	TLE	78.79	21.73	12.51	247.15	16.93	2.21	**46.04**	**16.36**	**2.10**
		KO224	2,938	TLE	TLE	TLE	77.74	37.78	28.96	268.38	18.08	2.15	**43.54**	**16.49**	**2.27**
		YH	2,987	TLE	TLE	TLE	79.68	41.83	31.87	190.59	16.58	**2.12**	**33.04**	**14.54**	2.06
D_2	hg18	hg19	3,011	24.42	68.76	**0.54**	**6.10**	31.68	2.41	299.45	18.56	2.22	12.37	**14.70**	1.84
		KO131	2,938	TLE	TLE	TLE	65.03	35.75	11.65	**13.46**	**8.39**	**1.51**	36.89	14.55	1.95
		KO224	2,938	TLE	TLE	TLE	68.58	26.80	11.63	**12.03**	**7.85**	**1.35**	37.780	14.94	2.02
		YH	2,987	TLE	TLE	TLE	64.16	21.41	11.04	**7.52**	**10.10**	2.16	27.64	14.32	**1.98**
D_3	KO224	hg19	3,011	TLE	TLE	TLE	195.19	22.36	11.61	443.15	21.45	2.19	**28.90**	**14.90**	**2.09**
		hg18	2,996	TLE	TLE	TLE	200.91	19.86	12.07	**18.79**	**10.17**	**1.42**	30.69	14.37	2.11
		KO131	2,938	11.57	80.49	**0.83**	6.57	27.91	1.68	**5.98**	**7.23**	1.41	7.95	13.72	1.99
		YH	2,987	31.08	68.05	**0.52**	29.02	37.05	3.56	**8.81**	14.01	2.07	21.96	**13.82**	1.94
D_4	YH	hg19	3,011	TLE	TLE	TLE	37.11	22.55	13.02	433.41	20.14	2.13	**27.11**	**15.22**	**1.87**
		hg18	2,996	TLE	TLE	TLE	34.18	48.18	12.26	**17.22**	**7.25**	**1.37**	27.61	14.23	2.05
		KO131	2,938	36.28	73.66	**0.53**	19.16	25.01	4.41	**13.05**	**8.14**	1.37	27.99	14.47	1.94
		KO224	2,938	31.08	68.05	**0.52**	16.02	37.21	3.90	**11.57**	**7.89**	1.29	28.66	14.42	1.95

TABLE 1.4.3: Performance evaluation of three algorithms using various metrics. Best values are shown in bold face. A.Size and R.Size refer to Actual Size in MB and Reduced Size in MB, respectively. C.Time and D.Time refer to the Compression Time and Decompression Time in minutes, respectively.

		iDoComp			ERGC			NRGC			Gain	
Dataset	A.Size	R.Size	C.Time	D.Time	R.Size	C.Time	D.Time	R.Size	C.Time	D.Time	iDoComp	ERGC
D_1	11,859	241.36	121.99	75.89	816.23	65.52	8.70	**137.47**	**61.64**	**8.52**	43.04%	83.16%
D_2	11,874	203.87	115.64	36.73	332.46	**44.90**	**7.24**	**114.68**	58.51	7.79	45.22%	65.51%
D_3	11,932	431.69	107.18	28.92	476.73	**52.86**	**7.09**	**89.50**	56.81	8.13	79.27%	81.23%
D_4	11,883	**106.47**	132.95	33.59	475.25	**43.42**	**6.16**	111.37	58.34	7.81	-4.60%	76.57%

1.4.1 Experimental environment

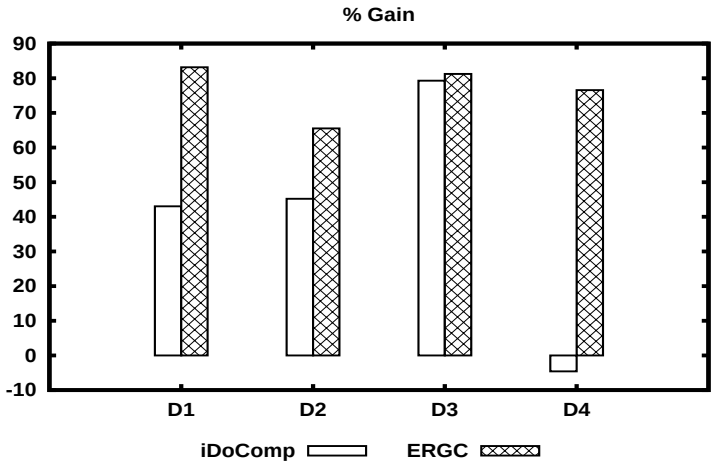
We have compared our algorithm with the best known algorithms existing in the referential genome compression domain. In this section we summarize the results. All the experiments were done on an Intel Westmere compute node with 12 Intel Xeon X5650 Westmere cores and 48 GB of RAM. The operating system running was Red Hat Enterprise Linux Server release 5.7 (Tikanga). NRGC compression and decompression algorithms are written in standard Java programming language. Java source code is compiled and run by Java Virtual Machine (JVM) 1.6.0.

1.4.2 Datasets

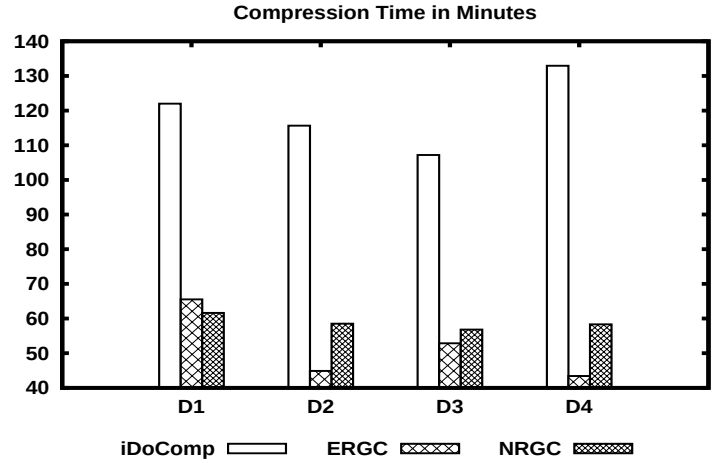
To measure the effectiveness of our proposed algorithm, we have done a number of experiment using real datasets. We have used hg19, hg18 release from the UCSC Genome Browser, the Korean genomes $KOREF\_20090131$ (KOR131 for short) and $KOREF\_20090224$ (KOR224 for short) [1], and the genome of a Han Chinese known as YH [12]. To show the effectiveness of our proposed algorithm NRGC each dataset acts as a reference. When a particular dataset is chosen to be the reference the rest act as targets. By following this procedure any bias related in using a particular reference is omitted. We have taken chromosome 1-22, X, and Y chromosomes (i.e., a total of 24 chromosomes) for comparison purposes. Please see Table 3.3.2 for details about the datasets we have used.

1.4.3 Outcomes

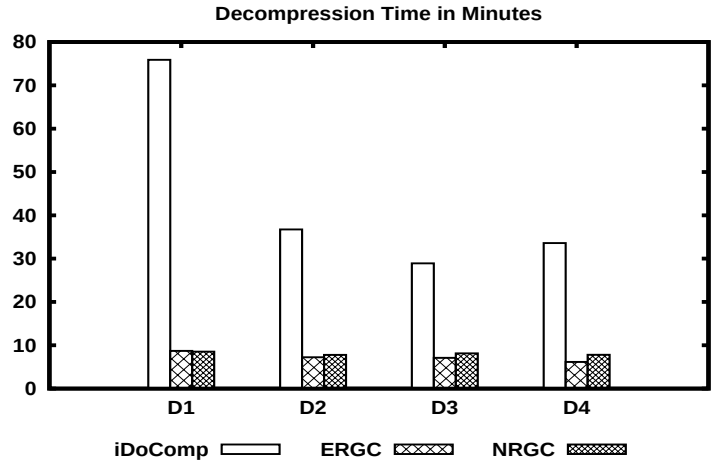
Next we discuss details on the performance evaluation of our proposed algorithm NRGC in terms of both compression and CPU elapsed time. We have compared NRGC with three



(A) % Gain over iDoComp and ERGC



(B) Compression times of different methods



(C) Deompression times of different methods

FIGURE 1.4.1: Performance comparisons of iDoComp, ERGC, and NRGC methods

TABLE 1.4.4: Phase-wise time decomposition of NRGC. CPU-elapsed times are given in minutes.

Dataset	Reference	Target	1^{st} Phase	2^{nd} Phase	3^{rd} Phase	Total
D_1	hg19	hg18	2.39	5.23	6.62	14.25
		KO131	2.63	5.30	8.42	16.36
		KO224	2.81	5.63	8.04	16.49
		YH	2.10	5.38	7.05	14.54
D_2	hg18	hg19	2.25	5.02	7.42	14.70
		KO131	2.40	5.29	6.84	14.55
		KO224	2.46	5.44	7.03	14.94
		YH	2.02	5.41	6.88	14.32
D_3	KO224	hg19	2.12	5.83	6.94	14.90
		hg18	2.43	5.41	6.52	14.37
		KO131	2.46	5.40	5.85	13.72
		YH	1.97	5.33	6.52	13.83
D_4	YH	hg19	2.20	5.99	7.03	15.22
		hg18	2.39	5.41	6.42	14.23
		KO131	2.40	5.45	6.61	14.47
		KO224	2.38	5.39	6.64	14.42

of the four best performing algorithms namely GDC, iDoComp and ERGC using several standard benchmark datasets. GReEn is one of the state-of-the-art algorithms existing in the literature. But we could not compare GReEn with our algorithm. The site containing the code of GReEn was down at the time of experiments. Although run time and compression raio of ERGC was impressive, it did not perform meaningful compression when the variation between target and reference is large. It performs well when the variation between target and reference is small which is not always the case in our experiments. In fact NRGC is a superior version of ERGC. GDC, GReEn, iDoComp and ERGC are highly specialized algorithms designed to compress genomic sequences with the help of a reference genome. These are the best performing algorithms in this area as of now.

Effectiveness of various algorithms including NRGC is measured using several performance metrics such as compression size, compression time, decompression time, etc. Gain measures the percentage improvement in compression achieved by NRGC when compared to

iDoComp and ERGC. Comparison results are shown in Table 3.2.2 and Table 3.3.1. Clearly, our proposed algorithm is competitive and performs better than all the best known algorithms. Memory consumption is also very low in our algorithm as it processes one and only one part from the target and reference sequences at any time. Please, note that we do not report the performance evaluation of GDC for every dataset, as it ran for at least 3 hours but did not complete the compression task for some datasets. We refer to it in this research work as Time Limit Exceeded (or TLE in short).

At first consider the dataset D_1. In this case we consider the hg19 human genome as the reference. Targets include hg18, KO131, KO224 and the YH human genome. iDoComp performs better in compressing the hg18 genome by employing hg19 as the reference. In all the other cases NRGC performs better in compressing KO131, KO224 and YH than all the other algorithms of interest. In fact NRGC compresses approximately 2 times better than iDoComp for those particular genomes. NRGC is also faster than iDoComp in terms of both compression and decompression times. Please see Table 3.2.2 for more details. Now consider the overall evaluation for dataset D_1 given in Table 3.3.1. The total size of the target genomes is 11,859 MB. NRGC algorithm compresses it to 137.47 MB corresponding to a compression ratio of 86.26. On the other hand, iDoComp achieves a compression ratio of 49.13. Specifically, the percentage improvement NRGC achieves with respect to iDoComp is 43.04%. Compression and decompression times of NRGC are almost 2× and 9× less than those of iDoComp. Note that we did not include the performance evaluation of GDC as in most of the cases it fails to compress the data within 3 hours. The average performance of ERGC is poor. The percentage improvement NRGC achieves over iDoComp is 83.16%.

Next consider the dataset D_2. In this case we consider the hg18 human genome as the reference and the rest as targets. iDoComp performs better in compressing the hg19 genome; in all the other cases NRGC performs better in terms of compression and elapsed

times. In fact NRGC compresses approximately $1.5 - 2.0\times$ better than iDoComp for those particular genomes (e.g., KO131, KO224 and YH). NRGC is also faster than iDoComp in terms of both compression and decompression times. Please see Table 3.2.2 for more details. Now consider the overall performance for the dataset D_2 given in Table 3.3.1. The percentage improvements NRGC achieves with respect to ERGC and iDoComp are 65.51% and 45.22%, respectively. Compression and decompression times of NRGC are also very impressive compared to iDoComp and comparable with ERGC. For the D_3 dataset the percentage improvements NRGC achieves over ERGC and iDoComp are 81.23% and 79.27%, respectively. The compression achieved by NRGC on the D_4 dataset is slightly lower than that of iDoComp. Please, see Figure 3.2.1 for visual details of different evaluation metrics.

1.5 Discussion

Our proposed algorithm is able to work with any alphabet used in the genomic sequences of interest. Other notable algorithms existing in the domain of referential genome compression can perform compression only with a restricted set of alphabets used for genomic sequences e.g., $\sum = \{A, a, C, c, G, g, T, t, N, n\}$. These characters most commonly seen in biological sequences. But there are several other valid characters frequently used in clones to indicate ambiguity about the identity of certain bases in sequences. In this context our algorithm is not restricted with the limited set of characters found in $\sum$. NRGC also differentiates between lower-case and upper-case letters. GDC, GReEn and iDoComp can identify the difference between upper-case and lower-case characters defined in $\sum$ but algorithms like GRS or RLZ-opt can only handle upper-case alphabet $\sum$. iDoComp replaces all the character in the genomic sequence with N that does not belong to $\sum$. Specifically, NRGC compresses

the target genome file regardless of the alphabets used and decompresses the compressed to produce a file that is exactly identical to the target file. GDC, iDoComp and ERGC perform the similar job. But GReEn does not include the metadata information and outputs the sequence as a single line instead of multiple lines, i.e., it does not encode the line-break information.

The difference between two genomic sequences can be computed by globally aligning them as the sequences in the query set coming from the same species are similar and of roughly equal size. Let R and T denote the reference and target sequences, respectively as stated above. The time complexity of a global alignment algorithm is typically $O(|R||T|)$, i.e., quadratic in terms of the reference and target lengths. Global alignment is solved by employing dynamic programming and thus is a very time and space intensive procedure specifically if the sequences are very large. In fact it is not possible to compute the difference between two human genomes using global alignment in current technology. Instead if we divide the reference and target into smaller segments and globally align the corresponding segments, the time and space complexities seem to be improved. But there are two shortcoming in this approach - (1) it still is quadratic with respect to segment lengths and (2) because of large insertions and/or deletions in the reference and/or target the corresponding segments may come from different regions (i.e., dissimilar). To quantify this issue we propose a placement scheme which efficiently finds the most suitable place for a segment in the reference. The segment is then compressed by our greedy variation detection algorithm.

From the experimental evaluations (please see Table 3.2.2) it is evident that ERGC performs better than GDC, iDoComp, and NRGC in 9 out of 16 datasets. It is also not restricted to the alphabets defined in $\sum$. But the main limitation of ERGC is that it performs better only when the variations between the reference and the target genomes are small. If the variations, i.e., insertions and/or deletions are high between the reference

and the target, its performance degrades dramatically. Since hg19 contains large insertions and/or deletions, ERGC fails to perform a meaningful compression while using this genome as the reference or the target. On the contrary NRGC performs better than ERGC (and other notable algorithms) on an average (please see Table 3.3.1). This is due to the fact that NRGC can handle large variations between the reference and target genomes. The main difference between NRGC and ERGC is that NRGC at first finds a near optimal placement of non-overlapping segments of target onto the reference genome and then records the variations. On the other hand, ERGC tries to align the segments contiguously and due to its look-ahead greedy nature it fails to align the segments when there are large insertions and/or deletions in the reference and/or the target genomes. In this scenario ERGC concludes that the segments could not be aligned and stores them as raw sequences.

As discussed previously, our proposed algorithm NRGC runs in three phases. At first it computes a score for each of the non-overlapping segments. These segments are then aligned onto the reference genome in the second phase using the scores computed in the first phase. After finding the best possible alignment, NRGC records the variations in the final phase. We provide the time elapsed in each phase in Table 3.2.1. Computing scores takes less time compared to alignment and record variation phases. This is due to the fact that the placement procedure performs sorting twice and searches for a non-overlapping placement for each segment. The execution time can be reduced by restricting the search to within certain regions of the reference genome. The third phase performs k-mer production, hash table generation, and recording variations. This is why it also consumes higher CPU cycles than the first phase.

1.6 Conclusions

In this research work we have proposed a novel referential genome compression algorithm. We employ a scoring based placement technique to quantify large variations among the genomic sequences. NRGC runs in three stages. At the beginning the target genome is divided into some segments. Each segment is then placed onto the reference genome. After getting the most suitable placement we further divide each segment into some non-overlapping parts. We also divide the corresponding segments of the reference genome into the same number of parts. Each part from the target is then compressed with respect to the corresponding part of the reference. A wide variety of human genomes are used to evaluate the performance of NRGC. It is evident from the simulation results that the proposed algorithm is indeed an effective compressor compared to the state-of-the-art algorithms existing in the current literature.

Chapter 2

A Non-referential Reads Compression Algorithm

In the era of modern sequencing technology, we are collecting a vast amount of biological sequence data. The technology to store, process, and analyze the data is not as cheap as to generate the sequencing data. As a result, the need for devising efficient data compression and data reduction techniques is growing by the day. Although there exist a number of sophisticated general purpose compression algorithms, they are not efficient to compress biological data. As a result, we need specialized compression algorithms targeting biological data. Five different NGS data compression problems have been identified and studied. In this research work we propose a novel algorithm for one of these problems. We have done extensive experiments using real sequencing reads of various lengths. The simulation results reveal that our proposed algorithm is indeed competitive and performs better than the best known algorithms existing in the current literature.

2.1 Introduction

High-throughput or NGS technologies parallelize the sequencing process and produce millions to billions of short reads (of length 25 - 100 bp) simultaneously in a single run. Some of the sequencing technologies dominating the NGS market today are Massively parallel signature sequencing (MPSS), 454 pyrosequencing, Illumina (Solexa) sequencing, SOLiD sequencing, Ion semiconductor sequencing, etc. The reads are short and very large in number. To process and analyze the sequencing data, at first we need to efficiently store this vast amount of data. Specifically the increase in sequencing data generation rate is outpacing the rate of increase in disk storage capacity. Furthermore, when the size of the data transmitted through the internet increases, the transmission cost and congestion in the network also increase. Thus it is vital to devise efficient algorithms to compress biological data. General purpose algorithms do not consider some inherent properties of sequencing data, e.g., repetitive regions, identical reads, etc. Exploiting these properties one can devise better algorithms compared to general purpose data compression algorithms. In this research work we offer a novel algorithm to compress biological sequencing reads effectively and efficiently. Our algorithm achieves better compression ratios than the currently best performing algorithms in the domain of reads compression. By compression ratio we mean the ratio of the uncompressed data size to the compressed data size.

The following five versions of compression have been identified in the literature: 1) *Genome compression with a reference.* Here we are given many (hopefully very similar) genomic sequences. The goal is to compress all the sequences using one of them as the reference. The idea is to utilize the fact that the sequences are very similar. For every sequence other than the reference, we only have to store the difference between the reference and the sequence itself; 2) *Reference-free Genome Compression.* This is the same as

problem 1, except that there is no reference sequence. Each sequence has to be compressed independently; 3) *Reference-free Reads Compression.* Reference-free reads compression algorithms are needed in biological applications where there is no clear choice for a reference; 4) *Reference-based Reads Compression.* In this technique complete read data need not be stored but only the variations with respect to a reference genome are stored; and 5) *Metadata and Quality Scores Compression.* In this problem we are required to compress quality sequences associated with reads as well as metadata such as read name, platform, and project identifiers.

In this research work we focus on problem 3. We present an effective reference-free reads compression algorithm namely NRRC: *Non-Referential Reads Compression Algorithm.* This algorithm takes any FASTQ file as input and outputs the compressed reads in FASTA format. To begin with, reads are clustered based on a hashing scheme. Followed by this clustering, a representative string is chosen from each cluster of reads. Compression is independently done for each cluster. In particular, the representative string in any cluster is used as a reference to compress the other reads in this cluster. Simulation results show that our proposed algorithm performs better than the best known algorithms existing in the current literature.

2.2 Related Works

We now briefly survey some of the algorithms that have been proposed in the literature to solve the problems of biological data compression.

In referential genome compression the goal is to compress a set S containing a large number of similar sequences. The core idea of reference-based compression can be described as follows. We first choose a reference sequence R from S. Then we compress every other

sequence $s \in S$ by comparing it with R. Brandon et al. [3] have used various coders like Golomb, Elias, and Huffman to encode the mismatches. Christley *et al.* [5] have proposed the DNAzip algorithm that exploits the human population variation database, where a variant can be a single-nucleotide polymorphism (SNP) or an indel (an insertion or a deletion of multiple bases). In contrast to DNAzip, Wang *et al.* [21] have presented a *de novo* compression program, GRS, which obtains variation information by using a modified UNIX diff program. The algorithm GReEn [40] employs a probabilistic copy model that calculates target base probabilities based on the reference. Given the base probabilities as input, an arithmetic coder was then used to encode the target.

Reference-free genome compression algorithms compress a single sequence at a time by identifying repeats in the given sequence and replacing these repeats with short codes. For example, BioCompress [30] and BioCompress-2 [31] methods are based on Lempel-Ziv (LZ) style substitutional algorithm to compress exact repeats and palindromes. BioCompress-2 also utilizes an order-2 context-based arithmetic encoding scheme to store the non-repetitive regions. Alternatively, GenCompress [26] and DNACompress [27] identify approximate repeats and palindromes so that a large fraction of the target sequence can be compressed to get a high compression ratio. Similarly GeNML [35] divides a sequence into blocks of fixed size. Then, blocks rich in approximate repeats are encoded with an efficient normalized maximum likelihood (NML) model; otherwise, plain or order-1 context based arithmetic encoding is used. Some other well-known reference-free genome compression algorithms are BIND [24], DELIMINATE [39], COMRAD [36], DNAEnc3 [40], and XM [4].

Reads compression methods can also be categorized into reference-based and non reference-based, similar to genome compression techniques. Next we survey some of the non-referential reads compression algorithms. G-SQZ method has been proposed by Tembe *et al.* [44] where the frequency of each unique tuple <base, quality> is computed. Each tuple is then encoded

by generating a Huffman code where the more frequent a tuple is the less number of bits is needed to encode it. After this, the encoded tuples along with a header containing meta-information like the platform and the number of reads are written to a binary file. The DSRC algorithm [29] divides the input into blocks of 32 records and every 512 blocks are clustered to make a superblock. Each of the superblocks is then indexed and compressed independently using LZ77 coding scheme. Quip [33] uses an arithmetic encoding scheme based on high order Markov chains. Fqzcomp and Fastqz are proposed by Bonfield and Mahoney [23]. In these methods, the sequences are compressed by exploiting an order-N context model and an arithmetic coder. BEETL is an algorithm of Coax *et al.* [28]. In this algorithm, repeats among the reads are identified using the Burros-Wheeler Transform (BWT) data structure [25]. The transformed data is then compressed exploiting general purpose compression algorithms like gzip, bzip2, or 7-zip. A similar algorithm is SCALCE [32] where the consistent parsing algorithm [43] is used to find an identical longest 'core' substring from the clustered reads. The reads within a cluster are then compressed using other standard compression algorithms.

In a reference-based technique complete read data need not be stored but only the variations with respect to a reference genome are stored. Reference-based algorithms typically run in two steps. In the first step all the reads are aligned to a reference genome of interest by using one of the NGS aligners such as Bowtie [37], BWA [38], Novoalign (http://www.novocraft.com), etc. In the second step, the mapped positions and variations are encoded using methods such as arithmetic coding and Huffman coding. Some of the algorithms in this domain are GenCompress, SlimGene, CRAM [8], Quip, NGC [42], Samcomp, etc. In addition to the mapping and encoding procedure, CRAM uses a *de Bruijn graph* based assembly approach where assembled contigs are used to map all the unaligned reads. Quip assembles the genomic sequence from the given reads using a *de Bruijn* graph based *de novo* assembler.

2.3 Methods

In this section we present our novel algorithm for problem 3. Specifically, we present an algorithm for the following problem: Reference-free Reads Compression. **For this version of compression, our algorithm achieves better compression ratios than the currently best known algorithms for this problem.** We provide details of our Non-Referential Reads Compression (NRRC) algorithm next. There are 4 basic steps in our novel algorithm. At first NRRC clusters the given set of reads based on overlaps and similarity. For each cluster a consensus sequence is created. All the reads in the cluster are compressed using the consensus as the reference. I.e., for each read we only store its difference with the consensus. Two reads are said to be *neighbors* of each other if they have a large overlap (with a small Hamming distance in the overlapping region). We find the neighbors of each read in steps 1 and 2 and use this neighborhood information to cluster the reads and perform compression (in step 3 and 4). In step 1 we find the *potential neighbors* of each read and in step 2 we find the true neighbors of each read. More details follow.

2.3.1 Finding potential neighbors

Potential neighbors for any read are found using two hashings. In each hashing we generate all the k-mers (for some suitable value of k) of all the reads and hash them based on these k-mers. A read is a potential neighbor of another read if they are hashed into the same value in at least one of the two hashings. For every read we collect all the potential neighbors from the two hashings and merge them. Followed by this, in step 2 we find the neighbors of each read as explained in the next section. Having two different hashings enables us to maximize the chances of finding as many of the neighbors as possible for each of the reads.

In the first hashing, we generate the k_1-mers in each read and hash the reads based on

these k_1-mers (for some suitable value of k_1). Let $h_1(.)$ be the hash function employed. By a hash bucket (or simply a bucket) we mean all the reads that have the same hash value with respect to at least one of the k_1-mers in them. If read R_1 has a k_1-mer x, read R_2 has a k_1-mer y, and if $h_1(x) = h_1(y)$ then we say that the reads R_1 and R_2 fall into the same bucket. Any read R will be hashed into at most $r - k_1 + 1$ buckets, where $r = |R|$. For every read R we collect *potential neighbors* from the buckets that R falls into. All the reads that fall into at least one of the buckets that R falls into will be called *potential neighbors* of R. We perform one more such hashing by generating k_2-mers of reads. Here k_1 and k_2 are appropriate integers chosen to optimize performance. In the second hashing also we collect potential neighbors for each read. The potential neighbor lists collected for each read from the two hashings are merged together.

Whenever we hash k-mers (where k is either k_1 or k_2) we record necessary information about each of the k-mers such as the read associated with it and its starting position in the read. We need this information to find and align overlapping reads. To reduce memory usage we only record a unique read id associated with any read and an integer corresponding to the starting position of the k-mer. As no hash function is perfect, two similar k-mers may be hashed into two different buckets. Also, two dissimilar k-mers might be hashed into the same bucket. In this case we will loose some potential neighbors which could play important roles in compression. Having two different hashings enables us to maximize the chances of finding as many of the neighbors as possible for each of the reads. After finding the potential neighbors of each of the reads by traversing the hash buckets, we merge the neighbors information.

2.3.2 Finding neighbors

After having collected potential neighbors for each read, we do some pruning to eliminate those potential neighbors that are not likely to be neighbors. Let R be any read and let R' be another read that has a sufficient overlap with R. For instance, a suffix of R could overlap with a prefix of R'. In the pruning step we compute the Hamming distance between the two reads in the overlapping region. If this distance is less than a threshold, we will keep R' as a neighbor R. If not, we will prune R' from the neighbor list of R. Note that the same two reads might fall into more than one buckets together. In this case we will identify and use the largest overlap between the pair. This is how merging of neighbors' information between two hash buckets is also done.

2.3.3 Aligning and building a consensus string

If R is any read and $L(R)$ is the list of neighbors of R, we correct R using $L(R)$. If R' is any read in $L(R)$ we already have found the maximum overlap between R and R' in Step 2. In Step 2 we have also ensured that the Hamming distance between R and R' in the overlapping region is within a small threshold. We align every R' (from $L(R)$) with R in a greedy manner using the overlapping region. The **greedy alignment** is done as follows. Let R' be a potential neighbor of R as stated above. As R and R' are potential neighbors, they must share at least one identical $k-$mer. We align R' with R using this k-mer as the anchor. We then extend this match on both sides as much as possible. Specifically, let $R = x_1x_2\cdots x_r$ and $R' = y_1y_2\cdots y_r$. Let the common k-mer between R and R' be $x_ix_{i+1}\cdots x_{i+k-1}$ and $y_jy_{j+1}\cdots y_{j+k-1}$. We identify the least $i_1 \leq i$ and the largest $i_2 \geq (i+k-1)$ such that the Hamming distance between $x_{i_1}x_{i_1+1}\cdots x_i\cdots x_{i+k-1}\cdots x_{i_2}$ and $y_{j-(i-i_1)}y_{j-(i-i_1)+1}\cdots y_{j+i_2-i}$ is $\leq d$. If $(i_2-i_1+1) \geq \frac{r}{2}$, we call R' as a neighbor of R. Since the error rate in NGS technology

is very small, we expect that if R and R' come from the same region of the genome, then they will share more than one $k-$mers. While processing the buckets of the hash table, we keep track of the $k-$mer for which the size of the overlapping region between R and R' (i.e., $i_2 - i_1 + 1$) is the biggest and at the same time the Hamming distance between them is within d. We align R and R' based on this k-mer. Note that the **greedy alignment** that we do does not take much time. After aligning the reads in $L(R)$ with R, we construct a consensus sequence by taking the calculated order of most frequent residues (here nucleotides), found at each position in the alignments.

Values of parameters such as k_1, k_2, etc. have been optimized to get the best results. To speed up the proposed algorithm we have used several techniques. After neighborhood calculation, if a read has less than two neighbors, we discard the read as the read is potentially too erroneous to be corrected. Also, if the size of any hash bucket is very large, we omit the entire bucket from further consideration as it potentially corresponds to repeated regions of the genomic sequence. Furthermore, if the size of a bucket is greater than a certain threshold we randomly pick some of the reads and discard the others from that bucket.

2.3.4 Compressing and encoding the reads

Each read in $L(R)$ and R are compressed using the consensus string. Specifically, for each read we only store its difference with the consensus string. Note that a read may appear in more than one cluster. In this case, the read is compressed only in that cluster where its compressed length will be the least. After compressing the reads using the consensus, they are encoded using Lempel-Ziv-Markov chain algorithm (LZMA) - a lossless data compression algorithm. It is basically a dictionary based compression algorithm having large dictionary sizes. The output of the dictionary is encoded using a range encoder (a variant of entropy

encoding method). To predict the probability of each bit, it uses a complex probability model roughly similar to the arithmetic encoding technique. In brief LZMA performs the compression in two basic steps. At first it detects matches using sophisticated and highly efficient dictionary-based data structure and generates a stream of literal symbols and phrase references. These are then encoded one bit at a time using a range encoder in the second step. It searches the space of many encoding and chooses the best one using a dynamic programming algorithm. Steps of the algorithm are shown in Algorithm 1.

Algorithm 1: Non-Referential Reads Compressor (NRRC)

Input: A set S of reads

Output A set S' of compressed reads

begin

1. Generate k-mers of each read and hash the reads based on these k-mers. Equal k-mers fall into the same bucket. If R is any read, any other read that falls into at least one of the buckets that R falls into is treated as a potential neighbor of R. For every read $R \in S$ create a list $P(R)$ of potential neighbors.

   Perform the above task twice with two different values for k and for every read merge the lists of potential neighbors from the two hash tables. If the size of a bucket is larger than a threshold, only a subset of the bucket is included in the potential neighbors identification process.

2. Let R be any read. Align every read in $P(R)$ with R. Let R' be any read in $P(R)$. If R and R' overlap sufficiently and if in the overlapping region the Hamming distance between R and R' is small, then we treat R' as a

neighbor of R. For every read $R \in S$ construct a list $L(R)$ of neighbors of R in this fashion.

3. Let R be any read. The neighbors of R reside in $L(R)$. Greedily align R' with R for every $R' \in L(R)$.

4. Make the consensus string R_C and compress R along with reads in $L(R)$ using R_C as a reference. Perform this step for every read $R \in S$. In this step, a read may be present in more than one cluster. In this case, the read is compressed in that cluster where its compressed length will be the least.

5. Encode the already compressed reads using Lempel-ZivMarkov chain algorithm (LZMA).

end

2.3.5 Time complexity analysis

In this section we analyze the time complexity of NRRC. Let n be the number of reads and r be the read length. In the first step of NRRC, we build hash tables and identify potential neighbors. The number of k-mers (k could either be k_1 or k_2) generated from each read is $r - k + 1$. Let $h(.)$ be the hash function employed. We think of the hash table as an array of buckets (or lists). Each bucket has an integer as its index. If the size of the array is N, then the index of any bucket is an integer in the range $[1, N]$. If k is small enough, one could employ direct hashing such that each k-mer is hashed into a bucket whose index is the k-mer itself (thought of as an integer). In this case, the hash array should be of size 4^k. If k is large, direct hashing may not be feasible. The expected size of each bucket is $\frac{(r-k+1)n}{N} = O\left(\frac{rn}{N}\right)$. The total time spent in hashing of Step 1 is $O(rn)$.

In the first step we also find potential neighbors of each read. A read falls into at most $r - k + 1 < r$ buckets and hence the expected number of potential neighbors for each read is $O\left(\frac{r^2 n}{N}\right)$. For every bucket we spend an expected $O\left(\left(\frac{rn}{N}\right)^2\right)$ time. Thus the total time spent in Step 1 has an expected value of $O\left(rn + \frac{r^2 n^2}{N}\right)$.

In Steps 2 and 3 we align reads. Specifically, if R is any read and $P(R)$ is the list of potential neighbors of R, then the expected size of $P(R)$ is $O\left(\frac{r^2 n}{N}\right)$. For every read $R' \in P(R)$, we align R' with R and compute the Hamming distance between R and R' in the overlapping region. Thus for every $R' \in P(R)$ we spend $O(r)$ time. As a result, the total time spent in Step 2 and 3 for each read is expected to be $O\left(\frac{r^3 n}{N}\right)$. Summing this over all the reads, the total expected time spent in Step 2 and 3 is $O\left(\frac{r^3 n^2}{N}\right)$.

In step 4, we form a consensus corresponding to the neighbors of each read. Let R be any read. Since the expected number of reads in $P(R)$ is $O\left(\frac{r^2 n}{N}\right)$, the expected time to build a consensus is $O\left(\frac{r^3 n}{N}\right)$. Subsequently, each read is compressed using the consensus. The expected time for this is also $O\left(\frac{r^3 n}{N}\right)$. Summing this over all the reads, the expected time spent in step 4 is $O\left(\frac{r^3 n^2}{N}\right)$.

In summary, the expected run time of NRRC (excluding the time for LZMA) is $O\left(rn + \frac{r^3 n^2}{N}\right)$.

TABLE 2.3.1: Illumina generated reads from human, mouse, and different organisms.

Dataset	Accession Number	# of Reads	Read Length	Description
D1	SRR037452	11,671,179	35	Human brain tissue (MAQC study)
D2	SRR635193.1	26,065,855	54	Pooled amnion from 5 term birth placentas
D3	SRR689233.1	16,407,945	90	Mouse oocyte
D4	SRR519063.1	26,905,342	51	*Pseudomonas aeruginosa* PAO1
D5	SRR001665.1	10,408,224	36	*Escherichia coli* str. K-12 substr. MG1655
D6	SRR361468	7,093,045	35	*Treponema pallidum* subsp. pertenue str. Gauthier
D7	SRR353563	7,061,388	100	*Leptospira interrogans* serovar Lai str. Lai
D8	SRR022866.1	12,775,858	76	*Staphylococcus aureus*
D9	SRR065202.1	11,954,555	42	*Haemophilus influenzae* Rd KW20

TABLE 2.3.2: Compression sizes of different reads compression algorithms in Bytes. Best results are shown in bold. Although PathEnc uses a reference for compression, compressed size of the reference was not added with the result it reported.

Dataset	2-Bit	SCALCE	fastqz	PathEnc	NRRC
D1	102,122,816	66,558,377	85,493,834	**45,180,142**	67,522,391
D2	351,889,042	95,474,107	179,766,179	**47,592,829**	74,048,376
D3	369,178,762	87,419,349	–	59,497,698	**50,795,839**
D4	342,043,110	23,210,258	90,128,271	15,797,769	**10,634,801**
D5	93,674,016	33,882,183	59,210,245	20,918,160	**17,339,502**
D6	62,064,143	19,403,853	35,952,162	13,064,384	**10,648,375**
D7	176,534,700	36,065,779	48,856,429	22,728,610	**15,201,009**
D8	242,741,302	110,790,886	128,966,807	91,259,052	**90,442,321**
D9	125,522,827	25,523,344	56,519,690	14,962,205	**10,424,120**

TABLE 2.3.3: Compression ratios of different reads compression algorithms with respect to 8-bit encoding. Although PathEnc uses a reference for compression, it was not considered while computing the compression ratio of PathEnc.

Dataset	2-Bit	SCALCE	fastqz	PathEnc	NRRC
D1	4	6.14	4.79	**9.04**	6.05
D2	4	14.74	7.83	**29.57**	19.01
D3	4	16.89	N/A	24.82	**29.05**
D4	4	58.95	15.18	86.61	**128.65**
D5	4	11.06	6.33	17.91	**21.61**
D6	4	12.79	6.91	19.00	**23.31**
D7	4	19.58	14.45	31.07	**46.45**
D8	4	8.76	7.53	10.64	**10.74**
D9	4	19.67	8.88	33.56	**48.17**

2.4 Simulation Results and Discussion

2.4.1 Experimental setup

We have compared our algorithm with the best known algorithms currently existing in the domain of reads compression. In this section we summarize the results. All the experiments were done on an Intel Westmere compute node with 12 Intel Xeon X5650 Westmere cores and 48 GB of RAM. The operating system running was Red Hat Enterprise Linux Server release 5.7 (Tikanga). NRRC compression and decompression algorithms are written in C++ and standard Java programming language, respectively. To compile the C++ source code we used g++ compiler (gcc version 4.6.1) with the -O3 option. Java source code was compiled and run by Java Virtual Machine (JVM) 1.6.0.

2.4.2 Datasets and algorithms used for comparisons

We have employed real datasets in our evaluation. Real datasets used are Illumina-generated short reads of various lengths. The nine experimental datasets listed in Table 3.2.1 have been taken from Sequence and Read Archive (SRA) at NCBI. To prove the effectiveness of our algorithm, we choose two different types of data. Datasets D1 to D4 consist of RNA-seq reads generated from transcriptomes of different species (i.e. human (D1-D2), mouse (D3), and bacterium (D4)). The rest (i.e. D5 to D9) are short read datasets generated from DNA molecules of different organisms.

We have compared our algorithm NRRC with three other well-known algorithms based on biological reads. Currently, MFCompress [41], PathEnc [34], SCALCE [32], and fastqz [23] are some of the most efficient reads compression algorithms available in the literature. Every algorithm we have compared against, except for PathEnc, is a *de novo* compression

algorithm. PathEnc needs a reference genome to generate a statistical, generative model of reads. It is then employed in a fixed-order context, adaptive arithmetic coder. It does not align reads with the reference. In this context PathEnc falls in between reference based and non-reference based reads compression algorithms.

2.4.3 Discussion

Now we discuss how we have used other methods to compare with our algorithm. SCALCE version 2.7 executable was used with its default parameters. It encodes sequence data without considering the positions of Ns. As SCALCE is targeted for compressing FASTQ files, it generates three output files with extension .scalcen (for read names), .scalcer (reads), and .scalceq (qualities). We report only the size of .scalcer file it produced. Fastqz compression tool can compress FASTQ files using a refernce genome. We report the results of the *de novo* version of fastqz compression algorithm. It produces three files namely .fxh (header/metadata information), .fxb (reads), and .fxq (quality scores). The file size we report is the size of the .fxb file. Fastqz was not able to run on D3 dataset. PathEnc was used with its default parameter settings. The file sizes reported are the sizes of all its output files. Since PathEnc needs a reference sequence to build the model, we have provided a specific biological sequence of interest for each of the datasets. For datasets D1-D4 we used a set of human transcriptomes as the reference (as was done by PathEnc). We used Sanger-assembled genomic sequences of interest for the rest. MFCompress [41] was also run with specific parameter settings. But the results have been omitted due to its consistently poor performance.

We have done extensive experiments to realize that our algorithm NRRC is indeed an effective and competitive reads compression tool. Please, see Table 3.2.2 and Table 3.3.1

for detailed simulation results. Table 3.2.2 and Table 3.3.1 present the compressed sizes and compression ratios produced by different algorithms including NRRC, respectively. The algorithm that has the best compression ratio is shown in bold face. Clearly, our proposed algorithm is competitive and performs better than all the best known algorithms in a majority of the datasets. Specifically, NRRC produces poor results in D1 and D2 datasets. Although PathEnc uses a reference for compression, we did not add the compressed size of the reference in the end result. If we add the compressed size of the reference for each of the end results, NRRC will perform better than PathEnc on every dataset. NRRC does not record the identifiers of the reads in the compressed file (similar to other algorithms we have compared). So, the reads will not be in the same order as in the original file. It also discards duplicate reads (similar to PathEnc) if any. Clearly, these will not affect any downward analysis of reads. Since SCALCE and fastqz are FASTQ compression algorithms, these algorithms compress metadata, reads, and quality scores in a single run. So, for a fair comparison we have not shown the run times of our algorithm. NRRC is a single-core algorithm. On the contrary PathEnc is a multi-core algorithm. If we consider linear speed-up and CPU-hour, NRRC is generally faster than PathEnc.

2.5 Conclusions

Data compression is a vital problem in biology especially for NGS data. Five different NGS data compression problems have been identified and studied in the literature. In this research work we have presented a novel algorithm for one of these problems, namely, reference-free reads compression. From the simulation results it is evident that our algorithm indeed achieves compression ratios that are better than those of the currently best known

algorithms. We plan to investigate the possibility of employing the techniques we have introduced in this research work for solving the other four compression problems.

Chapter 3

Hybrid Error Correction Algorithm for Short Reads

Sequencing technology has advanced rapidly. Millions to billions of short reads are sequenced from a DNA molecule in a single run by parallelizing the whole procedure. Since it is a very cost effective procedure and can be performed in a laboratory environment within a brief period of time, we see an explosion of the biological sequencing data. But there is a tradeoff between the abundance and accuracy of the sequencing reads. The limitations of the sequencing technology result in errors in the reads. The errors could be substitution(s), insertions and/or deletions in a single base or multiple bases. Although the errors are being greatly reduced with the advancement of the modern technology, it is still a serious concern as of today. The sequence assembler often fails to sequence the entire genome because of the errors in the reads. By identifying and correcting the erroneous bases of the reads, not only can we achieve high quality data but also the computational complexity of many biological applications can be greatly reduced. Traditional approaches employ overlaps among the reads to correct them. Biologists have successfully sequenced thousands of species and

this effort is growing continuously. As a result, the list of species for which references are available is growing rapidly. Considering this fact we have developed a novel hybrid error correcting algorithm called REFECT (REFerence based Error CorrecTion). We also call it HECTOR (Hybrid Error CorrecTOR). It employs both referential and *de novo* error correction techniques to correct errors in reads. We have done extensive experiments to reveal that REFECT is indeed an effective error correction algorithm.

3.1 Introduction

In the next-generation sequencing (NGS) technology short reads are generated by fragmenting the DNA molecule in random positions. After performing several rounds of this fragmentation and sequencing, multiple overlapping reads are obtained. Utilizing the overlapping information an assembler attempts to reconstruct the entire genomic sequence. If reads contain errors, the assembler will not be able to align the overlapping reads to build a continuous stretch of sequence. The resulting overlap graph contains multiple disconnected components. Each such component corresponds to a set of overlapping substrings of the genomic sequence and leads to a *contig*. Obtaining the exact orientation and precise order of the contigs is a very challenging and computationally intensive task. This step is widely known as *scaffolding*. If the errors can be detected and removed correctly, an assembler will produce a small number of high quality contigs. The less the number of contigs the less will be the amount of time needed to perform scaffolding accurately. If the reads contain erroneous bases, it will be very difficult and often impossible to find the structural variations among the individuals of the same species. Identification of structural variations among individuals is very important as many of these variants may be linked to debilitating diseases

in humans. As a consequence error correction is a very crucial step in sequence assembly.

We can define the error correction problem as follows. The input to the error correction problem is a set of biological sequencing reads from the same genome. The output will be the same set of reads with errors corrected. The erroneous reads should be identified and corrected with a very high confidence. Due to limitations in the sequencing technology there may be three types of errors introduced in the sequencing process. These are substitutions, insertions, and deletions. The types of error introduced solely depend on the sequencing technology used to produce the reads. If the NGS technology of interest produces reads of variable length, it can introduce insertion and/or deletion errors in the reads along with substitutions. For example, Illumina/Solexa generated reads are of fixed length and thus only have substitution errors. In our algorithm we specifically attempt to identify and correct the errors introduced by Illumina/Solexa sequencing technology and thus we consider only substitution errors in the reads. In fact most of the algorithms proposed in the literature assume only substitution errors.

In this research work we propose a novel paradigm for error correction. The basic idea is to use a reference sequence (whenever it is available). To illustrate this paradigm we have developed an effective and efficient error correction algorithm called REFECT (REFerence based Error CorrecTion). At the beginning REFECT exploits a reference genome to find and correct perfectly aligned reads. In the next phase REFECT corrects unaligned reads with the help of corrected reads found in the first phase using a novel *de novo* error correction algorithm. **To the best of our knowledge, the paradigm of using both reference and non-reference based information is novel and has not been used before.** In this context we redefine the definition of the error correction problem. We are given a set of adequately overlapping reads and the corresponding genome of the same species. The error correction problem is to identify and correct the reads that contain errors with a high

confidence.

Now we provide a brief outline of the proposed hybrid error correction algorithm. There are 2 basic and fundamental phases involved in REFECT. In the first phase REFECT aligns the given set of reads onto a known reference genome of the same species within a certain mismatch threshold. The uniquely aligned reads called *perfect* reads are corrected using not only by considering the reference genome but also by considering the possible mutations that might have occurred in the same genomic positions. The reason that some of the reads that cannot be aligned perfectly could be the presence of insertions and/or deletions in the reference genome or that the reads are very erroneous. These unaligned reads are called *imperfect reads*. The *imperfect* reads are then corrected using a *de novo* error correction technique. Both *perfect* and *imperfect* reads are employed in this phase to correct the *imperfect* reads.

3.2 Related Works

Correction of short reads is one of the most challenging and critical tasks in the domain of sequence assembly. To realize the relevant importance researchers invest a lot of time to identify and correct the erroneous reads. As a consequence numerous algorithmic techniques exist in the current literature specially targeting to correct short reads generated from next-generation sequencing platforms. The techniques used for error correction can be broadly categorized into three basic types: k-spectrum based, multiple sequence alignment (MSA)-based, and suffix tree/array based. Next we provide a survey of some of the best-known algorithms of each type.

3.2.1 *k*-spectrum based algorithms

The algorithms in this category are based on decomposing each read into overlapping substrings of a fixed length k. Each substring is called a k-mer and the set of all k-mers is termed as k-spectrum [45]. [45] and [46] first incorporated a k-spectrum based error correction algorithm into an assembly tool known as Euler SR. The steps involved here are to find the trusted (i.e., most probably true) k-mers from the input data and correct each read containing only sequences from the spectrum. According to [45] a k-mer is considered solid if its number of occurrences exceeds a predefined threshold and insolid otherwise. Reads containing insolid k-mers are then transformed to solid k-mers using a minimum number of edit operations. A slight variation of this algorithm is proposed by [47]. [48] presents a CUDA (Compute Unified Device Architecture)-enabled parallel version of spectral alignment based error correction proposed by [45] and [46]. Quake [49] follows the same k-mer spectrum based error correction scheme as described above. In addition to this, it introduces quality values and rates of specific miscalls computed from each sequencing project. Reptile [50] also incorporates k-mer spectrum approach and exploits quality information to identify erroneous bases with a very high probability. It corrects errors by simultaneously examining possibilities to correct erroneous reads by employing a Hamming distance based approach and contextual information between neighboring reads. The algorithmic approach of [51] is very similar to Reptile. Musket [52] is an efficient multi-stage k-mer based error correction tool specially designed for Illumina generated short reads. It employs Bloom filter to count the number of occurrences of all non-unique k-mers. To correct errors Musket employs three mechanisms namely two-sided conservative correction, one-sided aggressive correction and voting-based refinement. RACER [53] and BLESS [54] are two of the best k-spectrum based error correctors.

3.2.2 Multiple sequence alignment (MSA) based algorithms

In this approach after finding the probable aligned reads, the consensus of the aligned reads are computed by incorporating multiple sequence alignments techniques. The errors are then detected and corrected based on the columns of the consensus. Some of the early error correction tools using multiple alignments include but not limited to MisEd [55] and Arachne [56]. Coral [57] and ECHO [58] are two of the most recently developed multiple alignments based algorithms. Coral starts by indexing all the k-mers and their reverse complements. It records each k-mer and a list of reads associated with the k-mer by creating a hash table. After indexing, each read is taken as a base read and is aligned with other reads that share at least one k-mer with the base read. Needleman-Wunsch algorithm is used to align and build the consensus from the aligned reads. It then detects and corrects the erroneous bases using the consensus.

3.2.3 Suffix tree/array based algorithms

Multiple sequence alignment based approaches are time intensive. SHREC [59] and HiTEC [60] avoid the computation of multiple alignments by traversing a suffix tree/array data structure. SHREC is based on the generalized suffix tree. On the contrary HiTEC is based on the more space efficient suffix array data structure. One of the variants of SHREC is Hybrid-SHREC [61].

3.3 Materials and Methods

There are 2 basic phases in REFECT. In the first phase a set S_P of perfect reads from the entire space S of reads is identified. Perfect reads are those reads that can be uniquely aligned onto a reference genome G within d mismatches. d can be obtained either from the average error rate known for the sequencing technology of interest or from empirical evidence. Here error rate refers to the probability of a base being incorrectly changed to another base due to limitations of the sequencing technology. Reads in S_P are then corrected. In the second phase of REFECT, imperfect reads i.e., the reads that cannot be aligned onto the reference within d mismatches, are corrected using both perfect and imperfect reads. We denote the set of imperfect reads as S_I. A good number of reads fall into S_I either because of insertions and/or deletions in the reference genome or because the reads are highly erroneous. Details of our algorithm including the time complexity are provided next.

3.3.1 Correcting perfect reads

As the genomic similarities among the different organisms of the same species are very high, we can utilize this fact to correct some erroneous bases of sequencing reads. It can be achieved by aligning the reads with a known reference genome of the same species within some mismatches. An erroneous base can then be corrected by replacing the corresponding base of the reference genome. But there are some potential pitfalls in this approach. Some bases can be mutated and we may change the correct bases wrongly. So, we have to be careful to avoid wrong corrections. There are 2 basic steps in the first phase of REFECT.

Finding perfect and imperfect reads

In this step the given set of reads S is aligned onto the known reference genome G of the same species within d mismatches. RMAP is employed for reads alignment but the alignment can be done by using any reads alignment tool existing in the current literature. Assume that a read R can be aligned in multiple locations in the reference genome G within d mismatches. In this case it may not be possible to identify a unique location for placing R. So, if a read R can be aligned to more than one location in the reference genome G, we do not consider R as a potential perfect read and call it an 'unaligned' read. If a read R cannot be aligned at all in any location of G, then again R is called 'unaligned'. A read is put into S_P only if it can be aligned in a unique position in G. All the other reads will be put into S_I and are treated as imperfect reads. Please, note that R_P and R_I will refer to a perfect read and an imperfect read, respectively.

Correcting perfect reads

Consider a prefect read R_P that can be uniquely aligned onto the reference genome G within d mismatches. We can correct the d mismatched bases of R_P by replacing it with the corresponding bases of G. But some mismatches could have occurred due to mutations, e.g., single nucleotide polymorphisms (SNPs, in short). If we blindly correct mismatched bases, the corrections could be erroneous and more importantly the variations between one individual and another cannot be identified. So, we need some guards against this error-prone replacement. As it is nearly impossible to know the exact positions of the possible mutations in the genomic sequence for every species, we employ a very simple strategy to infer a possible mutation site. It is done as follows. For a given reference G, we have five counters namely C_A, C_C, C_G, C_T, and C_N for each of the positions in G. They correspond

to the number of occurrences of A, C, G, T, and N, respectively in a given position of G. Whenever a read R_P is aligned within d mismatches, we increment the respective counters of the aligned positions in G. After aligning all the reads of S_P we can create a consensus string (i.e., the assembled genome) by concatenating the most occurring genome base through a traversal of the counters of each position. However, while correcting a mismatched base of R_P, we employ five counters of the corresponding position in G. If the fraction of occurrence of the most occurring base is $\geq \tau$, we correct the old base with this new one otherwise we correct it with the corresponding base in G. Here τ is an appropriately chosen threshold. The base N in R_P is always corrected by the corresponding base in G.

By employing this simple strategy stated above, we can preserve variations between two organisms/individuals with a high probability. If a variation (i.e., SNP) occurs in a genomic position between two individuals, it must be reflected by the reads covering that specific position of the genomic sequence. Suppose we have constructed the genomic sequence T from the given set of reads S and x is the position where two individuals (i.e., genomic sequences G and T) differ. Let A and C be the bases at position x of G and T, respectively. After aligning the set of reads S onto the reference G, let S' be the set of reads that cover position x of G. As we have a variation at x, the positions of reads in S' that are aligned at x of G must have base C. Some of them can have other than C due to erroneous base calling. If we measure the frequency of each base of S' that coincides with the position x, the most frequent base will be C with a high probability. If the fraction of occurrences of C is $\geq \tau$, we replace every base of S' with C in x. Otherwise, we replace it with base A. This is how the variations are preserved between two individuals.

3.3.2 Correcting imperfect reads

We can identify and correct erroneous bases of a read R_I using reads that sufficiently overlap with R_I. After collecting such overlapping reads, we can build a consensus sequence by aligning them. The consensus then can be used to identify and correct erroneous bases in R_I. In fact every error correction algorithm exploits this basic scheme to correct the errors. Specifically, the more the number of overlapping reads of R_I, the more can we be confident in correcting the errors in R_I. The main challenge lies in identifying overlapping reads correctly and efficiently. We employ a hashing based scheme to identify overlapping reads of R_I. Correction of any imperfect read R_I happens next. The algorithm designed to correct imperfect reads is a variation of the proposed algorithm in [18]. Next we briefly describe the algorithm. There are 3 steps in this phase of REFECT.

Finding potential neighbors of imperfect reads

Two different hashings are employed to identify overlapping reads of each imperfect read R_I. In the first hashing we decompose each read R from S into overlapping substrings of fixed length k_1. These k_1-mers are then hashed into a hash table H. In this setting similar k_1-mers fall into the same bucket in H. So, each bucket represents similar reads with respect to at least one k_1-mer. For every read R_I from S_I we collect potential neighbors from the buckets where at least one k_1-mer of R_I falls into. All the reads that fall into at least one of the buckets that R_I falls into will be potential neighbors of R_I. Since no hash function is perfect, two dissimilar k_1-mers might fall into same hash bucket. Thus the potential neighbors of R_I are not necessarily the reads that overlap with R_I.

One more hashing is performed by generating k_2-mers from each read of S. In the second hashing scheme we also obtain potential neighbors for each read R_I as described above. The

potential neighbor lists collected for each read R_I from the two different hashings are then merged together. Note that two similar k-mers may be hashed into two different buckets. Also, two dissimilar k-mers might be hashed into the same bucket. As a result, we might miss some potential neighbors that could help in correcting a read. By employing two different hashing schemes we can maximize the chances of finding as many of the neighbors as possible for each of the reads.

Finding neighbors of imperfect reads

Since the selection of potential neighbors of R_I is solely based on similar k-mers shared between them, it does not necessarily indicate that they are coming from the same genomic region. Two dissimilar k-mers can be hashed into the same hash bucket. So, after collecting potential neighbors of each read R_I, we eliminate those potential neighbors that are not likely to come from the same genomic region. The elimination is done by considering the length of the overlapping region and the Hamming distance between R_I and its neighbors. This is the most time consuming step in the REFECT algorithm. Let R' be a potential neighbor that has a sufficient overlap with R_I. In the elimination step we compute the Hamming distance between the two reads in the overlapping region. If this distance is less than a predefined threshold, we keep R' as a neighbor of R_I. If not, we eliminate R' from the neighbor list of R_I. Since there can be more than one k-mer shared among the reads, the same two reads might fall into more than one bucket together. In this case we identify and use the largest overlap between the pair.

Aligning and correcting imperfect reads

Let $L(R_I)$ be the list of neighbors of R_I. To correct each erroneous base of R_I using $L(R_I)$, we align each R' from $L(R_I)$ onto R_I using our greedy alignment algorithm [18]. This is done by aligning similar k-mers between R' and R_I. But there can be more than one k-mer shared by R' and R_I. While processing the buckets of the hash table H, we keep track of the $k-$mer for which the size of the overlapping region between R_I and R' is the longest and at the same time the Hamming distance between them is within the predefined threshold d. We align R_I and R' based on this k-mer. We perform a greedy alignment that does not take much time [18]. After aligning the reads in $L(R_I)$ with R_I, we build the consensus string and correct each base of R_I using this consensus.

Note that in this step even though we only correct imperfect reads, both perfect and imperfect reads are used to perform the correction. Thus the neighbor list $L(R_I)$ of R_I contains both perfect and imperfect reads. In the alignment step we align all the reads of $L(R_I)$ with R_I. Since the perfect reads should be error free with a very high probability, larger weight has been assigned for perfect reads than imperfect reads while correcting R_I. For example, while correcting a specific position of R_I, we look at all the characters in the reads of $L(R_I)$ that occur in the same position. From out of these characters, any character from an imperfect read will be given a weight of 1 and any character from a perfect read will be given a weight of w (for some $w > 1$), while calculating the consensus character for this position. When the coverage (i.e., the size of the neighboring list $L(R_I)$ of R_I) is high, we choose a smaller value for w than when the coverage is low. Steps of the algorithm are shown in Algorithm 3.1.

Algorithm 3.1: REFerence based Error CorrecTion (REFECT)

Input: A set S of reads, a reference genome G, Hamming distance d and a threshold τ
Output: A set S_C of corrected reads
begin

1. Align the given set S of reads onto the reference genome G within d mismatches. Alignment can be done using any existing reads alignment tool. If a read R can be uniquely aligned in G, then R is called perfect and put into S_P. Every other read is imperfect and goes to S_I.
2. Create five counters namely C_A, C_C, C_G, C_T, and C_N for each of the positions in G. C_A, C_C, C_G, C_T, and C_N correspond to the number of occurrences of A, C, G, T, and N bases, respectively, in a given position of G.
3. Whenever a perfect read R_P is aligned within d mismatches, increment the respective counters of the aligned positions in G. While correcting a mismatched base of R_P, we will employ the five counters of the corresponding position in G.
4. For each read $R_P \in S_P$, if the fraction of occurrences of the most occurring base in an aligned position of G is $\geq \tau$, we correct the old base of R_P with this new one otherwise we correct it with the corresponding base in G. The base N in R_P is always corrected by the corresponding base in G.
5. Generate k-mers of each read $R \in S$ and hash the reads based on these k-mers. Equal k-mers fall into the same bucket. If R is any read, any other read that falls into at least one of the buckets that R falls into is treated as a potential neighbor of R. For every imperfect read R_I create a list $P(R_I)$ of potential neighbors.

   Perform this step twice with two different values for k and for every read merge the lists of potential neighbors from the two hash tables. If the size of a bucket is larger than a threshold, only a subset of the bucket is included in the potential neighbors identification process.
6. Let R_I be any imperfect read. Align every read in $P(R_I)$ with R_I. Let R' be any read in $P(R_I)$. If R_I and R' overlap sufficiently and if in the overlapping region the Hamming distance between R_I and R' is small, then we treat R' as a neighbor of R_I. For every read R_I construct a list $L(R_I)$ of neighbors of R_I in this fashion.
7. R_I is corrected using the reads in $L(R_I)$. Greedily align R' with R_I for every $R' \in L(R_I)$. R_I is corrected by taking a consensus across every column in the alignment. Perform this step for every imperfect read R_I.

3.3.3 Complexity analysis

In this section we analyze the time complexity of REFECT. Let n be the number of reads, r be the read length, g be the reference genome length, and c be the coverage.

In Step 1 of REFECT, we can align a read in $O(g)$ time and hence Step 1 can be completed in $O(gn)$ time. Step 2 takes $O(g)$ time. Step 3 takes $O(nr)$ time. Step 4 also takes $O(nr)$ time.

Let n_1 be the number of perfect reads and n_2 be the number of imperfect reads (with $n_2 = n - n_1$).

In Step 5 we build hash tables. The number of k-mers (k could either be k_1 or k_2) generated from each read is $r - k + 1$. Let $h(.)$ be the has function employed. We think of the hash table as an array of buckets (or lists). Each bucket has an integer as its index. If

the size of the array is N, then the index of any bucket is an integer in the range $[1, N]$. If k is small enough, one could employ direct hashing such that each k-mer is hashed into a bucket whose index is the k-mer itself (thought of as an integer). In this case, the hash array should be of size 4^k. If k is large, direct hashing may not be feasible. The expected size of each bucket is $\frac{(r-k+1)n}{N} = O\left(\frac{rn}{N}\right)$. The total time spent in building the hash tables is $O(rn)$.

In Step 5 we also find potential neighbors of each read. A read falls into at most $r-k+1 < r$ buckets and hence the expected number of potential neighbors for each read is $O\left(\frac{r^2n}{N}\right)$. For every bucket we spend an expected $O\left(\left(\frac{rn}{N}\right)^2\right)$ time. Thus the total time spent in finding potential neighbors has an expected value of $O\left(\frac{r^2n^2}{N}\right)$.

In Step 6 we align reads. Specifically, if R_I is any imperfect read and $P(R_I)$ is the list of potential neighbors of R_I, then the expected size of $P(R_I)$ is $O\left(\frac{r^2n}{N}\right)$. For every read $R' \in P(R_I)$, we align R' with R and compute the Hamming distance between R and R' in the overlapping region. Thus for every $R' \in P(R_I)$ we spend $O(r)$ time. As a result, the total time spent in Step 6 for each read is expected to be $O\left(\frac{r^3n}{N}\right)$. Summing this over all the reads, the total expected time spent in Step 6 is $O\left(\frac{r^3nn_2}{N}\right)$. This is also the expected time spent in Step 7.

In summary, the expected run time of REFECT is:
$O\left(gn + \frac{r^3nn_2}{N} + \frac{r^2n^2}{N}\right)$.

3.4 Results

The effectiveness of our algorithm REFECT has been evaluated by comparing it with some of the state-of-the-art algorithms namely, Coral, Racer, and BLESS. We have evaluated REFECT on a number of Illumina/Solexa datasets and compared the results with the aforemen-

tioned error correction algorithms. The simulation results show that our proposed algorithm is indeed very effective and competitive. More details follow.

3.4.1 Datasets

We have employed real datasets in our evaluation. Real datasets used are Illumina-generated short reads of various lengths and coverages (Please, see Table 3.2.1). The ten experimental datasets listed in Table 3.2.1 have been taken from the Sequence and Read Archive (SRA) at NCBI. Reference genomes are Sanger assembled bacterial genomes of various kinds. Please, note that G_R, $|G|$, $|R|$, and $|r|$ refer to accession number of the reference genome, genome length, total number of reads, and read length, respectively.

3.4.2 Experimental setup

All the experiments were done on an Intel Westmere compute node with 12 Intel Xeon X5650 Westmere cores and 48 GB of RAM. The operating system running was Red Hat Enterprise Linux Server release 5.7 (Tikanga). To compile the C++ source code we used g++ compiler (gcc version 4.6.1) with the -O3 option. Time was measured by taking the CPU clock time which gives the instruction level elapsed time a program takes.

3.4.3 Evaluation metrics

We measure the effectiveness of our proposed algorithm REFECT by taking 2 different performance evaluation techniques based on (1) *de novo* assembly of the corrected reads and (2) alignment of the corrected reads onto a known reference genome.

Evaluation metrics based on *de novo* assembly

One of the main applications of the short reads is to *de novo* assemble the reads to get the underlying genomic sequence. Correction of the reads could help the *de novo* assembly process in a number of ways. Every *de novo* assembler available in the literature is based on constructing a graph representing the overlaps among the reads and performing a relevant walk in this graph. If the reads contain less errors, then the graph construction procedure will not only be easier and more accurate but also will take less amount of time. In addition, the memory consumption will also be reduced drastically since the number of spurious overlaps will decrease.

Several statistical measures have been introduced to assess the performance of *de novo* assemblers including, but not limited to, N50 statistics, number of contigs, mean length of the contigs, and coverage of the contigs. These measures could also be used to determine the effectiveness of read correction algorithms. The idea is to perform *de novo* assembly using the reads before and after error correction. We define one of the most used statistics next.

1. *N50:* The N50 statistic is an indicator on the quality of contigs produced by any assembler. N50 is similar to a mean or median, but gives a greater weight to longer contigs. Given a set of contigs, each with its own length, the N50 length is defined as the length for which the collection of all contigs of that length or longer contains at least half of the sum of the lengths of all contigs, and for which the collection of all contigs of that length or shorter also contains at least half of the sum of the lengths of all contigs. The more the N50 length the more accurate the assembler is.

To evaluate the effectiveness of REFECT on *de novo* assembly we employ Edena assembler [63]. Although Velvet [64] or SOAPdenovo [65] are some of the widely used *de novo*

assemblers, we use Edena primarily because Velvet and SOAPdenovo employ their own correction procedure while processing the graph. On the contrary Edena does not employ any correction scheme and is thus more suitable for evaluating the effectiveness of any error correcting algorithm.

Evaluation metrics based on alignment

Another measure of the effectiveness is to align the corrected reads onto the known reference genome of the same species and count the number of variations (i.e., mismatches). Although this procedure is routinely used by every error correction algorithm for performance evaluation (see e.g., [45]), it has some serious drawbacks. First of all variations between two individuals are counted as substitution errors. So, if an error correction algorithm changes some true bases and it conforms with the reference genome, its performance gain will be falsely increased. In the first phase of our proposed method we keep the variations by utilizing the majority voting scheme and in the next phase we increase the weight of each base of the perfect reads. It ensures to keep variations with a very high confidence. One advocacy in favor of the alignment based performance evaluation is that the variations among the individuals of the same species are very small and so, it can reflect the effectiveness of the error correction methods. In this context, we have used RMAP (v2.05) introduced by [66]. It efficiently aligns short reads with a known genome by minimizing mismatches. For evaluating the accuracy we need to align as many corrected reads as possible so that the result will be correct with a high confidence. Keeping this in mind, to align the resulted reads from the error correction tools, we have allowed up to 10 mismatches per read for all of the datasets listed in Table 3.2.1.

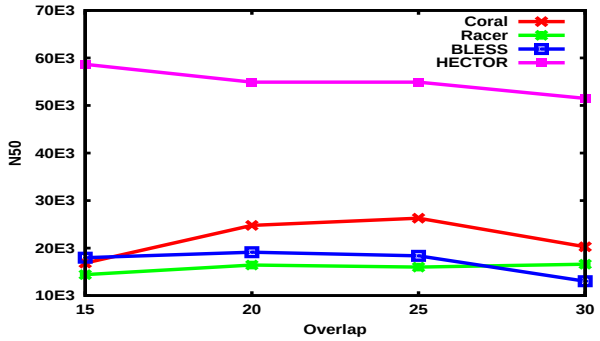
TABLE 3.4.1: Real Datasets from Sequence Read Archive (SRA)

Dataset	Name	Accession	G_R	$\|G\|$	$\|r\|$	$\|R\|$	Coverage
D1	*T. pallidum*	SRR361468	CP002376.1	1,139,417	35	7,093,045	217
D2	*E. coli*	SRR001665	NC_000913.2	4,639,675	36	10,408,224	80
D3	*L. lactis*	SRR088759	NC_013656.1	2,598,144	36	4,370,050	60
D4	*P. aeruginosa*	SRR396641	NC_002516.2	6,264,404	36	9,306,557	53
D5	*E. coli*	SRR022918	NC_000913.1	4,771,872	47	6,740,651	68
D6	*B. subtilis*	DRR000852	NC_000964.3	4,215,606	75	1,744,210	31
D7	*E. coli*	SRR396532	NC_000913.2	4,639,675	75	4,341,061	70
D8	*E. coli*	SRR396536	NC_000913.2	4,639,675	75	3,453,957	55
D9	*L. interrogans L*	SRR353563	NC_004342.2	4,338,762	100	3,530,694	81
D10	*L. interrogans C*	SRR397962	NC_005823.1	4,277,185	100	3,559,265	83

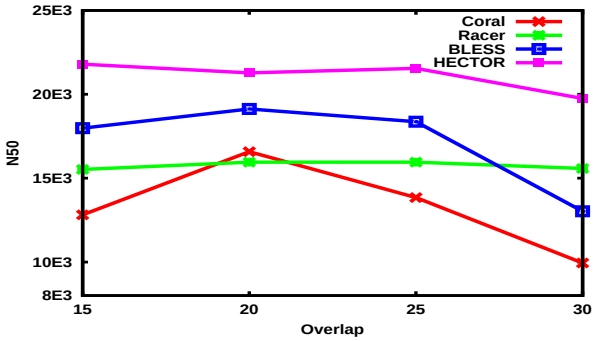
TABLE 3.4.2: Performance evaluation on real sequencing datasets

Dataset	Method	% Sensitivity	% Accuracy	% Mapped Reads	CHD	CPU Time (m)
D1	Coral	50.53	50.31	95.42	12,51,680	99.95
	Racer	93.68	93.49	**96.61**	2,38,631	**7.69**
	BLESS	92.03	91.99	96.34	2,28,545	10.31
	REFECT	**98.51**	**98.40**	96.48	**49,973**	11.25
D2	Coral	61.68	61.14	96.45	7,35,597	67.68
	Racer	94.25	94.09	**97.11**	1,78,568	**9.18**
	BLESS	91.78	91.77	96.93	1,68,292	14.29
	REFECT	**97.31**	**97.14**	97.04	**67,330**	14.90
D3	Coral	71.06	70.64	95.79	2,65,080	40.98
	Racer	96.98	96.87	95.81	45,839	12.58
	BLESS	94.33	94.10	95.74	53,344	**5.42**
	REFECT	**97.75**	**97.70**	**96.00**	**23,152**	17.88
D4	Coral	81.57	79.61	98.16	70,240	21.87
	Racer	93.97	93.04	98.24	52,293	**3.74**
	BLESS	89.72	88.64	98.12	36,819	15.45
	REFECT	**95.39**	**95.13**	**98.37**	**21,381**	9.77
D5	Coral	33.68	33.22	65.32	46,60,501	141.64
	Racer	93.92	93.89	**83.94**	15,01,057	13.93
	BLESS	93.34	93.33	71.99	7,03,399	18.24
	REFECT	**97.42**	**97.35**	80.60	**5,19,788**	**11.45**
D6	Coral	74.40	74.07	92.54	2,30,589	28.47
	Racer	93.42	93.30	**94.90**	1,07,599	**6.40**
	BLESS	95.11	95.11	93.55	53,511	7.43
	REFECT	**98.63**	**98.58**	94.51	**19,956**	7.50
D7	Coral	61.56	61.23	91.08	13,63,593	98.89
	Racer	89.97	89.77	94.99	3,76,446	10.38
	BLESS	94.98	94.95	90.99	1,83,656	17.02
	REFECT	**98.87**	**98.83**	**97.01**	**92,586**	**9.83**
D8	Coral	67.53	67.25	93.78	9,25,374	207.13
	Racer	88.87	88.75	96.09	4,50,519	13.68
	BLESS	94.17	94.12	93.34	1,61,075	13.18
	REFECT	**98.89**	**98.85**	**97.75**	**59,584**	**12.85**
D9	Coral	87.61	83.22	89.41	1,44,128	48.05
	Racer	93.94	93.67	89.95	59,603	**9.10**
	BLESS	96.54	96.46	89.60	27,100	17.25
	REFECT	**97.79**	**97.64**	**90.12**	**23,068**	13.78
D10	Coral	89.34	83.28	90.14	1,07,208	233.33
	Racer	94.20	93.81	90.79	50,336	**12.63**
	BLESS	95.88	95.75	90.49	22,993	18.12
	REFECT	**97.25**	**97.06**	**90.98**	**22,290**	16.83

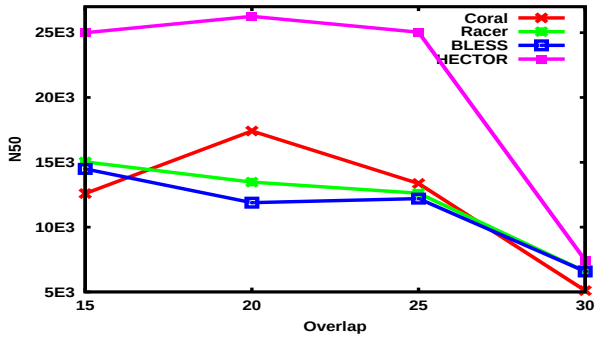
At first the error correction method of interest is given the whole set of reads. The resulting reads are then aligned onto the reference genome within 10 mismatches using RMAP and compute various performance metrics based on this alignment. A number of metrics have been introduced in the literature for judging the performance of any error correction algorithm. TP is a measure of how many erroneous bases have been corrected while FP is the number of true bases that have been changed incorrectly. TN shows how



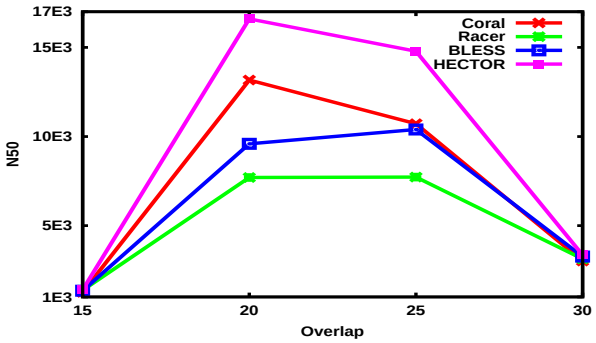
(A) N50 values for different overlap cutoffs of D1



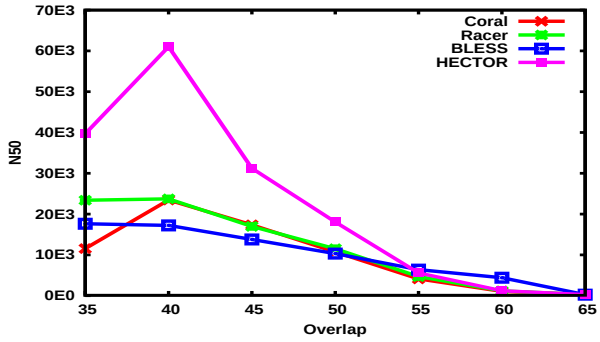
(B) N50 values for different overlap cutoffs of D2



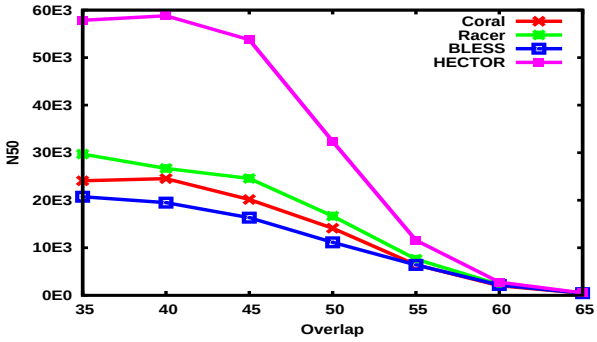
(C) N50 values for different overlap cutoffs of D3



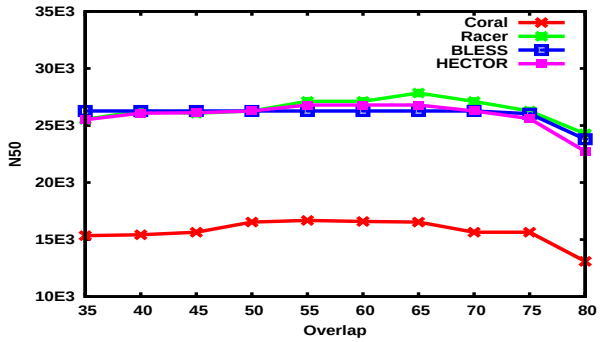
(D) N50 values for different overlap cutoffs of D4



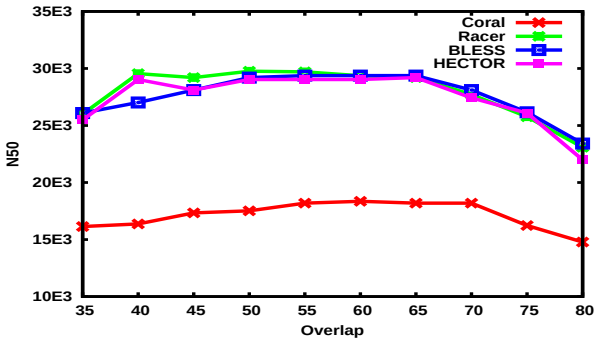
(E) N50 values for different overlap cutoffs of D6



(F) N50 values for different overlap cutoffs of D7

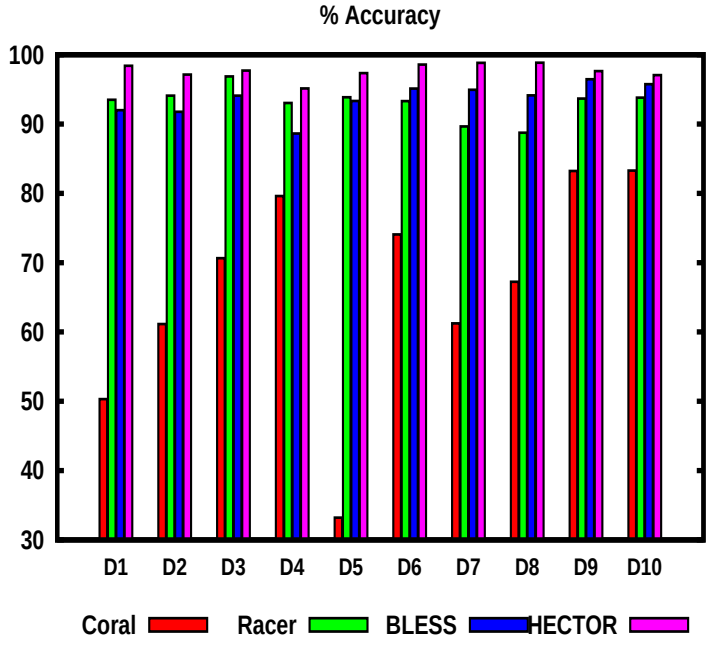


(G) N50 values for different overlap cutoffs of D9

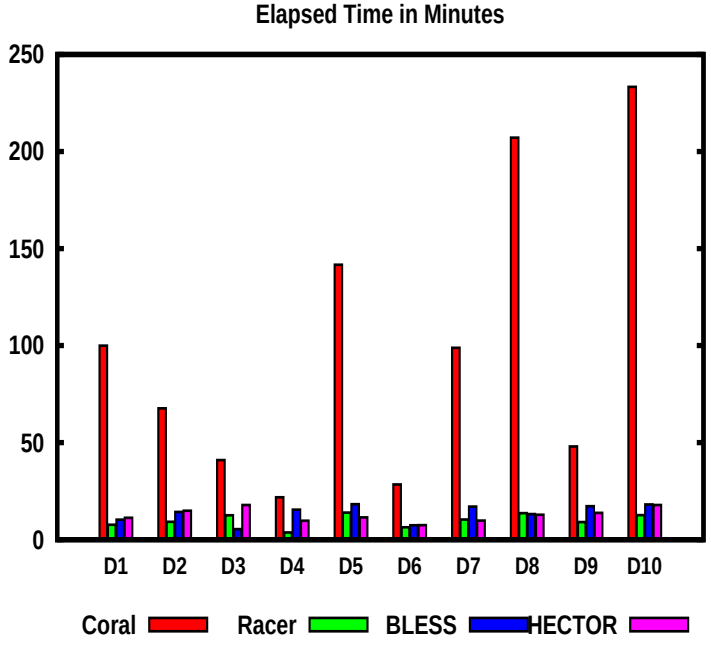


(H) N50 values for different overlap cutoffs of D10

FIGURE 3.4.1: N50 values for different overlap cutoffs of various datasets.



(A) % Accuracy of different algorithms



(B) Elapsed time of different algorithms

FIGURE 3.4.2: % Accuracy and elapsed time of different algorithms including REFECT for datasets D1-D10.

many true bases remain unchanged while FN is the number of erroneous bases that have not been detected by the algorithm. Using these statistics we define the following evaluation metrics next.

1. *Sensitivity:* Sensitivity (also called the true positive rate, or the recall rate) measures the proportion of actual positives which are correctly identified as such (e.g., the percentage of sick people who are correctly identified as having the condition). In this context sensitivity is defined as $\frac{TP}{TP+FN}$.

2. *Specificity:* Specificity (sometimes called the true negative rate) measures the proportion of negatives which are correctly identified as such (e.g., the percentage of healthy people who are correctly identified as not having the condition). So, the specificity is $\frac{TN}{TN+FP}$.

3. *Accuracy:* Accuracy indicates the fraction of errors effectively removed from the experimental dataset, i.e., $\frac{TP-FP}{TP+FN}$.

4. *Cumulative Hamming Distance (CHD):* After aligning a read R onto the genomic sequence, we calculate the Hamming distance d between the aligned read and the corresponding sequence in genome. Adding all such d for all the reads R, we get *CDH*. It reflects how close the corrected reads are to a genomic sequence of the same species in terms of substitution errors.

5. *% Mapped Reads:* The fraction of reads from the entire space of reads aligned onto the reference genome with up to d mismatches.

Please note that if the accuracy of an algorithm is large (i.e., close to 100), it is very effective in correcting errors. A negative value of accuracy indicates that the method of interest introduces more errors than it corrects.

3.4.4 Parameters configuration

An algorithm always should tune its parameters with respect to a given dataset. Our algorithm has a set of parameters that are tuned automatically. Keeping this in mind we took the default parameter values for the different error correction methods that we have used for comparison.

- *Coral:* Standard parameters.
- *Racer:* Appropriate genome length of interest.
- *BLESS:* Standard parameters.
- *REFECT:* No parameters to be selected. Parameters are empirically estimated based on an analysis of the input data. For example k_1 and k_2 varies from 14 to 17 and 20 to 30, respectively. Hamming distance ranges from 1 to 3, etc. The method has an interface where parameters can be fine tuned by the users if they are not satisfied with the results.

3.4.5 Discussion

We have compared our algorithm with 3 other well-known algorithms based on real sequencing reads. We have done extensive and rigorous experiments to realize that REFECT is indeed an effective and efficient error correction tool in this domain. Real sequencing data are taken from the Sequence Read Archive (SRA) as described above.

Next we discuss the N50 values for different overlap cutoffs given by the Edena. Edena takes the reads corrected by the algorithms of interest and outputs N50 values, contigs, min, max values, etc. The N50 values for REFECT-corrected reads exceed all other assemblies

for all the datasets except for D9 and D10 (Please, see Figure 1.7.1). It is comparable with Racer and BLESS for datasets D9 and D10. N50 values of REFECT-corrected reads are approximately 3× and 1.5× higher than that of other assemblies for dataset D1 and D3, respectively (Please, see Figure 1.7.1). For overlap cutoffs 35 to 45 N50 values of REFECT-corrected reads of datasets D6 and D7 are approximately 2× higher than that of other assemblies.

The alignment based performance evaluations for the datasets listed in Table 3.2.1 can be found in Table 3.2.2. For all the datasets D1 to D10 REFECT performs better than all the other algorithms in terms of sensitivity, accuracy, and cumulative Hamming distance *CHD*. Please, note that sensitivity and accuracy of REFECT are above 97% for all the datasets except D4. In some datasets e.g., D1, D6-D8, the sensitivity and accuracy are higher than 98%. *CHDs* of REFECT on an average are $2-5\times$ less than that of Racer and BLESS. Also, REFECT consumes less CPU cycles compared to all the other algorithms in some datasets such as D5, D7, and D8. As stated above we align the corrected reads onto the reference genome within 10 mismatches to find the % mapped reads. The reads from REFECT are aligned more than those from the other algorithms for datasets D3, D4 and D7-D10. It is slightly less than that of Racer for other datasets. Specificities are identical (i.e. 99%) for every algorithm across all the datasets, we did not include it in Table 3.2.2. Figure 3.4.2 presents a visual comparison of the different algorithms for all the datasets.

Please note that the run time of our algorithm is slightly higher than that of Racer in some datasets. Our *de novo* correction procedure is solely based on the greedy pair-wise alignments where a number of reads are aligned with the read to be corrected. This is a very time consuming step. Since our algorithm always takes nearly the same amount of time to perform error correction on various datasets (Please see Table 3.2.2), we believe that by introducing and implementing an efficient data structure, the runtime of REFECT can be

minimized significantly.

3.5 Conclusions

In this research work we have proposed an efficient, scalable, and robust error correction algorithm for correcting short reads. The steps of REFECT can be broken into two independent tasks. At first the reads are aligned with known genome sequence of the same species within certain mismatches and correct the aligned reads (i.e., perfect reads) accordingly. At the next step it builds k-mers of two different lengths and hashes the different k-mers into two different hash tables. By traversing the lists of the hash tables it finds the neighbors of each of the imperfect reads. Each imperfect read is then corrected using the neighbors of that read. We have introduced a number of techniques to correct reads more effectively. We have compared our algorithm with three other state-of-the-art algorithms and the experimental results reveal that REFECT is indeed effective and competitive.

Chapter 4

Genome-wide Splicing Events Detection Algorithm

RNA Sequencing (RNA-seq) based on next-generation sequencing (NGS) technology enables transcriptome analyses of entire genomes at a very high resolution. Due to limitations of the sequencing technology the reads are very short and erroneous. As a consequence it is a very challenging task to accurately map RNA-seq reads onto the genome and identify the splice junctions. There are two shortcomings in some of the existing best-known algorithms for identifying splice junctions: 1) the junction boundaries are predicted within a large range, 2) they take a long time and 3) the predictions of splice junctions are inaccurate. In this research work we propose a novel algorithm POMP to accurately detect the splice junctions considering all the shortcomings of the existing algorithms as stated above. It generates accurate candidate splice junctions utilizing expected properties around the splice junctions. These candidates are used as examples to train a learner. The model learnt by the learner can be used to identify splice junctions. Extensive experiments were done considering whole human genome splicing events. Experimental results show that POMP is indeed a more

effective and efficient algorithm compared to the other state-of-the-art algorithms in terms of sensitivity, specificity, and runtime.

4.1 Introduction

RNA Sequencing (RNA-seq) refers to the use of next-generation sequencing technologies to sequence complementary DNA (cDNA) in order to get information about a sample's RNA content. In genetics a cDNA is a DNA synthesized from a messenger RNA (mRNA) template in a reaction catalyzed by the reverse transcriptase and DNA polymerase enzymes. It is often used to clone eukaryotic genes in prokaryotes. RNA-seq generates millions of short sequence fragments or reads in a single run within a very brief period of time. The reads obtained from this can then be aligned to a reference genome in order to construct a whole-genome transcriptome map and thus can be used to measure levels of gene expression and to identify novel splice variants of genes. But to fully utilize such reads one needs to be able to accurately align the sequencing reads over the intron boundaries in a reference genome. This mapping problem represents a significant challenge given the small read lengths and inherent high error rates.

One of the main purposes of RNA-seq technology is to detect genome-wide splicing events efficiently and accurately. In molecular biology and genetics, splicing is a modification of a RNA molecule after transcription, in which introns (the parts of DNA sequence that are spliced out) are removed and exons (the parts of DNA sequence after splicing) are joined. This is needed for the typical eukaryotic messenger RNA before it can be used to produce a correct protein through translation. We can define and formulate the problem of interest as follows: Splice junctions are points on a DNA sequence at which superfluous DNA is

removed during the process of protein synthesis in higher organisms. The problem posed is to recognize, given a sequence of DNA, the boundaries between exons and introns, i.e., donor or acceptor sites. It is more formally stated as: Given a position in the middle of a window of DNA sequence elements (nucleotides), decide whether this is an intron-exon boundary, exon-intron boundary or not and align the reads accordingly.

In this research work we propose a new algorithm called POMP to detect genome-wide splicing events. It incorporates representatives from the clusters of highly similar reads and thus reduces the number of initially unmapped reads dramatically. It enhances the chances of detecting splicing events accurately and precisely. It also reduces the run time drastically. After getting the candidate splice junctions (SJs) from gapped alignment, we extract useful features inherent in the alignment and genomic sequence to group candidates into two classes, namely, accurate and inaccurate with a very high level of confidence. Non-linear Support Vector Machine (SVM) is subsequently employed for binary classification.

4.2 Related Works

Many methods and algorithms have been devised to detect genome-wide splicing events. There are two basic procedures for RNA-seq reads alignment without employing gene annotation. Those are *exon inference* and *gapped alignment*. TopHat [67] follows the first approach where it maps the reads to the reference genome using Bowtie [68] and then clusters all the Initially Mapped (IM) reads. IM reads are those reads that can be completely mapped to the reference without being split into several parts. TopHat calls the resulting clusters as islands and each island represents putative exon regions. To recognize potential splice junctions, TopHat first enumerates all the canonical donor and acceptor sites within

the island sequences. After enumeration, it considers all pairing of these sites that could form canonical (GT-AG) introns between neighboring (but not necessarily adjacent) islands. Each possible intron is then checked with the IUM reads using seed-and-extend strategy. IUM reads are those reads that cannot be completely mapped to the reference without being split into several parts. For each of possible splice sites, a seed is formed by appending a small amount of sequence upstream of the donor and downstream of the acceptor sites. Any read containing the seed is checked for a complete alignment to the exons on either side of the possible exons. This approach is proved to be very accurate since it infers exons with a reasonably high coverage. But TopHat may fail to detect splice junctions in a number of cases. For example, when the transcript is located in a region with a low sequencing depth, there might not be enough reads that straddle the junction for easy detection. As another example, TopHat may fail when the exon length is shorter than the read length.

SpliceMap [69] and several other methods [70], [72], [71] and [73] follow the second approach, namely, gapped alignment. There are four main steps in the SpliceMap algorithm. These are half-read mapping, seeding selection, junction search and paired-end filtering. At first each read is divided into halves and SpliceMap maps both halves of the read to the reference genomic sequence with a high probability. This step is done using any existing short alignment tool such as SeqMap [74] or ELAND. If a half-read is mapped exactly within the genomic sequence, it is called a mapped hit and dubbed as seeding. For each seeding, the alignment on the reference genome is then extended base by base to find the splicing point. It subsequently tries to find a partner splicing point that provides a perfect match for the corresponding residual sequence of the original read. Paired-end information of reads is used to filter out false positives. This mapping and extension approach is not an effective way to handle reads with sequencing errors, since the call rate will be low especially when the expression level is low. Also, SpliceMap is not able to effectively deal with reads

that can be mapped to multiple locations. They simply ignore the hits that are too close together. Another paper [75] proposes an algorithm that is similar to SpliceMap. In the alignment procedure every read is split into two non-overlapping parts or "anchors" that are aligned separately. The two anchors are then extended as long as they still match the reference sequence. If a splice junction is located between the gaps of the two anchors, there is a high probability that the two anchors correspond to two exons in the DNA sequence. TopHat2 [76] adapts both the strategies where short reads are mapped using *exon inference* and long reads are mapped using *gapped alignment*.

4.3 Methods

Our algorithm POMP runs in two phases. At the first phase it detects initially unmapped reads (IUMs) and constructs representatives from those IUMs. The representatives are then used to find the candidate splice junctions (SJs) by employing gapped alignment procedure. In the second phase we prune the candidate splice junctions to reduce the false positives with the help of a novel machine learning technique. In this context our algorithm can be thought of as a self-learning algorithm. We first generate a set of candidate splice junctions. These candidates are then further processed to get a set of highly accurate splice junctions. These form the positive examples for the learning step. We also generate a set of negative examples. These examples are then used to train the learner. The model learnt by the learner can then be used to identify splice junctions. Our learning is based on Support Vector Machines (SVMs). Details of our algorithm are provided next.

4.3.1 Finding candidate splice junctions

This phase has three sub-phases. In the first sub-phase we find a set of unmapped reads. Since the coverage of RNA-seq reads are very high, we can build clusters of similar reads. A representative string for each cluster can then be formed by taking the consensus from the overlapping reads. These representatives are naturally longer than the original reads. The bases in the representatives are also highly accurate. This is exactly what is done in the second sub-phase. The representatives are then aligned onto genome using our gapped alignment algorithm.

Generating initially unmapped reads

At first we align the reads onto the genomic sequence G. Any alignment tool can be used to perform this step. We used Bowtie alignment tool to perform this task within a predefined Hamming distance d, where $2 \leq d \leq 3$. The value of d depends on the length of the read. The reads which are aligned onto G within a Hamming distance d are called Initially Mapped (IM) reads and they form the set IM. The rest of the reads are called Initially Un-Mapped (IUM) reads and form the set IUM. Reads in IUM are of two types. The first type of reads in IUM could not be aligned since they span two or more exons in mRNA. The second type of reads in IUM cannot be aligned within a Hamming distance of d due to (more than d) errors. The first type of reads play a key role in detecting the splice junctions.

Generating representatives

In the next-generation sequencing technology millions to billions of short reads are generated in a single run within a very short period of time. Since reads are generated by fragmenting the genome in random positions, the chances of finding highly overlapped reads are very high.

If we manage to cluster highly overlapped IUM reads and create a representative string for each of the clusters, not only will the runtime drastically reduce but also the representatives could be mapped with a high level of accuracy. This is due to the fact that the representatives are built from the consensus of the clustered reads and the representatives are longer than the RNA-seq reads. The details of this sub-phase is provided next.

1. Let $r \in IUM$ be any read from the set of initially unmapped (IM) reads. At first each read $r \in IUM$ is decomposed into overlapping substrings of a fixed length k. These k-mers are then hashed into a hash table H. Each entry (i.e., bucket) $b \in H$ represents a set of similar reads (since they share a k-mer with a high probability).

2. For each read $r \in IUM$ we collect all other reads $L(r)$ that are sufficiently overlapped with r. Specifically, the Hamming distance in the overlapping region is no more than a predefined threshold τ, where the value of τ depends on the length of the overlapping region and $2 \leq \tau \leq 3$. $L(r)$ along with r forms a cluster c of highly overlapped reads coming from the same genomic region of G. Please note that in our procedure each read $r \in IUM$ can be in only one cluster c. Let C stand for the set of all such clusters.

3. For each cluster $c \in C$ we align each r' from $L(r)$ onto r using our greedy alignment algorithm [77]. This is accomplished by aligning similar k-mers between r' and r. But there can be more than one k-mer shared by r' and r. While processing the buckets of the hash table H, we keep track of the $k-$mer for which the size of the overlapping region between r and r' is the longest and at the same time the Hamming distance between them is within the predefined threshold τ as described in the previous step. We align r and r' based on this k-mer. We perform a greedy alignment that does not take much time. After aligning the reads in $L(r)$ with r, we build the consensus string. This consensus string is called in our algorithm as the representative (of c).

Mapping representatives

Each representative is divided into two non-overlapping segments of equal size. Each segment is then mapped onto the genome using Bowtie within a predefined Hamming distance d'. Segments can be mapped in several positions and thus have the potential to detect alternate splicing events. There are three cases to be considered while mapping the segments. Consider a representative S with segments be S_1 and S_2.

1. Segments S_1 and S_2 of the representative S can be mapped onto the genomic sequence G within a Hamming distance d' where $d' \leq 1$. It is possible that S_1 and/or S_2 can be mapped in several positions in G. If the order of the mapping is coherent i.e., the ending position p of S_1 is followed by the starting position q of S_2, (p, q) constitutes a candidate splice junction. The alignment of S_1 and S_2 in this case constitutes a *gapped alignment*. Let n_1 and n_2 be the number of mappings of S_1 and S_2, respectively, that are coherent. Then there will be at most $n_1 \times n_2$ possible candidate splice junctions corresponding to S.

2. Assume that only S_1 is mapped onto G within a Hamming distance of d' where $d' \leq 1$. In this scenario we fix S_1 onto G and extend it to the right by appending one base from S_2 at a time starting from left until the number of mismatches m is within a Hamming distance of d''. The value of d'' is dynamically calculated based on the modified length of S_2. Let S_1 appended with this portion of S_2 be denoted as S_1'. Let the remaining part of S_2 be referred to as S_2'. If S_1' ends in position p, S_2' is then mapped onto G with a starting point that is within the range $[p, p + c]$ where c is a user defined constant. Let S_2' start at q in the genome G. The pair (p, q) constitutes the donor and acceptor sites of the candidate splice junction.

3. This scenario is exactly the same as described above with S_1 replaced by S_2 and vice-

versa. We fix S_2 onto G and extend it to the left by appending one base from S_1 at a time starting from right until the number of mismatches m is within a Hamming distance d'' as described above. If S'_2 starts at position q, S'_1 is then mapped onto G within the range $[q, -(q + c)]$ where c is a user defined constant. Let S'_1 end at p in genome G. The pair (p, q) constitutes the candidate splice junction.

4.3.2 Finding highly accurate splice junctions

We get candidate splice junctions (SJs) from the previous phase. But this candidate set might contain a lot of false SJs which need to be eliminated. In this phase we apply Support Vector Machine (SVM) - a well-known machine learning technique - to classify candidate SJs into two categories, i.e., accurate and inaccurate SJs with a very high level of confidence.

Deriving initial accurate/inaccurate splice junctions

Support Vector Machine (SVM) has been developed by Vapnik emphet al. at AT&T Bell Laboratories [78] [79]. Kernel-based techniques (such as support vector machines, Bayes point machines, kernel principal component analysis, and Gaussian processes) represent a major development in machine learning algorithms. Support vector machines (SVMs) are a group of supervised learning methods that can be applied to classification or regression. They represent an extension to nonlinear models of the generalized portrait algorithm. The basic idea is to find a hyperplane which separates any given d-dimensional data perfectly into two classes. In brief, SVM is a supervised learning model with an associated learning algorithm. It analyzes data and recognizes patterns and is used for classification and regression analysis. Given a set of training examples, each marked with membership information to one of two categories, an SVM training algorithm builds a model. This model assigns unseen examples

into one category or the other, making it a non-probabilistic binary linear classifier. To build the model accurately we need an adequate number of training examples. In our case the candidate junctions will form the examples set. This set contains highly accurate and inaccurate SJs. Although it is very difficult to label the candidate SJs as true or false with 100% confidence without any prior knowledge, we can certainly select a small subset of true and false SJs by carefully investigating some properties around the candidate SJs.

To derive training examples for supervised classification we divide the set of representatives into three disjoint subsets. Uniquely Gapped Representatives (UGRs): the set UGR consists of a subset of the representatives in which each representative has only one canonical SJ (a SJ is canonical if the corresponding intron starts with GT and ends with AG). Multiple Gapped Representatives (MGRs): each representative in this subset MGR has more than one canonical SJs. Non-canonical Uniquely Gapped Representatives (NUGRs): each representative in this subset $NUGR$ must have only one non-canonical or semi-canonical SJ (a semi-canonical SJ is a SJ where the corresponding intron starts with AT and ends with AC or starts with GC and ends with AG). We define accurate and inaccurate spliced junctions next.

1. Initial accurate splice junctions (IASJs)

Each representative in UGR has only one canonical SJ as described above. We retain those SJs in which there is no mismatch in either side of the gapped alignments. We also retain any SJ under the following scenario. Consider any SJ S. Let A refer to the 12 bps to the left of the donor site of S and B refer to the 12 bps to the right of the acceptor site of S. There could be a number of representatives that have the same SJ S and have A as a substring and each such representative has been constructed from a number of IUM reads. If the total number of such reads across all the representatives that have the same SJ S and

share A is ≥ 5 then S is retained. Symmetrically, if the total number of reads across all the representatives that have the same SJ S and share B is ≥ 5 then also S is retained.

2. Initial inaccurate splice junctions (IISJs)

This set is acquired by selecting those candidate SJs which are supported by MGRs but not any UGRs. Those SJs that are supported by only NUGRs and whose mapping length is shorter than 12 bp are also retained. (The mapping length of any SJ S is the length of the shorter of S'_1 and S'_2.)

Deriving features

Every SJ can be represented by some characteristics locally observed around it. These characteristics are called features or dimensions in any classification algorithm. The features described bellow may decide altogether whether a candidate SJ is accurate or inaccurate. In this context let $J(a, b)$ denotes a splice junction where a and b refers the coordinates of donor and acceptor sites, respectively.

1. Exon-intron distribution

The distribution of the bases around exon-intron border in any genome G may give valuable information to classify a candidate SJ more accurately. We devise a novel technique to extract the hidden information around exon-intron boundary by employing k-mer distribution. The distribution is computed by considering highly accurate SJs from IASJs. In this procedure a region of fixed length l around exon-intron boundary of a SJ is extracted from the genome. This region is then decomposed into overlapping substrings of length k. This process is repeated for all SJs in IASJs. We sort the k-mers based on the abundance and take the top 10 k-mers. In a similar way regions of a fixed length l is extracted from a

candidate SJ of interest and they are decomposed into k-mers. If any of these k-mers from a candidate SJ can be found in top 10 list, we score it as 1; otherwise the score is 0.

2. Intron-exon distribution

In this procedure a region of fixed length l around the intron-exon boundary of a SJ is extracted. It is then decomposed into overlapping substrings of length k. This process is repeated for all SJs in IASJs and use the same procedure as described above.

3. Boundary coverage score

Boundary coverage score measures the average coverage depth of donor and acceptor sites together. Initially Mapped (IM) reads are used to compute this score. Since introns are spliced off from genome G to form mRNA, there should be a large concentration of mapped reads to the left and right regions of donor and acceptor sites of G, respectively. If we could formulate a meaningful feature from this observation, it should definitely help us in classifying accurate and inaccurate splice junctions. Let R_i be the total number of IM reads that are mapped onto the genome G starting from position i. We define coverage score of a region as the average number of reads mapped onto that region. Let a region be bounded by l_1 and l_2. The coverage score of this region can be expressed as $C_{score}(l_1, l_2) = \frac{1}{l_2 - l_1} \sum_{i=l_1}^{l_2} R_i$. Score for donor and acceptor regions are defined as $C_{score}(a) = C_{score}(a - c_1, a + c_2)$ and $C_{score}(b) = C_{score}(b - c_2, b + c_1)$, respectively where c_1 and c_2 are user defined constants. Now the boundary coverage score for a junction $J(a, b)$ is the score derived by summing these two scores, i.e., $B_{score} = C_{score}(a) + C_{score}(b)$.

4. Intron coverage score

Initially Mapped (IM) reads are used to calculate an intron coverage score. As introns are spliced out during the formation of the mRNA, it is intuitive that the average coverage

depth of the intron specified by the SJ will be very low. Let a and b be the co-ordinates (i.e., positions) of the donor and acceptor sites, respectively. We calculate the intron coverage score as the average number of reads mapped onto a base in the region bounded by the interval (a, b) as described above. In this scenario the scores for the donor and acceptor regions are defined as $C_{score}(a) = C_{score}(a - c_3, a + c_4)$ and $C_{score}(b) = C_{score}(b - c_4, b + c_3)$, respectively where c_3 and c_4 are user defined constants. Now the intron coverage score for a junction $J(a, b)$ is the score derived by summing these two scores, i.e., $I_{score} = C_{score}(a) + C_{score}(b)$.

5. Donor-acceptor pattern score

Almost 99% of the novel splice junctions contain canonical donor-acceptor sites (i.e. AC-GT). Very few of the junctions contain semi-canonical and non-canonical donor-acceptor sites. Based on this observation, a high score s_1 is given to the corresponding SJ if it contains canonical donor-acceptor sites. A medium score s_2 and a low score s_3 are given for semi-canonical and non-canonical sites, respectively.

6. Mismatches

It is the mean number of alignment mismatches of all the representatives mapped onto a candidate $J(a, b)$ of interest. Let $\Re$ denote the set of representatives mapped on to $J(a, b)$. Suppose each representative contributes m_i mismatches in the gapped alignment. The mismatch score is defined as the average number of mismatches of all the representatives while aligning onto $J(a, b)$, i.e., $M_{score} = \frac{1}{|\Re|} \sum_{i=1}^{|\Re|} m_i$.

7. Shortest mapping length

It is the maximum length of the shorter side of the gapped alignment among all the representatives around a candidate junction $J(a, b)$ of interest.

8. Junction overlapping number: It is defined as the number of UGRs mapped onto a candidate junction $J(a, b)$.

Classifying splice junctions

At the beginning SVM builds a learning model using training examples from IASJs and IISJs. As the representatives in our algorithm are highly accurate and long compared to IUM reads, the number of accurate SJs are larger than those of inaccurate SJs. If we consider all the SJs from IASJs and IISJs, SVM will produce a biased model. To develop an unbiased model, we randomly pick SJs from IISJs which are nearly equal in number to SJs in IASJs. After building the model we classify each of the candidate SJs as accurate or inaccurate and output accurate SJs.

4.4 Results

The effectiveness of our algorithm POMP has been evaluated by comparing it with the currently best-known algorithm TopHat2. Although MapSplice, SpliceMap, GSNAP [82], STAR [83], and RUM [84] are some of the state-of-the-art algorithms in this domain, they all produce poorer results compared to TopHat2. So, we have compared our algorithm with TopHat2 considering it as a benchmark. The experimental results show that our proposed algorithm is indeed very effective and competitive. More details follow.

TABLE 4.4.1: Genome-wide perfomance evaluation for all datasets.

Dataset	Read Length	Junctions	Method	Predicted	True	False	SN (%)	SP (%)	GN (%)	CPU Hour
D1	50 bp	140,856	TopHat2	137,701	121,482	16,219	86.25	88.22	87.23	25
			POMP	131,295	122,543	8,752	**87.00**	**93.33**	**90.16**	**17**
D2	75 bp	140,856	TopHat2	147,712	130,257	17,455	**92.48**	88.18	90.33	17
			POMP	126,841	117,087	9,754	90.05	**92.31**	**91.18**	**10**
D3	100 bp	140,856	TopHat2	153,630	133,463	20,167	**94.75**	86.87	90.81	25
			POMP	140,084	127,855	12,229	90.77	**91.27**	**91.02**	**18**

TABLE 4.4.2: Chromosome-wide performance evaluation for dataset D1 for first 10 chromosomes.

Chromosome	Simulated Junctions	Method	Predicted	True	False	SN (%)	SP (%)	GN (%)
1	14,162	TopHat2	13,868	12,177	1,691	85.98	87.81	86.90
		POMP	13,134	12,153	981	**86.15**	**92.71**	**89.43**
2	10095	TopHat2	9,523	8,406	1,117	83.27	88.27	85.77
		POMP	9,235	8,524	711	**84.43**	**92.3**	**88.37**
3	8,223	TopHat2	7,858	7,143	715	86.87	90.90	88.88
		POMP	7,968	7,369	599	**89.61**	**92.48**	**91.05**
4	5,181	TopHat2	4,790	4,309	481	83.17	89.96	86.56
		POMP	4,908	4,471	437	**86.29**	**91.09**	**88.69**
5	6,095	TopHat2	5,734	5,293	441	86.84	92.31	89.58
		POMP	5,701	5,371	330	**88.12**	**94.21**	**91.17**
6	6,562	TopHat2	6,350	5,664	686	86.32	89.20	87.76
		POMP	6,312	5,859	453	**89.28**	**92.82**	**91.05**
7	6,830	TopHat2	7,094	5,838	1,256	**85.48**	82.29	83.89
		POMP	6,041	5,465	576	80.01	**90.45**	**85.23**
8	4,650	TopHat2	4,409	4,009	400	**86.22**	90.93	88.57
		POMP	4,134	3,919	215	84.28	**94.8**	**89.54**
9	6,330	TopHat2	6,157	5,513	644	87.09	89.54	88.32
		HASM	5,907	5,515	392	**87.12**	**93.36**	**90.24**
10	5,482	TopHat2	5,278	4,639	639	**84.62**	87.89	86.26
		POMP	4,785	4,505	280	82.18	**94.15**	**88.17**

TABLE 4.4.3: Genome-wide performance evaluation for Initially Accurate Splice Junctions (IASJ).

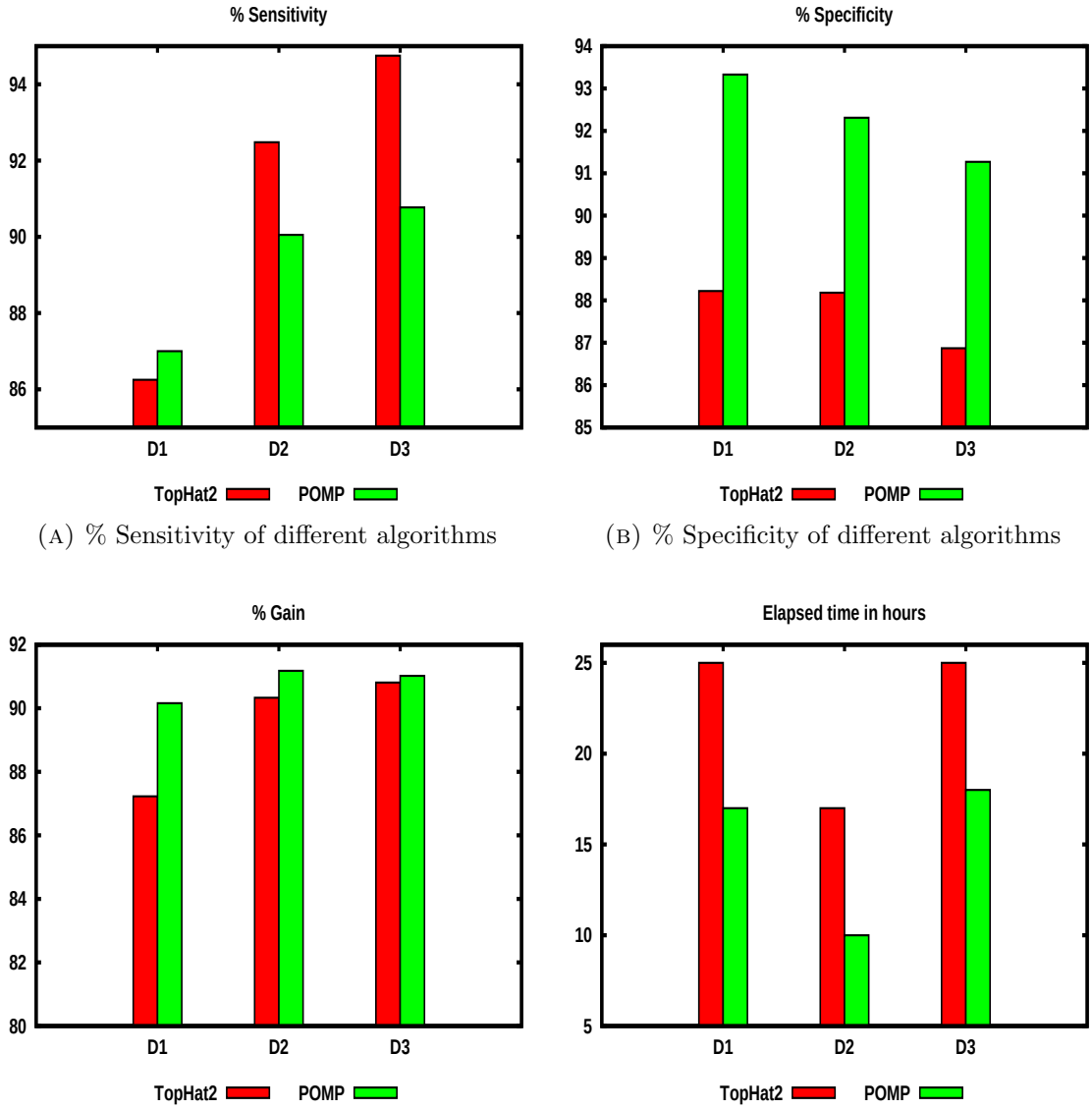
Dataset	Read Length	Predicted	True	False	SN (%)	SP (%)	GN (%)
D1	50 bp	103,895	99,025	4,870	70.30	95.31	82.81
D2	75 bp	107,714	103,150	4,564	73.23	95.76	84.50
D3	100 bp	113,734	108,516	5,218	77.04	95.41	86.23

4.4.1 Experimental datasets

Three datsets each having 40 million reads were simulated. Read lengths of these datasets are 50, 75, and 100 bp long, respectively. Maq [80] was used to generate simulated Illumina reads from human genome hg19 with a realistic error rate of 0.02. The number of reads were also proportional to the human datasets based on UCSC Known Gene Model. Thus the test datasets are nearly similar to the real transcriptome sequencing data. Expression levels were estimated using Cufflinks [81] which was based on isoforms defined by UCSC Known Gene Models.

4.4.2 Experimental environment

All the experiments were done on an Intel Haswell 2.60 GHz compute node with 24 Intel Xeon E5-2690 cores and 128 GB of RAM. The operating system running was Red Hat Enterprise Linux Server release 5.7 (Tikanga). The POMP algorithm has been implemented in C++ and standard Java programming language. To compile the C++ source code we used g++ compiler (gcc version 4.6.1) with the -O3 option. Java source code is compiled and run by Java Virtual Machine (JVM) 1.8.0_31.



(A) % Sensitivity of different algorithms

(B) % Specificity of different algorithms

(C) % Gain of different algorithms

(D) Elapsed time of different algorithms

FIGURE 4.4.1: Performance evaluations of different algorithms including POMP for datasets D1-D3.

4.4.3 Evaluation metrics

We measure the effectiveness of our proposed algorithm POMP using four different metrics. These metrics are defined below.

1. Sensitivity

The fraction of the simulated junctions correctly detected. Let the number of simulated and predicted junctions be T and P, respectively. Now, the Sensitivity can be written as $SN = \frac{P}{T}$.

2. Specificity

The fraction of the true junctions among all predicted junctions. Let the number of true junction detected by the algorithm of interest be N. Specificity now can be formulated as $SP = \frac{P}{N}$.

3. Gain

It is defined as the average of sensitivity and specificity. Please note that both sensitivity and specificity are important measures. Gain is a simple way to combine these two measures. It can be written in terms of SN and SP as $GN = \frac{SP+SN}{2}$.

4. Time

Measured elapsed time using total number of CPU clock cycles consumed by each of the algorithms of interest.

4.4.4 Outcomes

Consider the dataset D1 of 50 bp reads (please, see Table 4.4.1). The overall gains of POMP and TopHat2 are 90.16% and 87.23%, respectively. POMP is approximately 3% better than TopHat2 in this case. The sensitivity and specificity of POMP are better than those of TopHat2. The specificity of POMP and TopHat2 are 93.33% and 88.22%, respectively. POMP has an around 5% better specificity. This means that POMP not only predicts more

SJs but also detects true SJs more precisely. The runtime of POMP is also very impressive. It took 17 CPU hours where TopHat2 took 25 CPU hours. We show chromosome wide experimental evaluations of dataset D1 for first 10 chromosomes. Please, see Table 4.4.2 for more details. It is evident that chromosome wide gains of POMP are better than those of TopHat2. Results for datasets D2 and D3 are shown in Table 4.4.1. The overall genome wide gains of POMP are better than those of TopHat2. POMP also took less time to detect potential splice junctions. POMP took 10 CPU hours whereas TopHat2 took 17 CPU hours for dataset D2. On dataset D3 POMP took 18 CPU hours while TopHat2 took 25 CPU hours. Please, see Figure 1.7.1 for visual details.

4.5 Discussion

To the best of our knowledge POMP is the first algorithm in the domain of splice junction discovery which employed consensus strings (i.e., representatives) of highly overlapping reads coming from the same genomic region. It not only enhances the accuracy of the junction discovery but it also drastically reduces runtime. The limitations of the sequencing technology results in errors in the reads. The errors could be substitution(s), insertion(s) and/or deletion(s) in a single base or multiple bases. Although the errors are being greatly reduced with the advancement of the modern technology, it is still a serious concern as of today. By identifying and correcting the erroneous bases of the reads, not only can we achieve high quality data but also the computational complexity of many biological applications can be greatly reduced. When we build a representative from a cluster of similar reads, the bases of the representative are chosen by taking the consensus. As a result the representatives are naturally error-free with a very high probability and hence the alignment can be done with

a high accuracy.

We choose the overlapping regions of the reads in the clusters in such a way that the lengths of the representatives are no more than 1.2 times of the read length. This ensures that similar reads forming clusters are originating from the same genomic regions. Please note that some clusters can have only a single read. It is due to the fact that the coverage is very low in that region of mRNA. In this case a single read in a cluster forms a representative by definition. Generally representatives are longer than the read and thus splice junctions detections are very accurate. Consider the results shown in Table 4.4.3. It is based on the computation of initial accurate splice junctions. Without any statistical analysis and significant computation POMP can detect a subset of splice junctions with a very high level of accuracy. The specificity is always above 95% for all of the datasets. The overall gain for each of the datasets is also very impressive.

Our algorithm POMP detects donor sites and acceptor sites of SJs within a very high resolution. The true boundaries of SJs can be found by POMP within 20 bp resolution. On the other hand Tophat2 detects boundaries of SJs within 50-100 bp resolution. More specifically, let $J(p, q)$ denote a splice junction with p and q being the donor and acceptor sites in the genome G, respectively. Let p' and q' be the outputs of POMP corresponding to this junction. Then POMP guarantees that p' and q' will be bounded by $p-20 \leq p' \leq p+20$ and $q - 20 \leq q' \leq q + 20$, respectively. TopHat2 detects donor and acceptor sites of SJs very poorly within 20 bp resolution. For example on the D1 dataset TopHat2 discovers only 42 simulated SJs (out of 140,856 true SJs) within 20 bp resolution. The time complexity of our algorithm is also quite impressive. It runs 60-70% faster than TopHat2 and achieves a better accuracy.

To investigate the effectiveness of our algorithm POMP, we have done extensive experiments by considering 3 simulated RNA-seq datasets of different read lengths. The RNA-seq

datasets were generated in accordance with the real transcriptome data by introducing realistic gene and error model. The abundance of the simulated reads are also in proportion to the real reads. Each dataset contains 40 million reads and aligned with the hg19 human genome. Both sensitivity and specificity are important metrics to judge the performance of any algorithm. Thus we need a measure that combines these two into a single measure. Keeping this in mind we have introduced the gain metric as the average of sensitivity and specificity. From the experimental analysis it is evident that POMP is better than TopHat2 in terms of gain and running time. TopHat2 has better sensitivities on D2 and D3 datasets but lower specificities. It indeed detects a lot of alternating splicing events but with greater false positives. On the contrary POMP is more restricted in deciding true alternating splicing events. It balances between sensitivity and specificity and thus its overall gains are better than those of TopHat2 on all the datasets.

There are two issues in the proposed algorithm that have to be further investigated. At this point (1) POMP does not exploit the advantage of paired-end reads and (2) it divides the representatives into two equal halves. If POMP could exploit the advantage of paired-end reads, the specificity would be further increased. Since in this setting we know the relative distance between two representatives apriori, the gapped alignments will be more accurate and the number of false positives will be drastically reduced. On the other hand a single representative can stretch over two splice junctions. If we divide a representative into two halves and the coverage is low in that region we might potentially miss these junctions. By combining these two ideas we can further increase the sensitivity and specificity of our proposed algorithm. We plan to work on these ideas in future.

4.6 Conclusions

In this research work we have proposed a novel algorithm POMP to detect splicing events more accurately with a very short period of time compared to other stat-of-the-art algorithms. At the beginning similar sequencing reads from initially unmapped reads are clustered. From individual cluster *representatives* are formed which are naturally more accurate and longer than the given set of reads with a very high level of confidence. These *representatives* are then used to detect the splice junctions. SVM is used to prune candidate splice junctions by classifying them into two classes, i.e., accurate or inaccurate. Experimental results show that POMP is indeed effective and efficient compared to other best-known algorithms in this domain.

Chapter 5

Efficient and Scalable Scaffolding Algorithms

In the next generation sequencing techniques millions of short reads are produced from a genomic sequence at a single run. The chances of low read coverage to some regions of the sequence are very high. The reads are short and very large in number. Due to erroneous base calling, there could be errors in the reads. As a consequence, sequence assemblers often fail to sequence an entire DNA molecule and instead output a set of overlapping segments that together represent a consensus region of the DNA. This set of overlapping segments are collectively called contigs in the literature. The final step of the sequencing process, called scaffolding, is to assemble the contigs into a correct order. Scaffolding techniques typically exploit additional information such as mate-pairs, pair-ends, or optical restriction maps. In this research work we introduce a series of novel algorithms for scaffolding that exploit optical restriction maps (ORMs). Simulation results show that our algorithms are indeed reliable, scalable, and efficient compared to the best known algorithms in the literature.

5.1 Introduction

To conduct basic biological research such as but not limited to diagnostic, biotechnology, forensic biology, biological pathways and knowledge of DNA sequences has become inevitable. Scientists need to know the sequence of bases to reveal genetic information that is hidden in a particular segment of a DNA molecule. For example, they can use sequence information to identify which stretches of DNA molecule contain genes, as well as analyze those genes to detect potential changes in the sequence that may cause diseases. So, to obtain an in-depth knowledge of a particular DNA molecule, sequencing of that molecule is the primary step. DNA sequencing is any process that is used to map out the precise order of the nucleotides within a single strand of a DNA molecule. The structure of DNA was modeled as a double helix in 1953. The first notable method for sequencing DNA was developed during the 1970s known as Sanger sequencing. It is a method of DNA sequencing based on the selective incorporation of chain-terminating dideoxynucleotides by DNA polymerase during in vitro DNA replication [90] [91]. It was developed by Frederic Sanger and his colleagues in 1977 and was the most widely used sequencing technology until the advent of NGS technologies. An alternative to Sanger was shotgun sequencing [92] [93]. By the time the Human Genome Project (HGP) began in 1990, only a few scientific laboratories had the ability to sequence a mere 100k bases, and the total cost of sequencing remained very high. Since then, technological improvements and computerized automation have increased the sequencing speed and lowered the cost to the point where individual genes can be sequenced routinely, and some laboratories managed to sequence well over 100 million bases per year. Beginning in the late 1990s, the scientific community has developed a number of new DNA sequencing technologies including the first of the "next-generation" sequencing methods.

High-throughput or next-generation sequencing technologies parallelizes the sequencing

process and produce thousands or millions of short reads (25-100 bp) simultaneously at a single run. Some of the sequencing technologies dominating the NGS market today are Massively parallel signature sequencing (MPSS), 454 pyrosequencing, Illumina (Solexa) sequencing, SOLiD sequencing, Ion semiconductor sequencing, etc. An introductory review of these techniques can be found in [85]. After generating NGS reads, they can either be assembled *de novo* or aligned to a known reference sequence [86]. The choice solely depends on the biological application of interest as well as cost, effort, and time constraints. For example, if the intended application of interest is to determine a complete genomic sequence of a new species, we have to follow *de novo* sequencing strategy. On the contrary, identifying genetic variations in multiple strains of highly related genomes can be accomplished by aligning NGS reads to their reference genomes. This approach is cheaper and faster than *de novo* sequencing. But there are some limitations and challenges associated with this alignment approach. One of the most important challenges is in placing the reads within repetitive regions in the reference genome. Besides this, some of the regions existing in the source genome may not even exist in the reference genome. This could happen because of gaps in the reference genome [87]. The problem of aligning reads in repetitive regions can be solved by exploiting mate-pair reads information. *De novo* sequencing techniques also face challenges in repetitive regions and from low read coverages that result in gaps in the constructed sequence. The former can be overcome by employing mate-pair reads [88] or optical restrictions maps [89] information and the later can be solved increasing the read coverage.

In sequencing DNA is first shredded randomly into numerous smaller fragments. The resulting fragments are sequenced using the chain termination method to obtain reads. Multiple overlapping reads for the target DNA are obtained by performing several rounds of this fragmentation and sequencing. The resulting reads of these fragments are then reassembled

into their original order based on overlaps. Reassembly is done by a computer program ultimately yielding the complete and continuous sequence. A contig is a series of overlapping DNA sequences used to make a physical map that reconstructs the original DNA sequence of a chromosome or a region of a chromosome. It is a set of overlapping DNA segments that together represent a consensus region of DNA. If the coverage is large enough and the sequenced reads are error free, there should be only one contig containing the entire genome. But in the next generation sequencing technologies the coverage can be low resulting in gaps and the reads also can be erroneous. As a consequence sequence assemblers typically produce multiple contigs. Obtaining the exact orientation and precise order of the contigs is the next challenging task. This step is known as scaffolding.

In genomic mapping, a scaffold is a series of contigs that are in the correct order but not necessarily connected in one continuous stretch of the genomic sequence. So, a scaffold is not only composed of contigs but also gaps. The problem of finding the correct order of the contigs can be posed as the problem of finding a permutation of these contigs that optimizes an objective criterion. Scaffolding is known to be NP-hard. Any information about the orderings such as the sizes of fragments of the DNA molecule can indeed help in devising an efficient algorithm for scaffolding. We can get fragment size information by employing restriction enzymes. A restriction enzyme (or restriction endonuclease) is an enzyme that cuts DNA at or near some specific recognition nucleotide sequences known as restriction sites. Restriction enzymes are of three types and found in bacteria and archaea. A restriction enzyme acts against invading viruses by electively cutting up a foreign DNA in a process called restriction. In general, restriction enzymes recognize a specific sequence of nucleotides and produce a double-stranded cut in the DNA. The recognition sequences usually vary between 4 and 8 nucleotides, and they are generally palindromic sequences. The locations of these specific sequences of nucleotides on a DNA molecule are called restriction sites. A

restriction map detects known restriction sites within a sequence of DNA by cleaving it with a specific restriction enzyme. A restriction map provides a number of fragment sizes which collectively serve as a unique "fingerprint" or "barcode" for that sequence [94]. An optical restriction map (ORM) [95] is also similar to a restriction map with only one difference. It provides an ordered list of fragment sizes and this method has been combined with the assembly process to sequence whole genomes. Some of the recent research on ORMs in the context of contigs assembly can be found in [96], [220], [98], or [89].

We have employed ORMs in the context of scaffolding to find the relative order and correct placement of contigs produced by sequence assemblers. In this research work we propose several algorithms for scaffolding. We use a two phase strategy for scaffolding (just like the authors of [89]). In the first phase we compute a score for each contig corresponding to each possible placement of the contig in the ORM. In the second phase we utilize the scores computed in the first phase to come up with a non-overlapping placement of (possibly a subset of) the contigs in the ORM. In brief, we transform each contig into an ordered sequence of fragment sizes (just like the ORM). A greedy scoring scheme is then applied to find a score for each contig for each possible placement of the contig in the ORM. Greedy placement algorithms are then used to place the contigs in a correct order by using the matching scores. To validate the robustness of our proposed algorithms we have introduced different types of errors. Our simulation results on both real and synthetic data show that our algorithms are indeed scalable and efficient in terms of both accuracy and time.

5.2 Methods

There are two phases in our algorithm. In the first phase we compute a score for each contig corresponding to each possible placement of the contig in the ORM. In the second phase we utilize the scores computed in the first phase to come up with a non-overlapping placement of the contigs in the ORM. These two phases are described in Sections 5.2.1 and 5.2.2, respectively.

5.2.1 A scoring scheme

Overview

To effectively and correctly order the contigs we need a reliable scoring mechanism. As the genomic sequence can be composed of millions or even billions of characters, we should also consider the time spent by the proposed algorithms. There is a trade off between the time an algorithm takes and the accuracy it gives. We achieve a very good balance between these two. This is done by carefully formulating the scoring algorithm. For each contig, we generate an ordered list of fragment sizes. Since we know the sequence of the restriction enzyme from the ORM of the genomic sequence, we can easily find the ordered restriction fragment sizes of any contig by incorporating *in silico* digest of the restriction enzyme. The resulting list of ordered fragment sizes can be mapped with the ORM. Assuming that there are no errors either in the ORM or the fragment sizes of the contigs, for any given ordered fragment sizes of a contig, in general, there will exist a subset of matching ordered fragment sizes in the ORM. Exploiting this information we can order the contigs. But in a real world scenario the data often may not be error free. Errors could occur due to the omission of some restriction sites or a change in some fragment sizes (due to sequencing errors). To quantify

the effect of the errors a scoring mechanism is introduced.

Let $A = \{c_1^i, c_2^i, \ldots, c_{n_i}^i\}$ be the set of the ordered *in silico* fragment sizes of a contig C_i and $B = \{o_l, o_{l+1}, o_{l+2}, \ldots, o_{m-l+1}\}$ be the set of ordered fragment sizes of a particular region of the ORM stretching from the l^{th} fragment to the $(m-l+1)^{th}$ fragment. The score of the contig C_i for the region stretching from the l^{th} fragment to the $(m-l+1)^{th}$ fragment of the ORM is defined as follows:

$$Score(C_i) = \left| \sum_{j=1}^{n_i} c_j^i - \sum_{j=l}^{m-l+1} o_j \right| + P * MRS \tag{5.2.1}$$

where P and MRS are the penalty term and number of missed restriction sites, respectively. Penalty term P is user defined and should be very large. Under ideal circumstances where there are no errors in reads, there are no errors in the ORM, the assembly is perfect, etc., we should not tolerate any missed restriction sites. In this case P could be even ∞. But in practice, depending on the technology employed, we could expect to see some errors in every process. As a result, we have to use a finite penalty. The value of P will thus depend on the error rates in the different technologies. If the expected error rate is low, then P has to be large. If the expected error rate is high, then P has to be low. In our experiments a value of 999 for P seems to yield good results.

More details on our algorithms are given in the next section.

A greedy scoring algorithm

The input to the *Greedy Scoring* algorithm are an ORM of the genomic sequence of interest, an ordered list of fragment sizes for each contig, and a penalty term. The fragment sizes may not be known exactly. Each fragment size in general can be thought of as a random variable for which we know the mean and the standard deviation. For simplicity assume

that the standard deviation is the same (say σ) for all the fragment sizes. The algorithm proceeds greedily to calculate the score of each contig. In fact, the algorithm computes multiple scores for each contig. If m is the number of fragment sizes in the ORM, then the algorithm computes m scores for each contig.

Let $o_1, o_2, \ldots, o_m$ be the fragment sizes in the ORM. Let C be any contig and let the fragment sizes of C be $c_1, c_2, \ldots, c_n$. A score for C is computed by matching c_1 with o_1; Another score is computed by matching c_1 with o_2; and so on. In other words, we compute a score for C by matching c_1 with o_i for each possible value of i, $1 \leq i \leq m$. What is the score when c_1 is matched with o_i (for some specific value of i)? We correlate a prefix of C (of minimum length) with a prefix of $o_i, o_{i+1}, \ldots, o_m$ such that the two prefix sums are nearly the same (within σ). In other words, we identify the least integer u and an integer q such that $|\sum_{j=1}^{u} c_i - \sum_{j=i}^{i+q-1} o_i| \leq \sigma$. Once we find such u and q, we match c_u with o_{i+q-1}. Now we proceed recursively, i.e., we look for a prefix (of least length) of $c_{u+1}, c_{u+1}, \ldots, c_n$ and a prefix of $o_{i+q}, o_{i+q+1}, \ldots, o_m$ whose sums are nearly the same (up to σ); and so on.

The score for the resultant mapping of the contig C is obtained using Equation 3.3.1. For example, the partial score corresponding to the mapping of c_u with o_{i+q-1} is $|\sum_{j=1}^{u} c_j - \sum_{j=i}^{i+q-1} o_j| + [(u-1) + (q-1)] * P$. Such partial scores are computed and added. Note that when we map c_1 with o_i, the last fragment c_n of the contig will be mapped with some fragment o_t in the ORM. Corresponding to this mapping of the contig C, we refer to o_i as the starting fragment and o_t as the ending fragment. For the base case when $i = m$, we match c_n with o_m. Also when $\sum_{j=1}^{n} c_j > \sum_{j=i}^{m} o_j$ we match c_n with o_m.

More details of the algorithm can be found in Algorithm 2.1. The run time of our greedy scoring algorithm is $O(mnr)$, where m is the number of fragments in the optical map, r is the number of contigs and n is the maximum number of fragments in any contig.

Algorithm 5.1: Greedy Scoring

Input: $OpticalRestrictionMap[1..m]$, $ContigFragmentList[1..r][1..k]$, $Penalty, P$
Output: Contigs with associated scores $CS[1..r][1..m]$
begin

```
1   i ← 1
    repeat
2       j ← 0
3       case ← 0
        repeat
            set matched_sites, contig_frag_size, op_frag_size to 0
4           text_pos ← j + 1
5           pattern_pos, missed_res_sites to 1
            repeat
6               if (case == 0){
7                   contig_frag_size = ContigFragmentList[i][pattern_pos]
8                   op_frag_size = OpticalRestrictionMap[text_pos]
9               } else if (case == 1){
10                  contig_frag_size += ContigFragmentList[i][pattern_pos]
11              } else if (case == 2){
12                  op_frag_size += OpticalRestrictionMap[text_pos]
13              }
14              lower_bound = op_frag_size − std(text_pos)
15              upper_bound = op_frag_size + std(text_pos)
16              if (con_frag_size ≥ lower_bound and con_frag_size ≤ upper_bound){
17                  Increment pattern_pos, text_pos, and matched_sites by 1
18                  case = 0
19              } else if (con_frag_size < lower_bound){
20                  Increment pattern_pos, and missed_res_sites by 1
21                  case = 1
22              } else if (con_frag_size > upper_bound){
23                  Increment text_pos, and missed_res_sites by 1
24                  case = 2
25              }
26              if (pattern_pos ≥ |ContigFragmentList[i][1..k]|){
27                  Calculate score using Equation 1
28                  Insert the score along with the starting and ending positions in CS
29              }
            until pattern_pos ≤ |ContigFragmentList[i][1..k]|;
30          j ← j + 1
        until j ≤ m;
31      i ← i + 1
    until i ≤ r;
32  Return CS[1..r][1..m]
```

5.2.2 Placement schemes

The placement scheme utilizes the matching scores of the contigs to find the correct order. We propose three different placement algorithms that are described below.

Some notations

The list of ordered fragment sizes in the ORM is $o_1, o_2, \ldots, o_m$. The number of contigs is denoted as r. Let the cotigs be $C_1, C_2, \ldots, C_r$. The number of fragments in C_i is denoted as n_i, for $1 \leq i \leq r$. The list of ordered fragment sizes corresponding to C_i is $c_1^i, c_2^i, \ldots, c_{n_i}^i$, for $1 \leq i \leq r$. Let k denote $\max_{i=1}^{r} n_i$.

Greedy placement algorithm 1 – GPA1

GPA1 takes as input the contigs and the ORM together with the output of Algorithm 2.1. If m is the number of ordered fragments in the ORM, then the number of scores associated with each contig will be m, as described in the previous section. The algorithm proceeds as follows: At first the matching scores associated with each contig are sorted individually in increasing order. The first position of the sorted list of each contig contains the minimum score among all the scores. As the penalty term is very large, this matching score is the best score for placing this contig anywhere in the ORM.

We now sort the contigs based on the indices of the starting fragments corresponding to the best scores. As an example, assume that there are 5 contigs $C_1', C_2', \ldots, C_5'$ and consider their best scores. For each such score there is a starting fragment and an ending fragment. If the starting fragments of these contigs are o_5, o_{11}, o_3, o_{22}, and o_7, respectively, then the sorted order of the fragments will be o_3, o_5, o_7, o_{11}, and o_{22}. So the corresponding contigs with respect to its starting fragments will be $C_3', C_1', C_5', C_2', C_4'$. In general, let this sorted

order be $C_1, C_2, \ldots, C_r$. Followed by this, we attempt to place the contigs in the ORM in this order (using the mapping corresponding to the best score). Specifically, we first try to place C_1; Next we attempt to place C_2; and so on. When we try to place any contig C, we check whether the starting and/or ending fragments of C will overlap with any of the already placed contigs. If there is such an overlap, we discard C and move onto the next contig in the sorted list.

A detailed pseudocode is supplied in Algorithm 2.2. Let m be the number of fragments in the optical map, and r be the number of contigs. Intuitively the number of matching scores of each contig C_i is at most $O(m)$. Since the matching score is an integer, sorting matching scores of each contig C_i takes at most $O(m)$ time. So, the execution time of lines 2-7 in Algorithm 2.2 is $O(mr)$. Sorting contigs with respect to starting fragment takes $O(r)$ time (line 8). In the worst case lines 9-12 take $O(r^2)$ time. Since $r \ll m$, the run time of Algorithm 2.2 is $O(mr)$.

Greedy placement algorithm 2 – GPA2

GPA2 proceeds as follows: At first the matching scores associated with each contig are sorted individually in increasing order. Note that we consider m possible matchings for each contig and hence each contig has a list of m mappings and scores. Let the list of mappings (in sorted order of the matching scores) for contig C be L_C.

The number of matching sites for a contig mapping is defined to be the number of fragments in the contig that are matched with fragments in the ORM. For each contig, we know that there are m scores (with one score per starting fragment or mapping). Corresponding to each starting fragment (i.e., mapping) we can also compute the number of matching sites. Thus for every contig, we have a list of m numbers of matching sites. We identify for each

Algorithm 5.2: Greedy Placement Algorithm 1 (GPA1)

Input: Contigs with associated scores $CS[1..r][1..m]$
Output: Set of ordered contigs, C
begin

1. Create array of structure $struct[1..r]$
2. **for** (each contig, c_i){
3. Sort the matching score in increasing order
4. Place $struct[i].contig \leftarrow c_i$
5. Place $struct[i].starting\_position \leftarrow starting\_position$
6. Place $struct[i].ending\_position \leftarrow ending\_position$
7. }
8. Sort the array of $struct[1..r]$ with respect to $starting\_position$ in increasing order
9. **for** (each contig, c_i in $struct[1..r]$){
10. **if** (c_i is not overlapped with already placed contigs in C){
11. Place the contig c_i at the end of the list C
12. }
13. }
14. Return C

contig the mapping that has the largest number of matching sites. Let b_C be this number for contig C. We order the contigs based their b_C values in non-increasing order. Let the sorted list be $C'_1, C'_2, \ldots, C'_r$ based on their b_C values.

Place the contigs one-by-one based on the above sorted list starting from C'_1. For any contig C, mappings for this contig will be considered as per the list L_C. In other words, the first time when we try to place C, we will use the mapping found in $L_C[1]$. When we try to place C using this specific mapping, we check whether the starting and/or ending fragments of the contig will overlap with already placed contigs. If there is no overlap, we process the

Algorithm 5.3: Greedy Placement Algorithm 2 (GPA2)

Input: Contigs with associated scores $CS[1..r][1..m]$, Depth, d
Output: Set of ordered contigs, C
begin

```
1  Create array of structure struct[1..r]
2  for (each contig, c_i){
3      Sort the matched_sites in decreasing order
4      Place the sorted list in soretd_list variable
5      Place struct[i].contig ← c_i
6      Place struct[i].matched_list ← matched_sites
7     for (each matched sites, m_j in the matched_list){
8         Place struct[i].starting_position[j] ← starting_position[j]
9         Place struct[i].ending_position[j] ← ending_position[j]
10     }
11 }
12 Sort the array of struct[1..r] with respect to the greatest number of matched
   sites found in the first position of the matched_list
13 for (each contig, c_i in struct[1..r]){
14    for (j ← 1; j ≤ d; j ← j + 1){
15        if (c_i is not overlapped with already placed contigs in C){
16            Place the contig c_i at the end of the list C
17            Break
18        }
19    }
20 }
21 Sort the array C with respect to the starting position in increasing order
22 Return C
```

next contig. If there is an overlap while placing C (using the mapping in $L_C[1]$), we move to the next entry in L_C, i.e., $L_C[2]$. If successful, we process the next contig. If not, we move

Algorithm 5.4: Greedy Placement Algorithm 3 (GPA3)

Input: Contigs with associated scores $CS[1..r][1..m]$, Depth, d
Output: Set of ordered contigs, C
begin

```
1  Create array of structure struct[1..r]
2  for (each contig, c_i){
3      Sort the matching score in increasing order
4      Place the sorted list in soretd_list variable
5      Place struct[i].contig ← c_i
6      Place struct[i].score_list ← sorted_list
7     for (each score, s_j in the sorted_list){
8         Place struct[i].starting_position[j] ← starting_position[j]
9         Place struct[i].ending_position[j] ← ending_position[j]
10     }
11 }
12 Sort the array of struct[1..r] with respect to the least score found in the first
   position of the score_list
13 for (each contig, c_i in struct[1..r]){
14    for (j ← 1; j ≤ d; j ← j + 1){
15        if (c_i is not overlapped with already placed contigs in C){
16            Place the contig c_i at the end of the list C
17            Break
18        }
19    }
20 }
21 Sort the array C with respect to the starting position in increasing order
22 Return C
```

on to the next entry in L_C, and so on. We make repeated attempts to place C at most d times (where d is a user-specified parameter). If we are not successful in these d attempts,

we ignore C and proceed to process the next contig.

Additional details of the algorithm are supplied in Algorithm 2.3. Let m be the number of fragments in the optical map, and r be the number of contigs. The run time of lines 2-7 in Algorithm 2.3 is $O(mr)$ as discussed above. Sorting contigs with respect to the matched sites takes $O(r)$ time (line 8). Lines 13-20 take $O(rd)$ time. In line 21 sorting contigs with respect to starting fragment takes $O(r)$ time. Since $d \ll r \ll m$, the run time of Algorithm 2.3 is $O(mr)$.

Greedy placement algorithm 3 – GPA3

GPA3 takes as input the contigs and the ORM together with the output of Algorithm 2.1. If m is the number of ordered fragments in the ORM, then the number of scores (or mappings) associated with each contig will be m, as described in the previous section. The algorithm proceeds as follows: At first the matching scores associated with each contig are sorted individually in increasing order. The first position of the sorted list of each contig contains the minimum score (i.e., the best score) among all the scores.

We now sort the contigs based on their best scores. Let this sorted order be $C''_1, C''_2, \ldots, C''_r$. Followed by this, we place the contigs in the ORM in this order. Specifically, we first try to place C''_1; Next we try to place C''_2; and so on. Note that for any given contig and a corresponding score, we know the starting fragment as well as ending fragment (in the ORM). While trying to place any contig C, we check if there will be any overlaps with any of the contigs already placed. If so, we move on to the next entry in C's list and check if C can be placed based on the corresponding starting and ending fragments without any overlaps. We make a total of at most d such attempts to place C (where d is a user-defined parameter). If C cannot be placed successfully within these attempts, we drop C from further

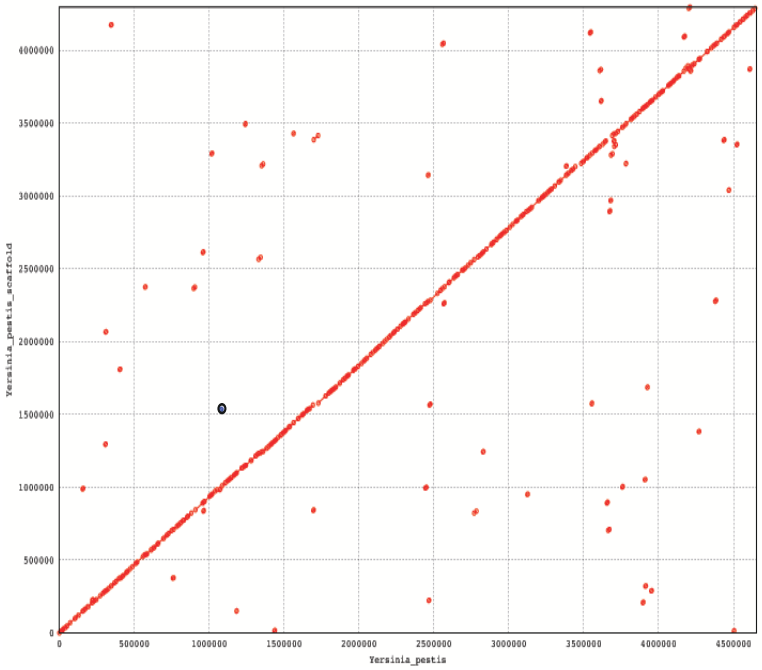


FIGURE 5.3.1: Aligning ordered contigs onto the *Yersinia pestis*

considerations and move on to the placement of the next contig.

A pseudocode of the algorithm can be found in Algorithm 5.4. Let m be the number of fragments in the optical map, and r be the number of contigs. The run time of lines 2-7 in Algorithm 5.4 is $O(mr)$ as discussed above. Sorting contigs with respect to the least matching score takes $O(r)$ time (line 8). Lines 13-20 take $O(rd)$ time. In line 21 sorting contigs with respect to starting fragment takes $O(r)$ time. Since $d \ll r \ll m$, the run time of Algorithm 5.4 is $O(mr)$.

5.3 Results and Discussions

To prove the effectiveness of our proposed algorithms we have done rigorous simulations on both real and synthetic datasets. The simulation results show that the algorithms are

TABLE 5.2.1: Results for *Yersinia pestis*

Contigs	Method	Missed Probability	% Resize	Conflicts	Wrong Placement	% Accuracy	Time (s)
50	GPA1	0.0	0	0	0	100.00	31.97
		0.1	5	0	0	100.00	29.45
		0.2	10	0	0	100.00	29.07
		0.3	20	27	1	44.00	25.99
	GPA2	0.0	0	0	0	100.00	34.10
		0.1	5	0	0	100.00	33.42
		0.2	10	0	0	100.00	30.83
		0.3	20	25	1	48.00	28.21
	GPA3	0.0	0	0	0	100.00	35.02
		0.1	5	0	0	100.00	32.41
		0.2	10	0	0	100.00	27.76
		0.3	20	12	2	72.00	27.24
100	GPA1	0.0	0	1	0	99.00	34.05
		0.1	5	4	0	96.00	31.23
		0.2	10	7	0	93.00	27.76
		0.3	20	45	6	49.00	25.92
	GPA2	0.0	0	1	0	99.00	31.44
		0.1	5	2	0	98.00	33.17
		0.2	10	4	2	94.00	26.18
		0.3	20	36	10	54.00	28.90
	GPA3	0.0	0	1	0	99.00	32.41
		0.1	5	0	0	100.00	30.10
		0.2	10	1	0	99.00	29.64
		0.3	20	27	6	67.00	29.04
200	GPA1	0.0	0	3	0	98.50	36.90
		0.1	5	8	0	96.00	33.28
		0.2	10	21	0	89.50	33.11
		0.3	20	69	4	63.5	29.61
	GPA2	0.0	0	3	0	98.50	33.56
		0.1	5	10	1	94.50	33.73
		0.2	10	19	3	89.50	34.29
		0.3	20	92	7	50.50	32.40
	GPA3	0.0	0	3	0	98.50	34.93
		0.1	5	5	0	97.50	35.96
		0.2	10	12	1	93.50	32.25
		0.3	20	52	5	71.5	32.16
400	GPA1	0.0	0	8	0	98.00	40.17
		0.1	5	20	2	94.50	35.00
		0.2	10	56	7	84.25	32.21
		0.3	20	120	15	66.25	30.47
	GPA2	0.0	0	8	0	98.00	34.77
		0.1	5	28	5	91.75	35.83
		0.2	10	47	25	82.00	33.15
		0.3	20	116	35	62.25	28.99
	GPA3	0.0	0	7	0	98.25	37.64
		0.1	5	18	0	95.50	31.70
		0.1	5	29	8	90.75	31.50
		0.3	20	162	21	76.75	31.70

TABLE 5.2.2: Results for *Yersinia enterocolitica*

Contigs	Method	Missed Probability	% Resize	Conflicts	Wrong Placement	% Accuracy	Time (s)
200	GPA1	0.0	0	0	0	100.00	43.37
		0.1	5	5	0	97.50	43.97
		0.2	10	18	0	91.00	38.92
		0.3	20	92	4	51.00	28.32
	GPA2	0.0	0	0	0	100.00	46.41
		0.1	5	3	0	98.50	45.47
		0.2	10	11	6	91.50	32.71
		0.3	20	84	10	53.00	32.29
	GPA3	0.0	0	0	0	100.00	41.10
		0.1	5	6	2	96.00	43.61
		0.2	10	11	0	94.50	40.41
		0.3	20	57	7	68.00	31.87
400	GPA1	0.0	0	9	0	97.75	46.67
		0.1	5	17	1	95.50	45.02
		0.2	10	45	1	88.50	37.00
		0.3	20	111	18	67.75	32.95
	GPA2	0.0	0	10	1	97.25	46.66
		0.1	5	26	4	92.50	49.04
		0.2	10	50	22	82.00	33.21
		0.3	20	135	26	59.75	31.90
	GPA3	0.0	0	9	0	97.75	43.89
		0.1	5	15	0	96.25	36.04
		0.2	10	29	5	91.50	33.53
		0.3	20	54	23	80.75	33.04

TABLE 5.2.3: Results for simulated data

Length	Contigs	Method	Placed	Observed Length	Difference	Edit Dist	Coverage	Time (s)
1×10^5 *bp*	7	GPA1	6	84689	15311	15483	84.69%	0.40
		GPA2	6	84689	15311	15483	84.69%	0.45
		GPA3	6	84689	15311	15483	84.69%	0.37
3×10^5 *bp*	34	GPA1	26	259619	40381	40923	86.54%	1.95
		GPA2	26	281905	18095	18917	93.97%	2.07
		GPA3	32	260662	39338	86220	86.89%	2.10
5×10^5 *bp*	52	GPA1	39	445727	54273	55210	89.15%	4.67
		GPA2	39	454376	45624	50185	90.88%	5.46
		GPA3	38	431582	68418	69285	86.32%	4.67
7×10^5 *bp*	53	GPA1	43	571908	128092	129160	81.70%	8.62
		GPA2	45	656139	45624	48593	93.73%	8.27
		GPA3	50	586588	113412	143189	83.80%	8.17

TABLE 5.2.4: Comparisons

Length	Method	Correctly Placed	Accuracy	Time (s)
5×10^5 *bp*	GPA1	49	98.00%	5.87
	GPA2	49	98.00%	4.65
	GPA3	49	98.00%	4.62
	Nagarajan et al. [89]	30	60.00%	1620
6×10^5 *bp*	GPA1	50	100.00%	8.52
	GPA2	50	100.00%	7.12
	GPA3	50	100.00%	7.12
	Nagarajan et al. [89]	32	64.00%	14400
7×10^5 *bp*	GPA1	49	98.00%	8.79
	GPA2	49	98.00%	8.19
	GPA3	49	98.00%	8.48
	Nagarajan et al. [89]	–	–	–
8×10^5 *bp*	GPA1	50	100.00%	11.77
	GPA2	50	100.00%	11.70
	GPA3	50	100.00%	10.64
	Nagarajan et al. [89]	–	–	–

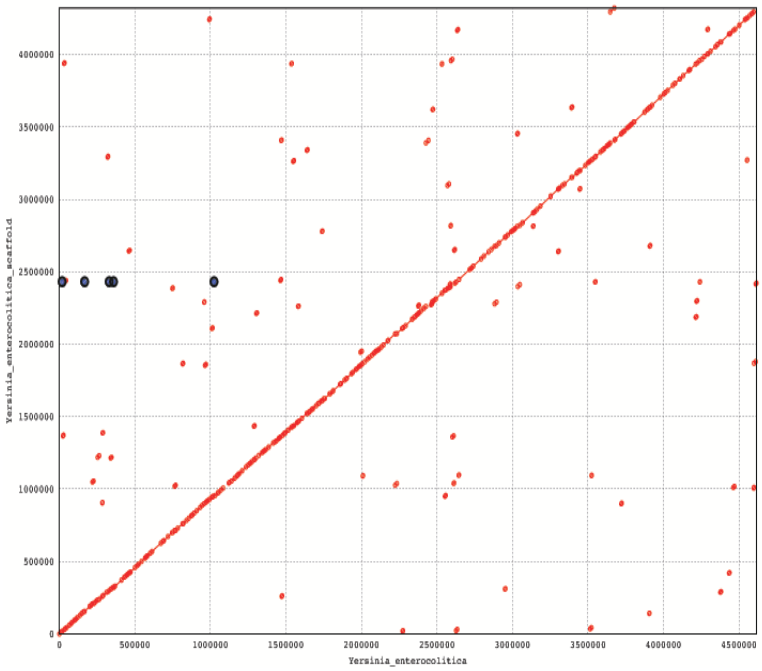


FIGURE 5.3.2: Aligning ordered contigs onto the *Yersinia enterocolitica*

indeed scalable and efficient. We have also compared our algorithm with one of the best known algorithms [89]. Our algorithm outperforms the aforementioned algorithm in terms of run time, by more than two orders of magnitude, and accuracy. The run time of the scaffolding algorithm of [89] is $O(m^2n^2r)$, where m is the number of fragments in the optical map, r is the number of contigs and n is the maximum number of fragments in any contig. In comparison, the run time of our algorithm is $O(mnr)$. In this section we present our experimental results. All the programs have been run on an Intel Core i5 2.3GHz machine with 4GB of RAM.

5.3.1 Real datasets

Real datasets are comprised of two strains of yersiniae bacteria, namely, *Yersinia pestis*, and *Yersinia enterocolitica*. The yersiniae are Gram-negative rods belonging to the family Enterobacteriaceae. They consist of 11 species of which three are pathogenic to humans. Those are *Yersinia pestis*, *Yersinia pseudotuberculosis*, and *Yersinia enterocolitica*. The genomic sequences of *Yersinia pestis* and *Yersinia enterocolitica* contain 4,653,728 bp and 4,615,899 bp, respectively. Each of the genomic sequences is randomly fragmented into a number of non-overlapping substrings/contigs of different lengths. We then permute the resulting contigs randomly to break the relative order existing among them. As we know the placement of the contigs when we generate them, we can easily detect whether our algorithms reconstruct the correct orderings from the randomly permuted contigs. To show the robustness of our proposed algorithms we introduce errors by discarding restriction sites with some probability. We also introduce errors by resizing, i.e., by increasing or decreasing the fragment sizes of the contigs.

We have generated 50, 100, 200, and 400 contigs from the genomic sequence of *Yersinia*

pestis and 200, and 400 contigs from *Yersinia enterocolitica*. Accuracy is defined as the fraction of the contigs placed correctly. If a contig cannot be placed, i.e, if the placement overlaps with other contigs, we call it a conflict. On the contrary when the placement of a contig is out of order (i.e. when the contig is misplaced) we call it wrong placement. From Table 5.2.1 and Table 5.2.2 it is evident that if there are no errors in the datasets, the accuracy found by applying the different methods is in the range: [97%, 100%]. The less the number of contigs, the more accurate the resulting placement of the contigs are. In this case, the algorithms are more resilient with errors. It is also the case that GPA3 is more robust against the errors introduced in the datasets.

To simulate practical scenarios, we have randomly generated reads of size 100 bp each from the two Yersinia strains. Contigs were created employing the String Graph Assembler (SGA) [99]. These contigs were then ordered using GPA2. After ordering we concatenated the ordered contigs to find the scaffold. As the sequences are very long, it is infeasible to calculate the edit distance between the original sequence and resulting scaffold. So, the genomic sequence and the corresponding scaffold are aligned using MUMmer [100]. The acronym "MUMmer" comes from "Maximal Unique Matches", or MUMs. It is based on the suffix tree data structure designed to find maximal exact matches of two input sequences. In Figure 1.7.1 we have aligned ordered contigs of *Yersinia pestis* onto the original sequence of *Yersinia pestis*. We have aligned ordered contigs of *Yersinia enterocolitica* onto the original sequence of *Yersinia enterocolitica*. The plots [Please see Figure 1.7.1 and Figure 1.7.2] represent the set of all MUMs between the two input sequences. Forward MUMs are plotted as red lines/dots while reverse MUMs are plotted as blue lines/dots (encircled). A line of dots with unit slope represents an undisturbed segment of conservation between the two sequences, while a line of dots with negative unit slope represents an inverted segment of conservation between the two sequences. As is evident, the alignments ordered contigs (i.e.

scaffold) are nicely placed onto the original sequences. The coverage of these two alignments is approximately 92% which proves the effectiveness of our algorithms.

5.3.2 Synthetic datasets

We have generated four genomic sequences of various sizes by choosing each character randomly from a uniform distribution. We generated reads of size 100 bp from each of the datasets such that the average coverage of the reads to a particular position of the sequence is around 5. Reads were generated by taking substrings of size 100 bp from randomly selected positions in the sequence. SGA [99] was used to generate contigs from the reads. The contigs were then ordered using our algorithms. ORM is created *in silico* by choosing a 4-bp long sequence acting as a restriction enzyme. The ordered fragment sizes of each contig are also created by employing the same procedure stated above. After getting the scaffold we calculate the edit distance between the original sequence and the resulting scaffold. It is intuitive that if the placement of the contigs in the scaffold is correct, then the following statement holds: $|Size(original\_sequence) - Size(constructed\_sequence)| \approx edit\_distance(original\_sequence,\ constructed\_sequence)$. Our simulation results show that this is indeed the case [Please see Table 5.2.3].

5.3.3 Comparison

We have compared our algorithms with one of the the best known algorithms existing in the literature [89]. The simulation results show that our proposed algorithms are superior in terms of both run time as well as accuracy. As the size of the sequence is increased more and more, our algorithms are faster and faster than [89]. We have compared our proposed algorithms with [89] by using synthetic datasets. The ground truth of exact ordering of

contigs is unknown in the case of real datasets as we do not know the placement of the contigs in prior. As optimal ordering is NP-hard, computationally it is impossible to find the correct placement when the number of contigs is large. So, to compare with [89] we have generated 4 artificial sequences of various sizes. 50 contigs were generated from each of the sequences. Contigs generation process is described in Section 5.3.1. Accuracy is calculated as the fraction of contigs placed correctly. As is evident from the simulation results, our algorithms are two orders of magnitude faster and our placements are also better [Please see Table 5.2.4] than [89]. In some cases we did not calculate the accuracy as it was taking an indefinite amount of time compared to our algorithms. '-' indicates this issue in Table 5.2.4.

5.4 Conclusions

Contig assembly is a very challenging task. In *de novo* assembly it is one of the most important steps to construct an entire genomic sequence from millions of reads produced by the sequencers. A series of algorithms has been proposed in this research work to order the contigs. ORM is used to calculate matching scores between the sequence and contigs. Contigs are then placed so that the overall cumulative matching scores are minimized. We have performed rigorous simulations on both real and synthetic datasets. The results show that our algorithms are efficient in terms of both run time and accuracy.

Part III

Genotype-phenotype Correlation

Chapter 1

Genome-wide Association Study

Advances made in sequencing technology have resulted in the sequencing of thousands of genomes. Novel analysis tools are needed to process these data and extract useful information. Such tools could aid in personalized medicine. As an example, we could identify the causes for a disease by comparing the genomes of people who have the disease and those who do not have this disease. Given that human variability happens due to single nucleotide polymorphisms (SNPs), we could focus our attention on these SNPs. Investigations that try to understand human variability using SNPs fall under genome-wide association study (GWAS). A crucial step in GWAS is the identification of the correlation between genotypes (SNPs) and phenotypes (i.e., characteristics such as the presence of a disease). This step can be modeled as the k-locus problem (where k is any integer). A number of algorithms have been proposed in the literature for this problem. In this research work we present an algorithm for solving the 2-locus problem that is up to two orders of magnitude faster than the previous best known algorithms. The case of $k > 2$ has not been studied in the literature. For the first time, in this research work we present an efficient algorithm for solving the 3-

locus problem that is several orders of magnitude faster than the brute force algorithm. The k-locus problem can be thought of as a special case of the closest pair problem (CPP). CPP is one of the well studied and fundamental problems in computing. Given a set of points in a metric space, the problem is to identify the pair of closest points. There are numerous applications where this problem finds a place. Examples include computational biology, computational finance, share market analysis, weather prediction, entomology, electro cardiograph, N-body simulations, molecular simulations, etc. As a result, any improvements made in solving CPP will have immediate implications for the solution of numerous problems in these domains. A naive deterministic algorithm can solve CPP in quadratic time. Quadratic time may be too much given that we live in an era of big data. Speeding up data processing algorithms is thus much more essential now than ever before. In this research work we present algorithms for CPP that improve (in theory and/or practice) the best-known algorithms reported in the literature for CPP.

1.1 Introduction

There are a number of ways in which any two human genomes can differ. Variations are largely due to the single nucleotide polymorphisms (SNPs in short) as well as deletions, insertions and copy number variations [107]. Any of these variations may result in alterations in an individual's traits, or phenotypes. A phenotype of interest can be anything from a disease risk to physical properties such as height. In genetic epidemiology, a genome-wide association study (GWA study, or GWAS), also known as whole genome association study (WGA study, or WGAS), is an examination of many common genetic variants in different individuals to observe if any variant is associated with a trait. GWAS plays a major role in

personalized medicine.

A lot of effort has been spent to identify mappings between phenotypical traits and genomic data. Due to the advent of next generation high throughput sequencing technologies, nowadays it is possible to study the genomic structure of individuals in detail. In GWAS, two different problems have been focused on. In single locus association study, researchers try to find out the association between phenotypical traits and individual SNPs. In two locus association study, the goal is to figure out the association between pairs of SNPs and phenotypical traits. A major task in this study is that of identifying the most correlated pair of SNPs. Two locus associations are also known as gene-gene interactions. Such interactions are believed to be major factors responsible for many complex phenotypical traits [117, 102, 106, 108, 109, 128]. It is a proven fact that such gene $\times$ gene interactions are the major actors to express many complex traits such as various human diseases. A generalization of this problem is that of identifying the k most important loci responsible for a specific phenotype. This is known as the k-locus problem.

Given that the number of SNPs found in humans is 10^5 to 10^7, a brute force way of scanning through every possible pair of SNPs to identify the most correlated pair is not feasible in practice. A number of algorithms for the two locus problem can be found in the literature. For instance, genetic algorithms are used in [118] and [127]. The algorithms proposed in [130] and [131] take $O(n^2m + nm^2)$ time, where n is the number of SNPs and m is the number of subjects. This algorithm is called FastANOVA. An algorithm with an expected run time of $O(mn^{1+\epsilon}\log^2 n)$, where $0 < \epsilon < 1$ is a constant, has been presented in [101]. This algorithm exploits an algorithm known for the Light Bulb Problem [119] and Locality Sensitive Hashing (LSH) [104]. In this research work we present an algorithm for the two locus problem whose expected run time is $O(mn^{1+\epsilon}\log n)$ (where $0 < \epsilon < 1$ is a constant). Our algorithm is asymptotically better than that of [101] by a logarithmic factor.

Moreover, we have employed a number of innovations at the coding level and hence the improvement is indeed by more than a logarithmic factor. Our algorithm is up to two orders of magnitude faster than prior algorithms (specifically FastANOVA [131] and the algorithm of [101]) on various benchmark datasets.

The two locus problem can be thought of as a special case of the closest pair problem (CPP). CPP has a rich history and has been extensively studied. On an input set of n points, the problem is to identify the closest pair of points. A straight forward algorithm for CPP takes quadratic (in n) time. Most of the algorithms proposed in the literature are concerned with the Euclidean space. In his seminal paper, Rabin proposed a randomized algorithm with an expected run time of $O(n)$ [121] (where the expectation is in the space of all possible outcomes of coin flips made in the algorithm). Rabin's algorithm used the floor function as a basic operation. In 1979, Fortune and Hopcroft presented a deterministic algorithm with a run time of $O(n \log\log n)$ assuming that the floor operation takes $O(1)$ time [110]. Both of these algorithms assume a constant-dimensional space (and the run times have an exponential dependency on the dimension). Other clasical algorithms include [120, 112]. Yao has proven a lower bound of $\Omega(n \log n)$ on the algebraic decision tree model (for any dimension) [129]. This lower bound holds under the assumption that the floor function is not allowed. One of the major issues with the above algorithms is the fact that their run times are exponentially dependent on the dimension.

Time series motif mining (TSMM) is a crucial problem that can be thought of as CPP in a large dimensional space. In one version of the TSMM problem, we are given a sequence S of real numbers and an integer ℓ. The goal is to identify two subsequences of S of length ℓ each that are the most similar to each other (from among all pairs of subsequences of length ℓ each). These most similar subsequences are referred to as *time series motifs*. Let C be a collection of all the ℓ-mers of S. (An ℓ-mer is nothing but a contiguous subsequence of S of

length ℓ). Clearly, the ℓ-mers in C can be thought of as points in $\Re^\ell$. As a result, the TSMM problem is the same as CPP in $\Re^\ell$. Any of the above mentioned algorithms can thus be used to solve the TSMM problem. A typical value for ℓ of practical interest is several hundreds (or more). For these values of ℓ, the above algorithms ([121],[110],[120],[112]) will take an unacceptable amount of time (because of the exponential dependence on the dimension). Designing an efficient practical and exact algorithm for the TSMM problem remains an ongoing challenge.

Mueen, et al. have presented an elegant exact algorithm called MK for TSMM [116]. MK improves the performance of the brute-force algorithm with a novel application of the triangular inequality. MK is currently the best-performing algorithm in practice for TSMM. A number of probabilistic as well as approximate algorithms are also known for solving this problem (see e.g., [103, 105, 111, 113, 114, 124, 125]). For instance, the algorithm of [105] exploits algorithms proposed for finding (ℓ, d)-motifs from biological data. The idea here is to partition the time series data into frames of certain width. Followed by this, the mean value in each frame is computed. This mean is quantized into four intervals and as a result, the original time series data is converted into a string of characters from an alphabet of size 4. Finally, any (ℓ, d)-motif finding algorithm is applied on the transformed string to identify the time series motifs. In this research work we present efficient algorithms for CPP. Our algorithms improve the results reported in several papers including [116], [105], and [101].

Note that all the experiments were done on an Intel Haswell compute node with 48 GB of RAM. The operating system running was Red Hat Enterprise Linux Server release 5.7 (Tikanga).

1.2 Notations and Definitions

Let $T = a_1, a_2, \ldots, a_n$ be a sequence of real numbers (or characters from a finite alphabet). An ℓ-mer of T is nothing but a subsequence of T of ℓ contiguous elements of T. The ℓ-mers of T are $T_i = a_i, a_{i+1}, \ldots, a_{i+\ell-1}$, for $1 \leq i \leq (n - \ell + 1)$.

If the elements of T are real numbers, then the Euclidean distance between T_i and T_j, denoted as $d(T_i, T_j)$, is $\sqrt{\sum_{k=0}^{\ell-1}(a_{i+k} - a_{j+k})^2}$.

If the elements of T are characters from an alphabet Σ, then the Hamming distance between T_i and T_j, denoted as $d(T_i, T_j)$, is $\sum_{k=0}^{\ell-1} \delta(a_{i+k}, a_{j+k})$ where $\delta(a, b) = 1$ if $a \neq b$ and $\delta(a, b) = 0$ if $a = b$ (for any $a, b \in \Sigma$). A sequence of characters can be thought of as a string of characters, since we can obtain a string from the sequence by concatenating the characters. Thus we'll use the terms 'a sequence of characters' and 'a string of characters' interchangeably.

Let $A = a_1, a_2, \ldots, a_n$ and $B = b_1, b_2, \ldots, b_n$ be two sequences of characters. Also, let the Hamming distance between A and B be d. Then, by the *number of matches* between A and B we mean $n - d$. Also, the *correlation* between A and B is defined to be $\frac{n-d}{n}$.

1.3 Some Preliminaries

In this section we provide some preliminary ideas and techniques that will be useful for solving the two locus problem. Specifically, in this section we consider the following problems: finding the most similar pair of character strings, and finding the least similar pair of character strings, where the similarity is in terms of the Hamming distance between strings.

1.3.1 Finding the most correlated pair of strings

For this problem we are given n Boolean vectors $\hat{b}_1, \hat{b}_2, \ldots, \hat{b}_n$ each of length t. The problem is to find the pair of vectors that are the most similar (i.e., the Hamming distance between them is the smallest). Note that, given two vectors, we can find the Hamming distance between them in $O(t)$ time. A straight forward algorithm to identify the most correlated pair of vectors takes $O(n^2 t)$ time. This algorithm computes the Hamming distance between every pair of vectors. We can achieve a better run time using randomization. We say that the correlation between a pair of strings is p if the Hamming distance between them is $t(1-p)$. Let p_1 be the correlation between the most correlated pair of strings and p_2 be the correlation between the second most correlated pair of strings.

Consider a matrix M of size $n \times t$, such that the ith row of M is $\hat{b}_i$, for $1 \leq i \leq n$. The idea of our algorithm is to iteratively collect pairs of strings that are candidates to be the most correlated. Once we collect enough pairs, we compute the distance between each pair in this collection and output the closest. In each iteration we pick q columns randomly. For any vector (or string), the values in these columns can be concatenated to get a q-bit integer. We hash the vectors based this integer value. Subsequently, we generate pairs as follows: Consider any bucket in the hash table. If there are m vectors in this bucket, then each pair of vectors in this bucket is added as a candidate to a list C. There are $O\left(n^{\frac{\log p_1}{\log p_2}} \log n\right)$ iterations in the algorithm.

We can show that after $O\left(n^{\frac{\log p_1}{\log p_2}} \log n\right)$ iterations, C will have the most correlated pair of bulbs with a high probability (i.e., with a probability of $1 - n^{-\Omega(1)}$). This algorithm is similar to the algorithm given in [119] but faster (by more than a logarithmic factor). We can generalize the above algorithm to arbitrary alphabets and get the following

Theorem 1.3.1. Let M be a matrix of size $n \times t$. Each entry in this matrix is an element

from some set Σ of cardinality σ. We can find the most correlated pair of rows of M in an expected $O\left(n^{1+\frac{\log p_1}{\log p_2}} \log n\right)$ time where p_1 is the correlation between the most correlated pair of rows, p_2 is the correlation between the second most correlated pair of rows. (Here correlation is based on Hamming distance. For example, p_1 is the largest fraction of columns in which any two rows agree).

Proof: Probability that the most correlated pair of strings falls into the same bucket is p_1^q. (Note that each row is a string). Probability that this pair does not fall into the same bucket in a given iteration is $(1 - p_1^q)$. Thus, the probability that this pair does not fall into the same bucket in z successive iterations is $(1 - p_1^q)^z$. This probability is $\leq \exp(-zp_1^q)$ using the fact that $(1 - x)^{1/x} \leq 1/e$ for any $0 < x < 1$. In turn, this probability will be $\leq n^{-\alpha}$ if $z \geq \frac{\alpha \log n}{p_1^q}$.

To ensure that the run time is as small as possible, we want to ensure that the size of C is not too large. Since we spend a linear time in each iteration of the algorithm, an optimal strategy will be to make sure that the number of candidate pairs generated in each iteration is also $O(n)$. If p_2 is the second largest correlation, the probability that any pair other than the largest correlated falls into the same bucket in any iteration is $\leq p_2^q$. If this probability is $\leq \frac{1}{n}$, then the expected number pairs generated in any iteration will be $\leq n$. This happens if $q = \frac{\log n}{\log(1/p_2)}$. For this value of q, the value of z becomes $\alpha \log n \; n^{\log p_1 / \log p_2}$.

Given that the expected time we spend in hashing in each iteration and the time for generating the pairs is $O(n)$, it follows that the expected run time of the algorithm is $O\left(n^{1+\frac{\log p_1}{\log p_2}} \log n\right)$. $\square$

Note: The run time of MCP is better than that of the light bulb algorithm presented in [119] by more than a logarithmic factor.

Let the above general algorithm be called MCP.

The case of random data. To get an idea of how large $\frac{\log p_1}{\log p_2}$ could get consider the case where all the strings are random (i.e., each character of any string is randomly chosen to be 0 or 1 with equal probability). Consider a collection of n random strings of length t each. Let A and B be any two of these vectors. The expected correlation between A and B is $1/2$, i.e., the expected number of positions in which they match is $t/2$. We can use Chernoff bounds to get high probability estimates on p_1 and p_2.

Chernoff bounds: If X is the sum of t independent Bernoulli trials with a probability p of success each, then the following are true, for any $0 < \delta < 1$ [115]:

$$Prob.[X < ((1-\delta)\mu] < \exp(-\mu\delta^2/2).$$

Here $\mu = tp$. Also, if $p \geq 1/2$, then, for any $x > 0$,

$$Prob.[X > tp + x] < \exp\left(\frac{-x^2}{2tp(1-p)}\right).$$

Using the above bounds, it follows that the probability that the correlation between A and B is more than $\frac{t}{2} + x$ is $\leq \exp(-2x^2/t)$. Thus the probability that there exists at least one pair whose correlation is $> \frac{t}{2} + x$ is $\leq n^2 \exp(-2x^2/t)$. As a result, we infer that the correlation between any pair is $\leq \frac{1}{2} + \sqrt{\frac{2\ln n - \ln u}{t}}$ with probability $\geq (1-u)$, for any $0 < u < 1$. Similarly, the probability that none of the pairs has a correlation less than $\frac{1}{2} - \sqrt{\frac{2\ln n - \ln u}{t}}$ is $\geq (1-u)$. Thus a high probability upper bound on p_1 is $\hat{p}_1 = \frac{1}{2} + \sqrt{\frac{2\ln n - \ln u}{t}}$ and a high probability lower bound on p_2 is $\check{p}_2 = \frac{1}{2} - \sqrt{\frac{2\ln n - lnu}{t}}$.

Table 3.2.1 displays these estimates for different values of n. For this table, the value of t has been chosen to be 200 and that of u is 0.01. It also shows the corresponding values of $\frac{\log \hat{p}_1}{\log \check{p}_2}$. From this table we see that for a fixed value of t when the value of n increases, the

value of $\frac{\log \hat{p}_1}{\log \check{p}_2}$ decreases.

TABLE 1.3.1: Correlation estimates for the case of random data.

n	$\hat{p}_1$	$\check{p}_2$	$\frac{\log \hat{p}_1}{\log \check{p}_2}$
10,000	0.8393	0.1607	0.0958
50,000	0.8622	0.1378	0.0748
100,000	0.8717	0.1283	0.0669

1.3.2 An experimental comparison of MK and MCP

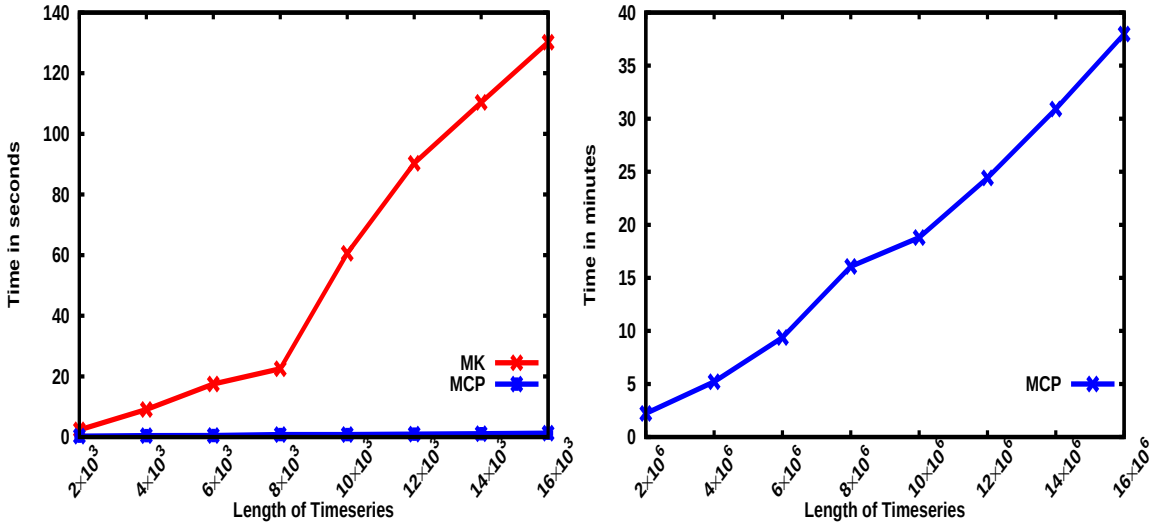
The algorithm of [105] for approximate TSMM partitions the input time series data T based on a window of size w (for an appropriate value of w), computes the mean of every window, and discretizes the mean into four possible values. As a result, the time series data is transformed into a string T' of characters form the alphabet $\{1, 2, 3, 4\}$. It then uses any (ℓ, d)-motif finding algorithm to find the motifs in T'. However, all the exact algorithms for finding (ℓ, d)-motifs take time that is exponential on ℓ and d. Note that the last step of finding (ℓ, d) motifs can be replaced with a problem of finding time series motifs in T' which is nothing but CPP in the domain of strings of characters, the motif length being ℓ.

TABLE 1.3.2: # of pairs and runtime comparisons in Hamming space on binary strings. CPU times are given in seconds.

		MK		MCP		Gain	Speed up
Dataset	Size	Processed Pairs	CPU Time	Processed Pairs	CPU Time	Processed Pairs	Time
D1	2×10^3	976,520	2.34	1,515	0.27	644.57	8.82
D2	4×10^3	3,437,171	9.09	6,099	0.44	563.56	20.81
D3	6×10^3	5,958,494	17.50	13,889	0.51	429.01	33.99
D4	8×10^3	7,513,580	22.51	24,308	0.80	309.10	28.29
D5	10×10^3	20,788,832	60.59	38,076	0.81	545.98	74.69
D6	12×10^3	29,722,565	90.28	54,707	0.95	543.30	94.87
D7	14×10^3	35,408,500	110.38	75,058	1.09	471.75	101.08
D8	16×10^3	42,735,676	130.22	97,934	1.26	436.37	103.06
D9	18×10^3	48,137,187	157.73	124,126	1.37	387.81	114.90
D10	20×10^3	119,362,627	381.11	152,837	1.51	780.98	251.86

TABLE 1.3.3: Number of pairs and runtime of MCP in Hamming space on binary strings. CPU times are given in minutes.

		Brute-force	MCP		Gain
Dataset	Size	Processed Pairs	Processed Pairs	CPU Time	Processed Pairs
D1	2×10^6	2×10^{12}	3.89×10^8	2.22	5,140.10
D2	4×10^6	8×10^{12}	15.56×10^8	5.21	5,139.87
D3	6×10^6	18×10^{12}	35.02×10^8	9.38	5,140.06
D4	8×10^6	32×10^{12}	74.46×10^8	16.07	4,297.45
D5	10×10^6	50×10^{12}	97.27×10^8	18.80	5,140.10
D6	12×10^6	72×10^{12}	140.08×10^8	24.42	5,140.02
D7	14×10^6	98×10^{12}	190.66×10^8	30.91	5,140.03
D8	16×10^6	128×10^{12}	249.02×10^8	37.98	5,140.09



(A) CPU times consumed by MK and MCP methods

(B) CPU times consumed by MCP method

FIGURE 1.3.1: Performance evaluations between MK and MCP methods.

One could employ MK to solve the problem of finding the most correlated pair of strings. The only difference is that we have to replace Euclidean distance with Hamming distance. We have implemented this algorithm. It turns out that MK does not perform well for the case of binary strings. To be fair, the authors of MK have not tested MK for this case. We have compared MK with MCP and the results are shown in Table 3.2.2. As this Table reveals, MCP has a much better performance than MK. Please, see Figure 1.3.1 for visual details. It is also clear that if we employ MCP in place of (l, d)-motif finding algorithms, the performance of the approximate TSMM algorithm given in [105] will improve significantly.

When n is very large, we *inject* pairs with known correlations. As an example, consider the problem of finding the largest correlated pair of rows in a $m \times n$ matrix M. Say we generate each row by picking each element to be either 0 or 1 with equal probability. For any two rows, clearly, the expected correlation is $\frac{1}{2}$. We can perform a probabilistic analysis to get a high probability bound on the largest correlation between any two rows (see Table 3.3.1).

For example, for $n = 10,000$, we generated several random data sets and computed the largest correlation in each and calculated an average. Let p be this value. To inject a pair with a correlation of p' where p' is $> p$ we generate a row a with all ones and another row b with $p'm$ ones and $m(1-p')$ zeros. We then replace (any) two rows of M with a and b. Clearly, the correlation between a and b is p'. The expected correlation between a and any other row of M (other than b) is $\frac{1}{2}$. Similarly, the expected correlation between b any other row of M (other than a) is $\frac{1}{2}$. Thus the pair (a, b) is likely to be the winner with high probability. We have picked a value for p' that is only slightly larger than p so as to get an accurate estimate on the run time. We have used a similar technique to inject pairs to find most correlated pairs as well. We stop our algorithm MCP after a fixed number of runs i.e., 50. We are always able to find the most correlated pair we injected within the fixed number of runs. In Table 3.3.1 we show the results for our algorithm. As evident from Table 3.3.1 MCP finds the injected pair within a very short period of time. As an example consider the dataset D8. It consists of 16 million elements, the motif length being 1,024. It processed 249.02×10^8 pairs only in 37 minutes to find the most correlated pair.

1.3.3 Identification of the least correlated pair of strings

The MCP algorithm identifies the closest pair of strings, from out of n given binary strings. An interesting question is if we can use the same algorithm to identify the **furthest** pair of strings. This problem has relevance in many problems including the two locus problem in GWAS. The authors of [101] present an elegant adaptation of the light bulb algorithm of [119] to solve this problem when the strings are binary. In this section we show how to adapt MCP to solve this problem. Our experimental comparison shows that our algorithm has a significantly better run time than that of [101].

Some notations

Let $m(x, y)$ stand for the number of matches between two strings (of equal length) x and y. For instance, if $x = 10010$ and $y = 00111$, then $m(x, y) = 2$ (since they match in positions 2 and 4). Let $X = x_1, x_2, \ldots, x_q$ and $Y = y_1, y_2, \ldots, y_q$ be two sequences of strings (each string having the same length). We define $M(X, Y)$ to be $\sum_{i=1}^{q} m(x_i, y_i)$.

Consider the sequences $A_i = a_1^i, a_2^i, \ldots, a_k^i$, for $1 \leq i \leq n$, where each a_j^i is 0, or 1 (for $1 \leq j \leq k$). Note that each A_i is a sequence of strings where each string is of length 1. Let $M(A_i, A_j) = u_{ij}$.

Each A_i can be thought of as a binary string. In the application of GWAS, we can let A_i correspond to the SNP i, for $1 \leq i \leq n$. Specifically, a_j^i is the value of the ith SNP in subject j, for $1 \leq j \leq k$. If we are interested in finding the two most correlated SNPs, then we can use MCP to identify this pair. On the other hand, if our goal is to identify the least correlated pair, then, it is not clear how to do this using MCP. To solve the two locus GWAS problem, we have to identify not only the most correlated pair of strings but also the least correlated pair.

Finding the least correlated pair

The authors of [101] present an elegant solution for this problem. The idea is to construct a matrix D of size $k \times 2n$ where each column of D corresponds to either a light bulb or its 'complement', (We can think of each light bulb as a string). Specifically, the first n columns correspond to the bulbs and the next n columns correspond to the complements of the bulbs. In other words, $D[j, i] = a_j^i$, for $1 \leq i \leq n, 1 \leq j \leq k$ and $D[j, i] = \bar{a}_j^i$ for $1 \leq j \leq k, (n + 1) \leq i \leq 2n$. Here, if x is any bit, then, $\bar{x}$ denotes its complement. Let $D_1 = \{q : 1 \leq q \leq n\}$ and $D_2 = \{q : (n + 1) \leq q \leq 2n\}$. The algorithm of [101] for finding

the least correlated pair works as follows. Consider all the pairs of columns (a, b) such that $a \in D_1$ and $b \in D_2$. From out of these pairs, identify the pair (a', b') of columns with the maximum number of matches. If $a' = i$ and $b' = n + j$, then (i, j) is the least correlated pair of bulbs. Finding such a pair (a', b') can be done using the light bulb algorithm of [119]. The correctness of this algorithm follows from the fact that if the two bulbs i and j have the least number of matches, then, column i and the complement of column j will have the most number of matches. In this approach we can replace the light bulb algorithm with MCP to drastically improve the run time (by more than a logarithmic factor).

Finding the least correlated pair - the case of zeros, ones, and twos

It is not clear how to extend the above idea when the alphabet under concern has three (or more) possible elements. The authors of [101] reduce such general cases to the case of zeros and ones using locality sensitive hashing (LSH).The measure of correlation used by [101] is different from what we use in this research work. We define the correlation between two strings A_i and A_j as $p_{ij} = \frac{M(A_i, A_j)}{k}$. In contrast, [101] use Pearson's correlation coefficient.

In this section we present an elegant algorithm for the problem of identifying the least correlated pair of strings without employing LSH. The idea of [101] is to map input strings into Boolean vectors. If i and j are any two strings, then the sequences A_i and A_j are mapped to Boolean vectors A'_i and B'_i by LSH such that the distance between A_i and A_j will be nearly the same as the distance between A'_i and A'_j with some probability. The larger the length of A'_i is, the better will be the accuracy of LSH in preserving distances.

Our algorithm also maps each A_i into a Boolean vector A'_i deterministically such that $|A'_i| = 3|A_i|$, for $1 \leq i \leq n$.

Consider an alphabet Σ of size 3 with $\Sigma = \{001, 010, 100\}$. Clearly, $m(x, y) = 3$ if $x = y$

and $m(x,y) = 1$ if $x \neq y$ for any $x, y \in \Sigma$. Also, $m(x,\bar{y}) = 0$ if $x = y$ and $m(x,\bar{y}) = 2$ if $x \neq y$. Here $\bar{y}$ stands for the string obtained from y by complementing each bit. For example, if $y = 010$ then $\bar{y} = 101$.

Consider the sequences $A_i = a_1^i, a_2^i, \ldots, a_k^i$, for $1 \leq i \leq n$, where each a_j^i is $0, 1$, or 2 (for $1 \leq j \leq k$). Note that each A_i is a sequence of strings where each string is of length 1. Let $M(A_i, A_j) = u_{ij}$. Assume now that we encode each a_j^i as follows (for $1 \leq i \leq n$ and $1 \leq j \leq k$): $0 \rightarrow 001; 1 \rightarrow 010$: and $2 \rightarrow 100$. Let the encoded version of A_i be denoted as A_i' for $1 \leq i \leq n$. Note that $|A_i'| = 3k$, for any $1 \leq i \leq n$.

It is easy to see that $M(A_i', A_j') = 3u_{ij} + (k - u_{ij}) = k + 2u_{ij}$, for any i and j $(1 \leq i, j \leq n)$. For any $A_i' = a_1'^i, a_2'^i, \ldots, a_{3k}'^i$, let $\bar{A}_i' = \bar{a}_1'^i, \bar{a}_2'^i, \ldots, \bar{a}_{3k}'^i$, for $1 \leq i \leq n$. Clearly, $M(A_i', \bar{A}_j') = 2(k - u_{ij})$, for any $1 \leq i, j \leq n$.

The following statement is true: If, from out of all the pairs of strings, (i, j) has the largest correlation, i.e., u_{ij} is the largest, then from out of all the Boolean vectors generated, A_i' and A_j' will have the largest correlation. Also, if u_{ij} is the smallest, then, A_i' and $\bar{A}_j'$ will have the least correlation (from out of the pairs $(A_i', \bar{A}_j'), i \neq j, 1 \leq i, j \leq n$).

We can now form a matrix D of size $3k \times 2n$ where the first n columns correspond to (transformed) strings and the next n columns correspond to complements of (transformed) strings. Let $D_1 = \{q : 1 \leq q \leq n\}$ and $D_2 = \{q : (n+1) \leq q \leq 2n\}$. Consider all the pairs of columns (a, b) such that $a \in D_1$ and $b \in D_2$. From out of these pairs, identify the pair (a', b') of columns with the maximum number of matches. If $a' = i$ and $b' = n + j$, then (i, j) is the least correlated pair of strings. Finding such a pair (a', b') can be done using the algorithm MCP.

Run time analysis

Theorem 1.3.2. Given n strings, we can find the closest pair of strings in an expected time of $O\left(n^{1+\frac{\log p_1}{\log p_2}} \log n\right)$, where p_1 and p_2 are the largest and the second largest correlation values, respectively. Also, we can find the least correlated pair of strings in an expected time of $O\left(n^{1+\frac{\log((2/3)(1-c_1))}{\log((2/3)(1-c_2))}} \log n\right)$, where c_1 and c_2 are the smallest and the next smallest correlation values, respectively.

Proof: When we transform input strings to binary sequences, the ordering of pairs is preserved in terms of correlations as we have shown before. Let p_1 be the correlation of the largest correlated pair and p_2 be the correlation of the second largest correlated pair. How do these values change in the transformed domain? If p_1' and p_2' are the transformed values of these correlations, respectively, it can be seen that $p_1' = \frac{1}{3} + \frac{2}{3}p_1$ and $p_2' = \frac{1}{3} + \frac{2}{3}p_2$.

If c_1 and c_2 are the correlations of the smallest and the second smallest correlated pairs, respectively, and if c_1' and c_2' are the transformed values of these, respectively, then we can see that: $c_1' = \frac{2}{3}(1 - c_1)$ and $c_2' = \frac{2}{3}(1 - c_2)$. To find the largest correlated pair, we can use MCP (Theorem 1.3.1). We use the mapping only to find the least correlated pair. □

The case of a general alphabet

We have thus far considered the case where the alphabet is $\{0, 1, 2\}$. We can extend the mapping to a general alphabet and get the following theorem.

Theorem 1.3.3. Given n strings, we can find the largest correlated pair of strings in an expected time of
$O\left(n^{1+\frac{\log p_1}{\log p_2}} \log^2 n\right)$, where p_1 and p_2 are the largest and the second largest correlation values, respectively. Also, we can find the least correlated pair of strings in an expected time of

$O\left(n^{1+\frac{\log((2/\sigma)(1-c_1))}{\log((2/\sigma)(1-c_2))}}\log^2 n\right)$, where c_1 and c_2 are the smallest and the next smallest correlation values, respectively, and there are σ characters in the alphabet.

Proof: Consider sequences from the alphabet $\{0, 1, \ldots, \sigma - 1\}$. In this case we map each element of this alphabet to a binary string of length σ where there is only one 1. Specifically, we use the following mapping: $0 \rightarrow 00\cdots001$; $1 \rightarrow 00\cdots010$; etc. As before, we don't need any mapping if our goal is to find the largest correlated pair. The mapping is used only to find the least correlated pair. □

We can improve the above theorem by employing a random mapping as follows: We will use a binary string of length σ to encode each symbol in the alphabet. The encoding for each symbol is obtained by (uniformly) randomly choosing each bit in the string (of length σ). Let x and y be any two symbols in the alphabet (with $x \neq y$) and let e_x and e_y be their encodings, respectively. Then, clearly, the expected value of $m(e_x, e_y)$ is $\frac{\sigma}{2}$. Also, the expected value of $m(e_x, \bar{e}_y)$ is $\frac{\sigma}{2}$. If c is the correlation between a pair of strings and if c' is the transformed value, then, it follows that the expected value of c' is $\frac{1}{2}(1 - c)$. An application of the Chernoff bounds will readily imply that the value of c' will indeed be very close to this expected value with a probability of $1 - \sigma^{-\Omega(1)}$. Therefore, we get:

Theorem 1.3.4. Given n strings, we can find the least correlated pair of strings in an expected time of

$O\left(n^{1+\frac{\log((1/2)(1-c_1))}{\log((1/2)(1-c_2))}}\log n\right)$, where c_1 and c_2 are the smallest and the next smallest correlation values, respectively, and there are σ characters in the alphabet. □

1.3.4 Finding the most correlated triple of strings

For this problem we are given n Boolean vectors $\hat{b}_1, \hat{b}_2, \ldots, \hat{b}_n$ each of length t. The problem is to find the triple of vectors that are the most similar (i.e., the Hamming distance among them is the smallest). Note that, given three vectors, we can find the Hamming distance between them in $O(t)$ time. A straight forward algorithm to identify the most correlated triple of vectors takes $O(n^3 t)$ time. This algorithm computes the Hamming distance between every triple of vectors. We can achieve a better run time using randomization. We say that the correlation among a triple of strings is p if the Hamming distance among them is $t(1-p)$. Let p_1 be the correlation among the most correlated triple of strings and p_2 be the correlation among the second most correlated triple of strings.

Consider a matrix M of size $n \times t$, such that the ith row of M is $\hat{b}_i$, for $1 \leq i \leq n$. The idea of our algorithm is to iteratively collect triples of strings that are candidates to be the most correlated. Once we collect enough triples, we compute the distance among each triple in this collection and output the top x closest triples of strings. In each iteration we pick q columns randomly. For any vector (or string), the values in these columns can be concatenated to get a q-bit integer. We hash the vectors based on this integer value. Subsequently, we generate triples as follows: Consider any bucket in the hash table. If there are m vectors in this bucket, then each triple of vectors in this bucket is added as a candidate to a list C. There are $O\left(n^{2\frac{\log p_1}{\log p_2}} \log n\right)$ iterations in the algorithm.

Theorem 1.3.5. Let M be a matrix of size $n \times t$. Each entry in this matrix is an element from some set Σ of cardinality σ. We can find the most correlated triple of rows of M in an expected $O\left(n^{1+\frac{2\log p_1}{\log p_2}} \log n\right)$ time where p_1 is the correlation of the most correlated triple of rows, p_2 is the correlation of the second most correlated triple of rows. (Here correlation is based on Hamming distance. For example, p_1 is the largest fraction of columns in which any

three rows agree).

Proof: Probability that the most correlated triple of strings falls into the same bucket is p_1^q. (Note that each row is a string). Probability that this triple does not fall into the same bucket in a given iteration is $(1 - p_1^q)$. Thus, the probability that this triple does not fall into the same bucket in z successive iterations is $(1 - p_1^q)^z$. This probability is $\leq \exp(-zp_1^q)$ using the fact that $(1 - x)^{1/x} \leq 1/e$ for any $0 < x < 1$. In turn, this probability will be $\leq n^{-\alpha}$ if $z \geq \frac{\alpha \log n}{p_1^q}$.

To ensure that the run time is as small as possible, we want to ensure that the size of C is not too large. Since we spend a linear time in each iteration of the algorithm, an optimal strategy will be to make sure that the number of candidate triples generated in each iteration is also $O(n)$. If p_2 is the second largest correlation, the probability that any triple other than the largest correlated falls into the same bucket in any iteration is $\leq p_2^q$. If this probability is $\leq \frac{1}{n^2}$, then the expected number triples generated in any iteration will be $\leq n$. This happens if $q = \frac{2 \log n}{\log(1/p_2)}$. For this value of q, the value of z becomes $\alpha \log n \; n^{2 \log p_1 / \log p_2}$.

Given that the expected time we spend in hashing in each iteration and the time for generating the pairs is $O(n)$, it follows that the expected run time of the algorithm is $O\left(n^{1+\frac{2 \log p_1}{\log p_2}} \log n\right)$. $\square$

Note: The above algorithm is asymptotically better than the algorithm in [119] by a logarithmic factor. Let this algorithm be called MCTa.

The case of random data. To get an idea of how large $2\frac{\log p_1}{\log p_2}$ could get consider the case where all the strings are random (i.e., each character of any string is randomly chosen to be 0 or 1 with equal probability). Consider a collection of n random strings of length t each. Let A, B , and C be any three of these vectors. The expected correlation between A, B, and C is 1/4, i.e., the expected number of positions in which they match is $t/4$. We can use

Chernoff bounds to get high probability estimates on p_1 and p_2.

Chernoff bounds: If X is the sum of t independent Bernoulli trials with a probability p of success each, then the following are true, for any $0 < \delta < 1$ [115]:

$$Prob.[X < ((1-\delta)\mu] < \exp(-\mu\delta^2/2).$$

Here $\mu = tp$. Also,

$$Prob.[X > (1+\delta)\mu] < \exp(-\mu\delta^2/3).$$

Using the above bounds, it follows that the probability that the number of matches among A, B, and C is more than $(1+\delta)t/4$ is $\leq \exp(-t\delta^2/12)$. Thus the probability that there exists at least one triple whose number of matches is $> (1+\delta)t/4$ is $\leq n^3 \exp(-t\delta^2/12)$. As a result, we infer that the correlation of any triple is $\leq \frac{1}{4} + \frac{1}{2}\sqrt{\frac{9\ln n - 3\ln u}{t}}$ with probability $\geq (1-u)$, for any $0 < u < 1$. Similarly, the probability that none of the pairs has a correlation less than $\frac{1}{4} - \frac{1}{2}\sqrt{\frac{6\ln n - 2\ln u}{t}}$ is $\geq (1-u)$. Thus a high probability upper bound on p_1 is $\hat{p}_1 = \frac{1}{4} + \frac{1}{2}\sqrt{\frac{9\ln n - 3\ln u}{t}}$ and a high probability lower bound on p_2 is $\check{p}_2 = \frac{1}{4} - \frac{1}{2}\sqrt{\frac{6\ln n - 2\ln u}{t}}$.

Table 3.3.3 displays these estimates for different values of n. For this table, the value of t has been chosen to be $1,000$ and that of u is 0.01. It also shows the corresponding values of $2\frac{\log \hat{p}_1}{\log \check{p}_2}$. From this table we see that for a fixed value of t when the value of n increases, the value of $2\frac{\log \hat{p}_1}{\log \check{p}_2}$ decreases.

TABLE 1.3.4: Correlation estimates for the case of random data.

n	$\hat{p}_1$	$\check{p}_2$	$2\frac{\log \hat{p}_1}{\log \check{p}_2}$
10,000	0.4055	0.1230	0.8615
50,000	0.4167	0.1139	0.8059
100,000	0.4213	0.1101	0.7836

1.3.5 Our new algorithm

The run time of the above algorithm will not be feasible when n is in the millions. For instance if the run time is $n^{2.5}$, $n = 10^6$ and we can perform 10^9 operations in a second, then the run time needed will be more than 11.5 days! If k is more than 3, this time will run into years! Thus we need better algorithms.

In this section we present a greedy algorithm called MCTb with a much better run time. The idea is to employ the maximum correlated pairs algorithm (c.f. Theorem 1.3.2). Details follow.

Algorithm MCTb

1) Identify the most correlated c pairs of strings
   (using Theorem 1.3.2), for some constant c.
   Let Q be the set of these pairs. $E := \emptyset$.
2) **for** each pair $(a, b) \in Q$ **do**
   for each $c \in Q - \{a, b\}$ **do**
   $E := E \cup \{(a, b, c)\}$;
3) For each triplet x in E compute the correlation
   and output the desired number of
   maximum correlated triplets.

The above algorithm greedily generates candidate triplets starting from maximum correlated pairs and identifies the maximum correlated ones from these candidates. The total number of candidates generated is $O(cn)$. This will be $O(n)$ if c is a constant. Assuming that t is a constant, the time spent in step 2 is $O(n)$. The expected time spent in step 1 is $O\left(n^{1+\frac{\log p_1}{\log p_2}} \log n\right)$ (c.f. Theorem 1.3.2). Step 3 takes $O(n)$ time. Thus, the expected run

time of this algorithm is $O\left(n^{1+\frac{\log p_1}{\log p_2}} \log n\right)$. Clearly this is much better than the time reported in Theorem 1.3.5.

1.3.6 An experimental evaluation

In this experiment we have tested MCTb by *injecting* triples with known correlations and checking the time MCTb takes to find them. As an example, consider the problem of finding the largest correlated triple of rows in a $m \times n$ matrix M. Say we generate each row by picking each element to be either 0 or 1 with equal probability. For any three rows, clearly, the expected correlation is $\frac{1}{4}$. We can perform a probabilistic analysis to get a high probability bound on the largest correlation between any two rows (see Table 3.3.3). For example, for $n = 10,000$, we generated several random data sets and computed the largest correlation in each and calculated an average. Let p be this value. To inject a triple with a correlation of p' where p' is $> p$, Clearly, the correlation of the three rows r_1, r_2, and r_3 is p'. The expected correlation of r_1 with any pair of rows (not including r_2 and r_3) is $\frac{1}{4}$. A similar statement holds for the rows r_2 and r_3. Thus the triplet (r_1, r_2, r_3) is likely to be the winner with high probability. We have picked a value for p' that is only slightly larger than p so as to get an accurate estimate on the run time. We stop our algorithm MCTb after a fixed number of runs i.e., 50. We are always able to find the most correlated triple we injected within the fixed number of runs. In Table 3.3.4 we show the results for our algorithm. As evident from Table 3.3.4 MCTb finds the injected triple within a very short period of time. As an example consider the dataset D8. It consists of 8 million strings of length 512 each. It processed 2.52×10^{10} triples in only 80 minutes to find the most correlated triple. For the brute force algorithm we were not able to run it for a million (or more) number of strings since the run time is prohibitive. An estimated time for the brute force algorithm for a

million number of strings will be 1,864 years! The last column in Table 3.3.4 shows the ratio of the number of triples (that will be) generated by the brute force algorithm to the number of triples generated by our algorithm MCTb. This can also be thought of as the speedup of our algorithm over the brute force algorithm. Our algorithm is 9 orders of magnitude faster.

Another way of testing our algorithm will be as follows. In this case we do not inject any triple with known correlations. We may be interested in finding out how many of the top q correlated triples are found by MCTb (for some relevant q such as 10 or 100). This measure has been used in such articles as [101]. Of course, to obtain this measure we have to be able to run the brute force algorithm to find out the actual top q correlated triples. We use this measure to identify the performance of our algorithm for the three locus association problem. In this case, we have used small values of n.

TABLE 1.3.5: Number of triples and runtime of MCTb in Hamming space on binary strings. CPU times are given in minutes.

		Brute-force	MCTb		Gain
Dataset	Size	Processed Triplets	Processed Pairs & Triplets	CPU Time	Processed Triplets
D1	1×10^6	1.67×10^{17}	4.83×10^8	2.84	3.46×10^8
D2	2×10^6	1.33×10^{18}	1.73×10^9	7.68	7.72×10^8
D3	3×10^6	4.50×10^{18}	3.75×10^9	13.93	1.20×10^9
D4	4×10^6	1.06×10^{19}	6.51×10^9	22.97	1.64×10^9
D5	5×10^6	2.08×10^{19}	1.00×10^{10}	34.96	2.07×10^9
D6	6×10^6	3.60×10^{19}	1.43×10^{10}	47.31	2.51×10^9
D7	7×10^6	5.72×10^{19}	1.94×10^{10}	63.60	2.95×10^9
D8	8×10^6	8.53×10^{19}	2.52×10^{10}	80.32	3.38×10^9

1.4 Two Locus Association Problem

The two locus association problem is defined as follows. Input is a matrix M of size $(m_1 + m_2) \times n$ where $m_1 + m_2$ is the number of patients (subjects) each with n SNPs. Here m_1 is the number of cases and m_2 is the number of controls. The cases are of phenotype 1 and the controls are of phenotype 0. Rows 1 through m_1 of M correspond to cases. Let this submatrix be called A. Rows $m_1 + 1$ through $m_1 + m_2$ of M correspond to controls and let this submatrix be called B. Each column of M corresponds to an SNP. The two locus association problem is to identify the pair of SNPs whose statistical correlation with phenotype is maximally different between cases and controls. As mentioned in [101], the goal is to identify the pair:

$$\arg\max_{i,j} |P_A(i,j) - P_B(i,j)|.$$

If Q is any matrix, then, $P_Q(i,j)$ stands for the correlation between the columns i and j of Q.

The algorithm of [101] exploits the light bulb algorithm of [119] and locality sensitive hashing (LSH) [104]. They use LSH to transform matrices A and B to A' and B', respectively. In particular, each column c_i of A is converted to a column c_i' of zeros and ones. The size of c_i is $1 \times m_1$ and the size of c_i' is chosen to be $u = \max\{m_1, m_2\}$. The matrix B is also transformed into B' in a similar manner using LSH. Followed by this, the pair of interest is identified.

To be precise, using A' and B', the matrix D is formed where

$$D = \begin{bmatrix} A' & A' \\ B' & \bar{B}' \end{bmatrix}$$

where $\bar{B}'$ is obtained from B' by complementing every element of B'. Note that D is of size $2u \times 2n$. Let $D_1 = \{1, 2, \ldots, n\}$ and $D_2 = \{n+1, n+2, \ldots, 2n\}$. Consider all the pairs of columns (i, j) such that $i \in D_1$ and $j \in D_2$. From out of these pairs, identify the pair (i', j') of columns with the maximum number of matches. If $i' = a$ and $j' = n + b$, then (a, b) is the pair of interest. The authors of [101] find this pair using the light bulb algorithm of [119].

We can use MCP instead of the light bulb algorithm to get the following theorem. In this case the run time will improve by more than a logarithmic factor.

Theorem 1.4.1. We can find the pair (i, j) of SNPs that maximizes $|P_A(i, j) - P_B(i, j)|$ in an expected time of $O\left(n^{1+\frac{\log p_1}{\log p_2}} \log n\right)$, where p_1 and p_2 are the smallest and the next smallest values of $|P_A(i, j) - P_B(i, j)|$, respectively, over all possible pairs (i, j) of SNPs.

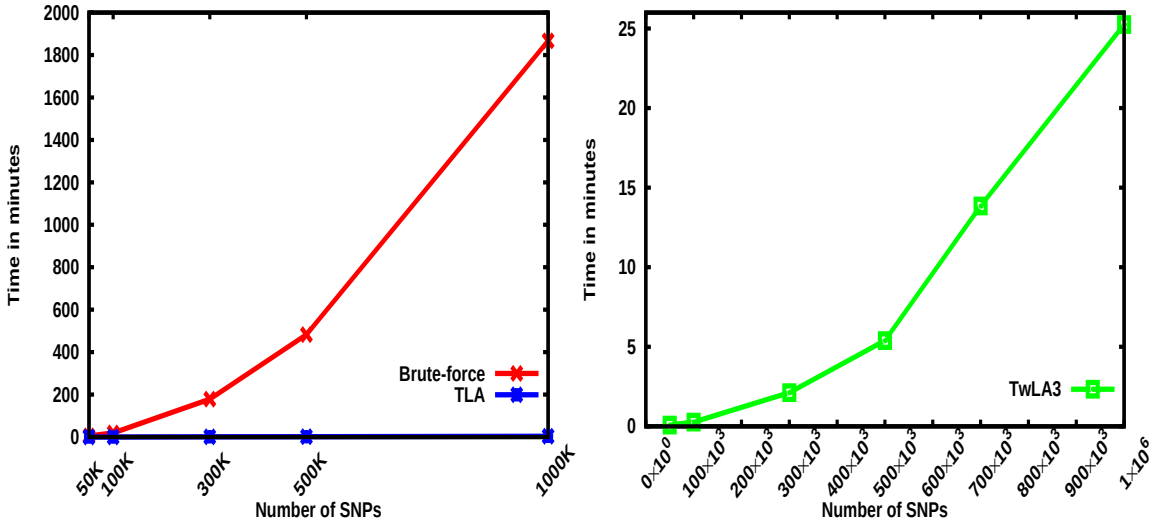
Proof: Similar to that of Theorem 1.3.1 and hence omitted. We call our algorithms for the two locus problem as TwLA2 and TwLA3 for binary and ternary cases, respectively.

TABLE 1.4.1: # of pairs and runtime comparisons on binary datasets for two locus problem. CPU times are given in minutes.

	Brute-force		TwLA2				Gain	Speed up
# SNPs	Processed Pairs	CPU Time	Processed Pairs	CPU Time	Top 10	Top 100	Processed Pairs	Time
5×10^4	25×10^8	4.97	715,553	0.13	0.74	0.87	3,493.80	39.29
1×10^5	10×10^9	19.00	2,861,457	0.22	0.74	0.92	3,494.72	84.85
3×10^5	90×10^9	178.23	25,749,044	0.87	0.80	0.94	3,495.28	203.96
5×10^5	25×10^{10}	482.30	71,529,899	1.35	0.88	0.83	3,495.04	356.09
1×10^6	10×10^{11}	1,867.05	286,110,791	3.48	0.80	0.75	3,495.15	536.00

TABLE 1.4.2: # of pairs and runtime comparisons on ternary datasets for two locus problem. CPU times are given in minutes.

	Brute-force		TwLA3			Gain	Speed up
# SNPs	Processed Pairs	CPU Time	Processed Pairs	CPU Time	Top 100	Processed Pairs	Time
5×10^4	2.50×10^9	5.31	5.72×10^7	0.08	0.81	43.69	63.35
1×10^5	1.00×10^{10}	21.09	2.36×10^8	0.27	0.79	42.37	78.42
3×10^5	9.00×10^{10}	194.43	2.12×10^9	2.12	0.78	42.40	91.50
5×10^5	2.50×10^{11}	482.30	5.63×10^9	5.39	0.76	44.36	91.29
7×10^5	4.90×10^{11}	1,051.36	1.18×10^{10}	13.85	0.78	41.67	75.87
1×10^6	1.00×10^{12}	T/O	1.20×10^{10}	25.26	–	83.33	–



(A) CPU times consumed by TwLA2 method (B) CPU times consumed by TwLA3 method

FIGURE 1.4.1: CPU elapsed times of TwLA2 and TwLA3 methods.

1.4.1 An experimental comparison

The notion of similarity (between two SNPs) used in [101] is Pearson's correlation coefficient. In this research work the similarity we use is based on the Hamming distance. We believe that the Hamming distance is a better measure since it can be used to derive other measures. In terms of run times, these two measures do not have a significant difference. Specifically, the complement of the Hamming distance is the measure of similarity we employ. The authors of [101] have tested their algorithms on different data sets (including random data). Since we do not have access to either of these data sets or their programs, the only comparison we can do was on the random data. As explained in [101], we have also generated SNPs from binomial distributions. In particular, for each subject, the value of each SNP is chosen uniformly randomly to be either 0 or 1 with equal probability. This dataset is called NOISE Data in [101]. The authors of [101] note that random data are the hadest instances for

the identification of the most (and least) correlated pair as well as the two locus problem. We have generated data of sizes 50K, 100K, 300K, 500K and 1,000K with 50 cases and 50 controls to demonstrate how effective our algorithm is. For example FastANOVA [131] took almost 1000 seconds for 42K SNPs. On the other hand for 50K SNPs our algorithm took only 7.8 seconds.

We have also compared the performance of our algorithm with the algorithm reported in [101]. Our results are shown in Table 1.4.1. The recall rates for us are much better than those of [101] (see Table 6 in [101]). Recall rates refer to the accuracy of the different algorithms in terms of how many of the top q correlated pairs were identified by these algorithms (for different values of q). For example, for 50K SNPs, the top 10 recall rate of [101] is 0.2, whereas it is 0.74 for our algorithm. As another example, for 100K SNPs, the top 10 recall rate of [101] is 0 whereas it is 0.74 for our algorithm. Table 3.3.2 displays the result for ternary (i.e., where SNPs are encoded by 0, 1, or 2) datasets.

In terms of run times we can only do an estimation of the speedup since they [101] do not report any actual run times. They report that the brute force algorithm takes several days using 1000 processors (for less than 1000K SNPs). To give the benefit of doubt, we take the time as 2 days. This means that for 100K SNPs, the brute force algorithm took 20 days on a single processor. For 100K SNPs, in Table 6 of [101], they indicate that their algorithm achieves a speedup of 184 over the brute force. This means that their algorithm took more than 156 minutes. Also they have used 400 subjects whereas we have used 100 subjects. This means that on 100 subjects and 100K SNPs, their algorithm will take 39 minutes. Our run time for 100K SNPs and 100 subjects is 0.22 minutes. This means that our algorithm is more than 177 times faster. Note that this is only a very conservative estimate. Also note that the improvement our algorithm achieves is quite significant since the typical processing times reported in the literature for the two locus problem are quite high. The improvement

we achieve is not only due to the novel algorithm but also due to a number of techniques we have used at the implementation level. We have used bit operations as much as possible. This also reduces the memory used. Any improvement in the run time achieved for solving the two locus problem could make a noticeable difference in GWAS. Please, see Figure 3.2.1 for visual details.

1.5 Three Locus Association Problem

We can define the three locus problem along the same lines as the two locus problem. We are interested in finding thee SNPs that are differentially correlated in a set of cases vs a set of controls. Input is a matrix M of size $(m_1 + m_2) \times n$ where $m_1 + m_2$ is the number of patients (subjects) each with n SNPs. Here m_1 is the number of cases and m_2 is the number of controls. There are three possible values for each SNP, namely, $0, 1$, or 2. The cases are of phenotype 1 and the controls are of phenotype 0. Rows 1 through m_1 of M correspond to cases. Let this submatrix be called A. Rows $m_1 + 1$ through $m_1 + m_2$ of M correspond to controls and let this submatrix be called B. Each column of M corresponds to an SNP. The three locus association problem is to identify the triple of SNPs whose statistical correlation with phenotype is maximally different between cases and controls. The goal is to identify a triple:

$$\arg\max_{i,j,k} |P_A(i,j,k) - P_B(i,j,k)|.$$

If Q is any matrix, then, $P_Q(i,j,k)$ stands for the correlation between the columns i, j, and k of Q.

A brute force algorithm can be conceived of that takes $O(n^3t)$ time as follows.

1) Construct a set C of all possible triples of SNPs.

2) *bestdiff*:=0; *besttriple*:=(0, 0, 0);

3) **for** each triple $(i, j, k) \in Q$ **do**

4) **if** *bestdiff* $< |P_A(i, j, k) - P_B(i, j, k)|$ **then**

5) *bestdiff* $:= |P_A(i, j, k) - P_B(i, j, k)|$;

6) *besttriple*:=(i, j, k);

7) Output *besttriple*.

The brute force algorithm will not be practically feasible. For instance if n is a million, the number of subjects is 100, and 10^9 basic operations take 1 second, the total time taken by the brute force algorithm will be 5.28×10^{11} years! To the best of our knowledge the three locus problem has not been addressed in the literature thus far. We propose an algorithm, called ThLA3 (Three Locus Algorithm for the case of an alphabet with three characters) similar to MCTb. We first identify a constant c number of the most relevant pairs of SNPs using the two locus algorithm TwLA3. Followed by this we generate candidate triples by augmenting one SNP at a time to each of these pairs. For each candidate we compute a score and output the triple whose score is the maximum. More details follow. By the most relevant c pairs of SNPs we mean c pairs (i, j) of SNPs whose $|P_A(i, j) - P_B(i, j)|$ value is the maximum.

Algorithm ThLA3

1) Identify the most relevant c pairs of SNPs
   (using the algorithm TwLA3), for some constant c.
   Let Q be the set of these pairs. $E := \emptyset$.

2) **for** each pair $(a, b) \in Q$ **do**

for each $c \in Q - \{a, b\}$ **do**

$E := E \cup \{(a, b, c)\}$;

3) For each triplet (i, j, k) in E compute

$|P_A(i, j, k) - P_B(i, j, k)|$ and output the desired

number of triples (i, j, k) whose

$|P_A(i, j, k) - P_B(i, j, k)|$ value is maximum.

We can analyze the algorithm ThLA3 along the same lines as MCTb and understand why this algorithm can be expected to perform well in practice. Expected time taken by step 1 is $O\left(n^{1+\frac{\log p_1}{\log p_2}} \log n\right)$, where p_1 and p_2 are the smallest and the next smallest values of $|P_A(i, j) - P_B(i, j)|$, respectively, over all possible pairs (i, j) of SNPs (c.f. Theorem 1.4.1). Step 2 takes $O(n)$ time. Step 3 also takes $O(n)$ time, assuming that t is $O(1)$. As a result, the expected run time of this algorithm is $O\left(n^{1+\frac{\log p_1}{\log p_2}} \log n\right)$. Experimental results reported in the next section reveal that this algorithm indeed performs well in practice.

TABLE 1.5.1: # of triples and runtime comparisons in Hamming space on binary datasets. CPU times are given in minutes.

	Brute-force		ThLA2			Gain	Speed up
# SNPs	Processed Triples	CPU Time	Processed Pairs & Triples	CPU Time	Top 10	Processed Triples	Time
1×10^3	1.67×10^8	5.03	1.10×10^6	0.01	0.55	1.51×10^2	672.89
2×10^3	1.33×10^9	39.64	2.39×10^6	0.01	0.72	5.56×10^2	2,890.32
3×10^3	4.50×10^9	129.10	3.89×10^6	0.02	0.72	1.16×10^3	6,682.5
1×10^5	1.67×10^{14}	–	1.08×10^9	1.19	–	1.54×10^5	–
3×10^5	4.50×10^{15}	–	9.10×10^9	7.19	–	4.95×10^5	–
5×10^5	2.08×10^{16}	–	2.49×10^{10}	22.27	–	8.36×10^5	–
7×10^5	5.72×10^{16}	–	4.86×10^{10}	45.17	–	1.18×10^6	–
1×10^6	1.67×10^{16}	–	9.78×10^{10}	98.50	–	1.71×10^6	–

TABLE 1.5.2: # of triples and runtime comparisons in Hamming space on ternary datasets. CPU times are given in minutes.

	Brute-force		ThLA3			Gain	Speed up
# SNPs	Processed Triples	CPU Time	Processed Pairs & Triples	CPU Time	Top 10	Processed Triples	Time
1×10^3	1.67×10^8	6.72	2.04×10^6	0.01	0.52	81	662.23
2×10^3	1.33×10^9	53.91	6.36×10^6	0.02	0.60	209	2,568.75
3×10^3	4.50×10^9	182.61	1.28×10^7	0.04	0.63	352	5,502.22
1×10^5	1.67×10^{14}	–	9.32×10^8	1.57	–	1.79×10^5	–
3×10^5	4.50×10^{15}	–	7.74×10^9	10.97	–	5.82×10^5	–
5×10^5	2.08×10^{16}	–	2.07×10^{10}	27.79	–	1.01×10^5	–
7×10^5	5.72×10^{16}	–	4.31×10^{10}	55.78	–	1.32×10^6	–
1×10^6	1.67×10^{16}	–	7.87×10^{10}	124.83	–	2.12×10^6	–

1.5.1 An experimental evaluation

To demonstrate the effectiveness of our three locus algorithms we have done extensive experiments on both binary (i.e. NOISE dataset) and ternary random datasets. We have generated data of sizes 1K, 2K, 3K, 50K, 100K, 300K, 500K, 700K and $1,000$K with 50 cases and 50 controls and we did not inject any correlated triplets in any of the datasets stated above. Since the brute-force method is a cubic time algorithm, it is not feasible to run it over big datasets. The estimated time to get results from brute-force method for 100 thousand SNPs is over 9 years! So, to prove the reliability of our algorithms ThLA2 and ThLA3, we have run the brute-force algorithm on smaller datasets and got the top ten closest triples. We then executed our algorithms on the same datasets and collected the top 10 triples and compared the results with those from the brute-force method. It is evident from Table 1.5.1 and Table 1.5.2 that both ThLA2 and ThLA3 correctly identify most of the top 10 SNP triplets. The speedups the algorithms achieve are more than $5,000\times$ on a dataset consisting of $3,000$ SNPs. On a dataset of size one million, ThLA2 and ThLA3 took around 100 to 120 minutes only. The runtimes are approximately linearly proportionate with the number of SNPs.

1.6 Conclusions

In this research work we have presented a novel algorithm for the two locus problem that plays a central role in GWAS. Our algorithm is two orders of magnitude faster than previous algorithms. We have also offered a greedy algorithm for finding the most correlated triple of strings. This algorithm is several orders of magnitude faster than the brute force algorithm. For the first time we consider the very challenging three locus problem. We also

offer algorithms for the closest pair problem (CPP). CPP is a ubiquitous problem that has numerous applications in varied domains. We have offered algorithms for Hamming distance. In summary, we improve the results presented in many prior papers.

Chapter 2

Genotype-phenotype Correlational Analysis

Single Nucleotide Polymorphisms (SNPs) are sequence variations found in individuals at some specific points in the genomic sequence. As SNPs are highly conserved throughout evolution and within a population, the map of SNPs serves as an excellent genotypic marker. Conventional SNPs analysis mechanisms suffer from large run times, inefficient memory usage, and frequent overestimation. In this research work, we propose efficient, scalable, and reliable algorithms to select a small subset of SNPs from a large set of SNPs which can together be employed to perform phenotypic classification. Our algorithms exploit the techniques of gene selection and random projections to identify a meaningful subset of SNPs. To the best of our knowledge, these techniques have not been employed before in the context of genotype-phenotype correlations. Random projections are used to project the input data into a lower dimensional space (closely preserving distances). Gene selection is then applied on the projected data to identify a subset of the most relevant SNPs. We have compared the performance of our algorithms with one of the currently known best algorithms called

Multifactor Dimensionality Reduction (MDR), and Principal Component Analysis (PCA) technique. Experimental results demonstrate that our algorithms are superior in terms of accuracy as well as run time. In our proposed techniques, random projection is used to map data from a high dimensional space to a lower dimensional space, and thus overcomes the curse of dimensionality problem. From this space of reduced dimension, we select the best subset of attributes. It is a unique mechanism in the domain of SNPs analysis, and to the best of our knowledge it is not employed before. As revealed by our experimental results, our proposed techniques offer the potential of high accuracies while keeping the run times low.

2.1 Introduction

A single-nucleotide polymorphism (SNP) is defined as a DNA sequence variation where a single nucleotide, i.e., A, T, C, or G in the genomic sequence differs among the individuals of a biological species. It is the most common type of genetic variation among people. If CCGAATC and CCGAATA are two sequenced DNA fragments from two different individuals, these fragments differ in only one nucleotide position and this is called a SNP [1]. If we make comparisons between any two human genomic sequences side by side, they will be almost 99.9% identical [2]. Having 3.2 billion base-pair genomes, individuals can have some 3.2 million differences in diploid genome. Most of the differences are due to SNPs. Even though most of the SNPs are of no biological significance or meaning, a fraction of the substitutions have functional consequence and these variations are the basis for the diversity found among humans [3]. SNPs are not evenly distributed across the whole genomic sequence. They occur more frequently in non-coding regions than in coding regions of the genomic sequence. Most

SNPs have no effect on health or development. Some of these genetic differences, however, have proven to be very important in the study of human health. Researchers have found SNPs that may help predict an individual's response to certain drugs, susceptibility to environmental factors such as toxins, and risk of developing particular diseases. SNPs can also be used to track the inheritance of genes accused for disease within families. Future studies will work to identify SNPs associated with complex diseases such as heart disease, diabetes, and cancer.

In this research work, the main problem of interest is to take as input (say) two groups of individuals separated based on some phenotypes, together with their genotypes information and identify the most relevant SNPs that can explain the groupings. Our new approach is based on two paradigms: gene selection [135], and random projections [136] to identify a subset of SNPs from a set of SNPs that can altogether differentiate two groups of individuals efficiently and reliably within a short amount of time. In the first approach, we employ a feature selection algorithm (FSA) to identify the k most relevant SNPs (where k can be chosen by the user) to differentiate a group of individuals from another. To validate this approach, we computed the p-value for each of the SNPs. It is found that a significant number of SNPs selected by the FSA has a very low p-value. In the second approach, we employ random projections to project the original data into a space of dimension d (where d can be chosen by the user). We then compute a subset of dimensions which can together differentiate two groups of individuals. We have done this in two steps. We take the best m SNPs found by using the FSA. For each subject we keep only these m SNPs. The modified dataset is then projected onto a k-dimensional space for various values of k. The FSA is then employed to identify a subset of dimensions that can best predict a particular class of subjects. Both of these approaches yield very good outcomes and our simulation results show that our proposed algorithms are indeed reliable, scalable, and efficient. They also

outperform one of the currently best performing algorithms [137] in terms of accuracy and runtime.

2.2 Background Summary

2.2.1 Data source

In this research work, we have performed a candidate gene study for a complex human behavior disorder, drug dependency using scalable, and efficient computational techniques. Although candidate gene studies have their own inherent limitations (reviewed in [138]), the use of smaller focused arrays possibly represents a more practical approach for many studies than the use of large scale arrays such as genome wide association studies (GWAS). These focused arrays are able to overcome the issues of inadequate gene coverage by providing full coverage for a limited number of candidate genes. Such focused arrays offer the advantages of lower cost and lower false discovery rate, especially in situations where a dataset may have inadequate power due to size or other reasons. Our genetic markers were obtained in a study conducted by National Institute of Alcohol Abuse and Alcoholism (NIAAA). For details about our data readers are referred to [139]. According to [139], the panel SNPs that we use in our study are able to extract full haplotype information for candidate genes in alcoholism, other addictions and disorders of mood and anxiety.

2.2.2 Feature selection

Feature selection techniques are used to efficiently select a subset of SNPs from a set of SNPs which can best define a system. They are different from other dimensionality reduction

techniques like projection-based (e.g., principal component analysis, random projections) or compression-based (e.g., using information theory) techniques. The latter techniques do not alter the original representation of the variables but just select a subset of them to best describe a system. A comprehensive and detailed review on feature selection techniques in bioinformatics can be found in [140]. Machine learning techniques can also be applied in the domain of SNPs selection [141]. Support Vector Machine (SVM), Genetic Algorithm (GA), Simulated Annealing (SA), Principal Component Analysis (PCA), etc have been applied widely in bioinformatics. Examples of works that employ SVM are [142, 143, 144]. [145] detects a subset of potential SNPs by using Simulated Annealing (SA) and also provides a comprehensive and detailed review of the current approaches to identify SNPs. PCA based research can be found, for example, in [146, 147].

2.2.3 Support vector machine (SVM)

Support Vector Machine (SVM) has been developed by Vapnik, et al. at AT&T Bell Laboratories [78, 79] which is the basis of gene selection algorithm. Kernel-based techniques (such as support vector machines, Bayes point machines, kernel principal component analysis, and Gaussian processes) represent a major development in machine learning algorithms. Support vector machines (SVMs) are a group of supervised learning methods that can be applied to classification or regression. They represent an extension to nonlinear models of the generalized portrait algorithm. The basic idea is to find a hyperplane which separates any given d-dimensional data perfectly into two classes. Assume that we are given l training examples $\{x_i, y_i\}$, where each example has d inputs ($x_i \epsilon \Re^d$), and a class label $y_i \epsilon \{-1, 1\}$ where $1 \leq i \leq l$. Now, all the hyperplanes in $\Re^d$ are parameterized by a vector (w), and a

constant (b), expressed in the equation:

$$w \cdot x + b = 0 \tag{2.2.1}$$

Here x is a point on the hyperplane, w is a n-dimensional vector perpendicular to the hyperplane, and b is the distance of the closest point on the hyperplane to the origin. Any such hyperplane (w, b) that separates the data leads to the function:

$$f(x) = sign(w \cdot x + b) \tag{2.2.2}$$

The hyperplane is found by solving the following problem:

Minimize $J = \frac{1}{2} \parallel w\|^2$; subject to $y_i(w \cdot x_i + b) - 1 \geq 0$, where $i = 1, \ldots, l$.

To handle datasets that are not linearly separable, the notion of a "kernel induced feature space" has been introduced in the context of SVMs. The idea is to cast the data into a higher dimensional space where the data is separable. To do this, a mapping function $z = \phi(x)$ is defined that transforms the d dimensional input vector x into a (usually higher) d' dimensional vector z. Whether the new training data $\{\phi(x_i), y_i\}$ is separable by a hyperplane depends on the choice of the mapping/kernel function. Some useful kernel functions are "polynomial kernel", and "GAUSSIAN RBF kernel". The *polynomial kernel* takes the form:

$$K(x_a, x_b) = (x_a \cdot x_b + 1)^p \tag{2.2.3}$$

where p is a tunable parameter, which in practice varies from 1 to $\sim$ 10. Another popular one is the Gaussian RBF Kernel:

$$K(x_a, x_b) = \exp\left(-\frac{\parallel x_a - x_b\|^2}{2\sigma^2}\right) \tag{2.2.4}$$

where σ is a tunable parameter. Using this kernel results in the classifier:

$$f(x) = sign\left[\sum_i \alpha_i y_i \exp\left(-\frac{\| x - x_i\|^2}{2\sigma^2}\right) + b\right] \tag{2.2.5}$$

which is a Radial Basis Function, with the support vectors as the centers. More details and applications of SVM can be found in [202, 197, 193].

2.2.4 Principal component analysis (PCA)

Principal component analysis (PCA) is a technique that takes any high-dimensional data to a lower-dimensional form by using the dependencies among the variables, without losing too much information. PCA is one of the simplest and most robust ways of doing such dimensionality reduction. It employs orthogonal transformations to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables. These uncorrelated variables are called principal components. PCA is also known as the Karhunen-Loeve transformation, the Hotelling transformation, the method of empirical orthogonal functions, and singular value decomposition. The number of principal components is less than or equal to the number of original variables. Here the first principal component has the largest possible variance.

Assume that we are given n-dimensional feature vectors and we want to summarize them by projecting it into a d-dimensional subspace. The simplest solution is to find the projections which maximize the variance. The first principal component is the direction in the feature space along which the projections have the largest variance. The second principal component is the direction which maximizes the variance among all the directions orthogonal to the first. The k^{th} component is the variance-maximizing direction orthogonal to the

previous $k-1$ components. More information regarding PCA can be found in [151, 152, 153].

2.3 Methods

In this section we summarize the *Feature Selection Algorithm* as well as the technique of *Random Projections*. Feature selection is a classification algorithm based on SVMs. For any classification algorithm there will be two phases. In the first phase the classifier is trained with some training data and this phase can be thought of as a learning phase. In the second phase, the classifier's accuracy is tested with test or treatment data. In this research work we utilize real data pertaining to subjects dependent on opium. We divide the set of input data into two groups: G_1 contains all the non-addicted subjects and G_2 is the set of all addicted subjects. We train the classifier using a training set which consists of 50 percent data from each of G_1 and G_2 (randomly chosen), respectively. The test set is formed using the other 50 percent from G_1 and G_2, respectively.

2.3.1 Feature selection

We have incorporated gene selection techniques [135] in our feature section algorithm to identify the correlation among the SNPs. The aim of gene selection algorithm is to identify the (smallest) subset of genes responsible for certain event(s). Please note that even though in the gene selection algorithm we refer to genes, the algorithm is generic and in general a 'gene' should be thought of as an arbitrary feature. Gene selection is based on SVMs and it takes as input n genes $\{g_1, g_2, g_3, \cdots, g_n\}$, and l vectors $\{v_1, v_2, v_3, \cdots, v_l\}$. As an example, each v_i could be an outcome of a microarray experiment and each vector could be of the following form: $v_i = \{x_i^1, x_i^2, x_i^3, \cdots, x_i^n, y_i\}$. Here x_i^j is the expression level of

the j^{th} gene g_j in experiment i. The value of y_i is either *+1* or *-1* based on whether the event of interest is present in experiment i or not. The problem is to identify a subset of genes $\{g_i^1, g_i^2, g_i^3, \cdots, g_i^m\}$ sufficient to predict the value of y_i in each experiment. Given a set of vectors, the gene selection algorithm learns to identify the minimum subset of genes needed to predict the event of interest and the prediction function. These vectors form the training set for the algorithm. Once trained, the algorithm is provided with a new set of data which is called the test set. The accuracy of gene selection is measured in the test set as a percentage of microarray data on which the algorithm correctly predicts the event of interest. The procedure solely relies on the concept of SVM.

Guyon, *et. al.* [194] introduced a naive gene selection algorithm called sort-SVM. Here the genes were sorted according to their corresponding weights and a subset of genes was selected from the sorted sequence and thus discarded the redundant information. The authors also developed an algorithm called Recursive Feature Elimination (RFE) which is based on the sensitivity analysis proposed by [155] where the change of cost function $DJ(i)$ caused by removing a given feature i is approximately measured by expanding the cost function (J) in Taylor series to second order. As a result, genes can be selected based on the weight value of each feature. In each iteration train the SVM and obtain the weights for all the remaining genes and then eliminate the gene with the smallest weight until two genes are left. Following are the basic steps involved in the Recursive Feature Elimination (RFE) algorithm: (1) Train the linear SVM; (2) Compute weight for each gene; (3) Remove the gene with the smallest weight; and (4) Repeat steps 1, 2, and 3 until only 2 genes are left.

The gene selection algorithm of Song and Rajasekaran [135] is based on the ideas of combining the mutual information among the genes and incorporating correlation information to reject the redundant genes. The Greedy Correlation Incorporated Support Vector Machine (GCI-SVM) algorithm of [135] can be briefly summarizes as follows: The SVM is trained only

once and the genes are sorted according to the norm of the weight vectors corresponding to these genes. Then the sorted list of genes are examined starting from the second gene. The correlation of each of these genes with the first gene is computed until one whose correlation with the first one is less than a certain predefined threshold is found. At this stage this gene is moved to the second place. Now the genes starting from the third gene are examined and the correlation of each of these genes with the second gene is computed until a gene whose correlation with the second gene is less than the threshold is encountered. The above procedure is repeated until the end of the sorted genes is reached. In the last stage, genes based on this adjusted sorted genes are selected. GCI-SVM brings the concept of sort-SVM and RFE-SVM altogether which makes it more efficient. These are: (1) GCI-SVM incorporates correlation information to remove the redundant genes; (2) Sort-SVM utilizes mutual information among genes but also may select redundant genes. GCI-SVM uses RFE-SVM concept which enables it to utilize the mutual information among genes; and (3) Other algorithms like RFE-SVM make use of recursion to remove the redundant genes which is very time consuming. GCI-SVM uses the combination of the above mentioned concepts together. This makes it time efficient. In a nutshell, GCI-SVM works as follows:

1. Compute the correlation coefficient for each pair of genes.
2. Train the SVM using the training data set.
3. Sort the genes based on their weight values.
4. Go through the sorted genes; pick those genes whose correlation with the previously picked genes is less than a threshold.
5. Move in order all picked genes to the front of the sequence; correspondingly, unpicked genes are moved to the end.

2.3.2 Random projections

Mapping a set of points from a higher dimensional space to a lower dimensional space in such a way that the pair-wise distances are closely preserved is a problem that has been studied widely. A finite set of n points in a d-dimensional Euclidean space R^d can be represented by a matrix $[A]_{n\times d}$, where each row represents a point in d dimensions. The objective is to identify a mapping $f : R^d \rightarrow R^k$ with negligible distortion in the distance between any pair of points. Here k is the dimension of the reduced space. Johnson and Lidenstrauss [156] have given an elegant randomized mapping such that the original pairwise distances are ϵ-preserved in the k-dimensional space.

Lemma (Johnson & Lindenstrauss): Given $\epsilon > 0$ and an integer n, let k be a positive integer such that $k > k_0 = O(\epsilon^{-2} \log n)$. For every set P of n points in R^d there exists $f : R^d \rightarrow R^k$ such that for all u, v in P:

$$(1-\epsilon) \parallel u-v \parallel^2 \leq \parallel f(u)-f(v) \parallel^2 \leq (1+\epsilon) \parallel u-v \parallel^2 \tag{2.3.1}$$

We can accomplish this mapping using the Achlioptas [136] method.

Theorem: Let P be an arbitrary set of n points in R^d, represented by a $n \times d$ matrix A. Given ϵ and $\beta > 0$, let,

$$k_0 = \frac{(4+2\beta)\log n}{\left(\frac{\epsilon^2}{2} - \frac{\epsilon^3}{3}\right)}. \tag{2.3.2}$$

For any integer $k > k_0$, let R be a $d \times k$ random matrix with $R(i,j) = r_{ij}$, where $\{r_{ij}\}$ are

independent random variables from either one of the following probability distributions:

$$r_{ij} = \begin{cases} +1 \text{ with probability } \frac{1}{2} \\ \\ -1 \text{ with probability } \frac{1}{2} \end{cases}$$

or,

$$r_{ij} = \sqrt{3} \begin{cases} +1 \text{ with probability } \frac{1}{6} \\ \\ 0 \text{ with probability } \frac{2}{3} \\ \\ -1 \text{ with probability } \frac{1}{6}. \end{cases}$$

Let $E = \frac{1}{\sqrt{k}} AR$ and let $f : R^d \rightarrow R^k$ map the i^{th} row of A to the i^{th} row of E. With a probability of at least $1 - n^{\beta}$, for all u, v in P, the following inequality holds:

$$(1 - \epsilon) \parallel u - v \parallel^2 \leq \parallel f(u) - f(v) \parallel^2 \leq (1 + \epsilon) \parallel u - v \parallel^2 \tag{2.3.3}$$

Using one of the probability distributions we can construct $[R]_{d \times k}$. Multiplication of $[A]_{n \times d}$ and $[R]_{d \times k}$ maps R^d to R^k.

2.3.3 Multifactor dimensionality reduction (MDR)

Multifactor dimensionality reduction (MDR) is a data mining procedure which detects and characterizes combinations of attributes or independent variables that can altogether interact to influence a dependent or class variable. MDR is designed primarily to identify interactions among discrete variables that can together act as a binary classifier. It is considered as a

nonparametric alternative to traditional statistical methods e.g., logistic regression. We can think of MDR as a constructive induction algorithm that can convert two or more discrete variables or attributes to a single variable or attribute. The method to create a new attribute or variable changes the representation space of the original data. The details of the MDR algorithm can be found in [137, 157, 158]. Authors in [159] develop the MDR-PDT algorithm by merging the MDR method with the genotype-Pedigree Disequilibrium Test (geno-PDT). Unlike ordinary MDR, it can identify single-locus effects or joint effects of multiple loci in families of diverse structure.

In the MDR algorithm, the observed data is divided into ten equal parts and a model is fit to each nine-tenths of the data (the training data), and the remaining one-tenth (the test data) is used to assess the accuracy to fit a model, thus using ten-fold cross-validation. Within each nine-tenths of the data, a set of n factors is selected and their possible multifactor classes or cells are represented in n dimensional space. The steps of the MDR algorithm, according to [137], can be described as follows:

1. In step one, the dataset is divided into multiple partitions to carry out cross-validation. MDR can be performed without performing cross-validation. But this is very infrequently done due to the potential for over-fitting [160]. It tries to fit the data, learn a concept, build a model based on the learned concept and apply the concept to predict from unseen data.

2. A subset of n discrete variables or factors is selected from the set of all variables or factors.

3. The chosen n variables and their possible multifactor classes are organized into n-dimensional space. For example, for two loci with three genotypes each, there are nine

possible two locus-genotype combinations. Then, the ratio of the number of cases to the number of controls is calculated within each multifactor class.

4. A reduction procedure on the n dimensional model to a one-dimensional model is carried out. This is done by labeling each multifactor class in n-dimensional space either as high-risk or low risk. If the cases to controls ratio meets or exceeds some threshold (e.g., ≥ 1.0), it is called high-risk. On the contrary, it is called low-risk, if that threshold is not exceeded. By following the procedure stated above, a model for both cases and controls is formed by pooling high-risk cells into one group and low-risk cells into another group. This reduces the n-dimensional model to a one-dimensional model (i.e., having one variable with two multifactor classes – high risk and low risk). In a nutshell, among all of the two-factor combinations, a single model that has the fewest misclassified individuals is selected.

5. The prediction error of each model is estimated by 10-fold cross-validation.

2.3.4 Normalization

Normalization is the process of scaling any data so that it falls within a specified range. There are many methods of normalization, such as min-max normalization, z-score normalization, normalization by decimal scaling, etc.

Min-max normalization

In min-max normalization, a linear transformation is performed on the original data. Assume that the minimum and maximum values of an attribute a are given by min_a, and max_a. Min-

max normalization maps a value v to v' in the new range $[new_{min_a}, new_{max_a}]$ by computing:

$$v' = \frac{v - min_a}{max_a - min_a}(new_{max_a} - new_{min_a}) + new_{min_a} \tag{2.3.4}$$

2.3.5 Discretization

Discreetization is the method of placing continuous values into discrete buckets. The simplest method for discretization is to determine the minimum and maximum values of the attributes and then divide the range into user defined number of intervals of equal length. Each interval I is associated with an integer value $V(I)$. Any value that falls in a particular interval I is mapped to the corresponding value $V(I)$.

2.4 Our algorithms

We have employed a dataset consisting of 1036 subjects denoted as s_1, s_2, s_3, $\cdots$, s_{1036} and 1212 SNPs denoted as snp_1, snp_2, snp_3, $\cdots$, snp_{1212}. The subjects are divided into two major groups as described above. $Group_1$ consists of subjects who are not addicted to opium and $Group_2$ consists of subjects who are addicted to opium. The input dataset can be represented as a 1036×1212 matrix. Our goal is to identify a subset of SNPs that can correlate well with the grouping. We have employed several versions of our algorithms and the details are summarized below:

2.4.1 Algorithm 1

In this algorithm [Please see Algorithm 1], we have used the feature selection algorithm to identify some of the best SNPs that can together identify two groups. The feature selection algorithm has two phases. In the first phase, the algorithm is trained with a training dataset. In this phase, the algorithm comes up with a model of concept. In the second phase of the algorithm, a test dataset is presented. The model learned in the first phase is used to classify the elements in the test dataset. As a result, the accuracy of the model learned can be computed. We divide the set of input data into two groups: $Group_1$ contains all the non-addicted subjects and $Group_2$ is the set of all addicted subjects. We train the classifier using a training set which consists of 50 percent of data from each of $Group_1$ and $Group_2$ (data is chosen randomly), respectively. The test set is formed using the other 50 percent from $Group_1$ and $Group_2$, respectively. Details are given in Algorithm 1. FSA is trained with the training set and it builds a model of concept by using SVM. We have used a number of kernel methods in SVM including Linear, Polynomial, GAUSSIAN RBF, and Sigmoid to build the model. The result is a $n \times m$ matrix, where n is the number of subjects and m is most influential features (here SNPs) of the training dataset by which we can infer whether a particular subject of interest is in $Group_1$ or $Group_2$ with certain confidence (here accuracy). After finding such features we calculate p-values of each feature and output it in increasing order of p-values along with accuracy.

Algorithm 1 Finding best SNPs using FSA

Input: $Group_1$, $Group_2$
Output: Best m SNPs and their p-values along with accuracy
begin

1. Construct training and test sets from $Group_1$ and $Group_2$.
2. Use the training set to train the feature selection algorithm and build the model of concept.
3. Select the most significant m SNPs to represent the genotype of the addicted subjects. Output of this stage is a $n \times m$ matrix where n is the number of subjects and m is the number of most influential features.
4. Use test set to compute the accuracy by using the model constructed in step 2.
5. Calculate p-values for all of the m SNPs.
6. Output m SNPs along with their p-values, and accuracy.

2.4.2 Algorithm 2

This algorithm [Please see Algorithm 2] employs random projections and feature selection algorithm together. The original dataset is trained with a training set to identify the best m SNPs. For each subject we keep only these SNPs. The modified dataset is projected onto k-dimensions for various values of k. For each value of k, we compute accuracy using the feature selection algorithm. We have also employed Principal Component Analysis (PCA) instead of Random Projection (RP) in Algorithm 2. The result is very interesting and intuitive. It is described in the results section. Details of the algorithm are given in Algorithm 2. At first, the algorithm constructs training set and test set by choosing data randomly from $Group_1$ and $Group_2$. $Group_1$ contains all the non-addicted subjects and $Group_2$ is the set of all addicted subjects. Training set consists of 50 percent of data from each of $Group_1$ and $Group_2$ (data is chosen randomly), respectively. The test set is formed using the other 50

percent from $Group_1$ and $Group_2$, respectively. FSA is then trained with the training set and it builds a model using linear SVM. The result is a $n \times m$ matrix where n is the number of subjects and m is most influential features. Through this set of features we can classify an unseen subject with certain accuracy. Random Projection (or PCA) is then applied onto these m features to reduce the feature space from m to k. Data normalization and data discretization are applied to this $n \times k$ matrix. The features and the accuracy are found with an invocation of Algorithm 1.

Algorithm 2 FSA with random projection

Input: $Group_1$, $Group_2$
Output: Best l SNPs and their p-values along with accuracy
begin

1. Construct training and test sets from $Group_1$ and $Group_2$.
2. Use the training set to train the feature selection algorithm and build the model of concept. Select the most significant m SNPs to represent the genotypes of the addicted subjects. Output of this stage is a $n \times m$ matrix where n is the number of subjects and m is the number of the most influential features.
3. **repeat**
4. Apply a random projection on the output of feature selection algorithm. In particular, project the data from m dimensions to k dimensions. Output of this step is a $n \times k$ matrix.
5. Apply data normalization (we use min-max normalization) on the $n \times k$ matrix.
6. Apply data discretization on the normalized $n \times k$ matrix.
7. Construct New_Group_1 and New_Group_2 from the $n \times k$ matrix and find the best l features and accuracy using Algorithm 1.
8. Calculate p-values for all the l features.
9. Output l features along with their p-values, and accuracy.

until *all the user chosen k dimensions are finished*;

2.4.3 Algorithm 3

In this algorithm [Please see Algorithm 3], we compare the accuracy and runtime of our Feature Selection Algorithm (FSA) and Multifactor Dimensionality Reduction (MDR) Algorithm. The FSA has been trained with training dataset and the algorithm comes up with a model which is applied to the test dataset to identify the best possible combination of SNPs with the highest accuracy. The MDR takes the dataset as a combination of two classes and returns a model with one or more combination of SNPs, accuracy, and CV consistency. Details of the algorithm are described in Algorithm 3.

Algorithm 3 Comparison of FSA and MDR

Input: $Group_1$, $Group_2$
Output: Best m SNPs with the corresponding accuracy
begin

1. Construct training and test sets from $Group_1$ and $Group_2$.
2. Use the training set to train the feature selection algorithm and build the model of concept. Select the most significant m SNPs to represent the genotypes of the addicted subjects. Output of this stage is a $n \times m$ matrix where n is the number of subjects and m is the number of the most influential features.
3. Output m SNPs along with the accuracy and time required to accomplish the task.
4. **repeat**
5. Run the MDR algorithm with time period, T.
6. Output SNPs along with the accuracy and time required to accomplish the task.

until *the user chosen time period* T *is over*;

2.5 Results and Discussion

We have done rigorous simulations to verify our proposed algorithms. These simulation results show that our algorithms indeed output significantly correct results which are illustrated next.

2.5.1 Algorithm 1

At first, we compute the p-values of each of the SNPs and sort them in increasing order of p-values [Please see Table 1]. After that we identify the best 32 SNPs using the feature selection algorithm and validate these SNPs with the top SNPs found in the previous step based on p-values. Here p-value calculation is based on logistic regression based test, and each p-value is calculated on a single SNP which is equivalent to a Chi-square test. In our feature selection algorithm we have employed linear SVM as well as some well-known kernels such as polynomial, GAUSSIAN RBF, and sigmoid to map the data from a space of low dimension to a space of high dimension [Please see Table 2, Table 3, Table 4, and Table 5].

Table 1 SNPs based on *p*-values

Rank	User defined SNP ID	*p*-value
1	X192	6.161326E-7
2	X592	3.907886E-4
3	X114	4.902156E-4
4	X483	6.466061E-4
5	X569	9.02912E-4
6	X253	0.001703205
7	X230	0.002096796
8	X1033	0.002481348
9	X275	0.00249018
10	X407	0.002598933

Table 2 Best 10 SNPs from the feature selection algorithm

Rank	User defined SNP ID	*p*-value
1	X114	4.902156E-4
2	X458	0.002744632
3	X961	0.01519576
4	X704	0.01878017
5	X519	0.03505115
6	X100	0.0374225
7	X989	0.03831268
8	X216	0.04014865
9	X365	0.04285033
10	X1100	0.04807944

A subset of SNPs is selected by employing linear SVM in the Feature Selection Algorithm.

Table 3 Best 10 SNPs from the feature selection algorithm

Rank	User defined SNP ID	*p*-value
1	X114	4.902156E-4
2	X458	0.002744632
3	X961	0.01519576
4	X704	0.01878017
5	X519	0.03505115
6	X100	0.0374225
7	X989	0.03831268
8	X216	0.04014865
9	X365	0.04285033
10	X1100	0.04807944

A subset of SNPs is selected by employing non-linear SVM in the Feature Selection Algorithm. Here we have used Polynomial Kernel to map the set of SNPs from a low dimension to a high dimension.

In the case of linear SVM, please note that the third best SNP (in terms of the p-value) was one of the SNPs that the feature selection algorithm has picked [Please see Table 1, and Table 2]. A simple calculation shows that if we pick 32 SNPs at random, the probability that one of them will be one of the three best SNPs (in terms of p-values) is 7.6%. This indicates that the feature selection algorithm is capable of identifying statistically significant SNPs.

Table 4 Best 10 SNPs from the feature selection algorithm

Rank	User defined SNP ID	p-value
1	X114	4.902156E-4
2	X483	6.466061E-4
3	X569	9.02912E-4
4	X1033	0.002481348
5	X407	0.002598933
6	X1120	0.002646448
7	X709	0.002852061
8	X1200	0.003385855
9	X702	0.003590515
10	X178	0.00382676

A subset of SNPs is selected by employing non-linear SVM in the Feature Selection Algorithm. Here we have used GAUSSIAN RBF Kernel to map the set of SNPs from a low dimension to a high dimension.

Table 5 Best 10 SNPs from the feature selection algorithm

Rank	User defined SNP ID	p-value
1	X114	4.902156E-4
2	X483	6.466061E-4
3	X569	9.02912E-4
4	X1033	0.002481348
5	X407	0.002598933
6	X1120	0.002646448
7	X709	0.002852061
8	X1200	0.003385855
9	X702	0.003590515
10	X178	0.00382676

A subset of SNPs is selected by employing non-linear SVM in the Feature Selection Algorithm. Here we have used Sigmoid Kernel to map the set of SNPs from a low dimension to a high dimension.

Also, the accuracy obtained is pretty good (73.805%) [Please see Table 6]. If we use the polynomial kernel by setting the parameter $p = 1$ [Please see Equation 3], the same subset of SNPs is picked and the maximum accuracy is also identical as in the case of linear SVM [Please see Table 2, Table 3, and Table 6].

In the case of GAUSSIAN RBF and sigmoid kernel, the best SNPs found by these kernels

Table 6 Comparison of time and maximum accuracy of different methods. Execution times are given in minutes.

Method name	Type	Maximum % accuracy	Elapsed time
FSA	Linear	73.805	5
FSA	Polynomial	73.805	0.17
FSA	GAUSSIAN RBF	45.698	0.15
FSA	Sigmoid	45.698	0.16
Random projection	FSA (Linear) + RP	73.805	–
PCA	FSA (Linear) + PCA	73.685	–
MDR	–	68.65	60

In this table "–" in the fourth column means much less time than for any other method.

included five of the best SNPs picked by simple p-value calculations [Please see Table 1, Table 4 and Table 5]. Here these kernels produce the same subset of SNPs and maximum accuracy [Please see Table 6]. Although by employing GAUSSIAN RBF and sigmoid the FSA is able to pick statistically significant genes compared to other methods described above, the accuracy obtained is very poor, i.e., 45.698% [Please see Table 6]. Please note that, we have chosen a large number of subsets of the SNPs and computed the quantities of interest for each such subset. The results are very similar.

2.5.2 Algorithm 2

The second algorithm employs random projections and feature selection together. At first, we take the best 32 SNPs given by the feature selection algorithm and apply random projection over these dataset containing those SNPs and project the data onto a space of 5, 10, 15, 20, 25, and 30 dimensions. FSA is then applied to these reduced dimension to classify the subjects of interest. For all of the reduced dimensions, we always get the maximum accuracy of 73.805%. This result indeed indicates that according to the Achlioptas [136] method the mapping of a set of points from a higher dimensional space to a lower dimensional space

closely preserves the pair-wise distances. Without any loss of generality, we can thus project the large dataset into a lower dimensional space and can get the same result.

We have also employed PCA instead of random projection in Algorithm 2 to compare the accuracy given by our techniques. The procedure is the same as described above. After applying FSA we pick the top 32 SNPs and apply PCA technique to find principal components of the feature space. The result is a list containing the coefficients defining each component (sometimes referred to as loadings), the principal component scores, etc. We then compute the 1^{st} principal component scores to 15^{th} principal component scores of each of the SNPs for each subject. After this data normalization and data discretization have been applied. FSA is then applied to the reduced dimensions of 10, and 15 respectively to classify the subjects of interest. The resulted maximum accuracy found was 73.685% [Please see Table 6]. Clearly, our random projection method beats PCA in term of accuracy. Here again we see that random projections in conjunction with feature selection are very effective in identifying statistically significant features of the input.

2.5.3 Algorithm 3

This approach validates the result of our feature selection algorithm that it indeed gives more accurate results than another well known algorithm called multifactor dimensionality reduction or MDR. MDR has been used to identify potential interacting loci in several phenotypes. MDR is a SVM-like gene-selection classifier algorithm. We have compared our gene selection algorithm with MDR in terms of accuracy and runtime. This comparison reveals that our algorithm outperforms MDR with respect to the time to calculate the best number of SNPs that can together serve as a classifier. We ran MDR with the time intervals of 10 minutes, 20 minutes, 30 minutes, and 60 minutes. The SNPs identified by our algorithms

form the best subset of SNPs which are also given by MDR after running for 10 minutes and above whereas our FSA takes only 5 minutes to find the best SNPs with an accuracy of 73.805% [Please see Table 6] by employing linear SVM. But if we use polynomial kernel, FSA takes only 0.17 minutes [Please see Table 6]. Here accuracy is the measure of how much confident we can be that the resulting SNPs can together serve as a classifier to distinguish two groups of subjects. Both programs were run on the same 2.8 GHz dual core machine.

Java implementation of MDR has been used for the analysis of 1212 SNPs. There are three types of search methods available for driving the MDR, namely, exhaustive, forced and random. For each attribute count specified, *Exhaustive Method* exhaustively examines each combination of attributes. This search method has no options. *Forced Method* examines only one attribute combination. The combination must be specified in the provided text field as a comma-separated list of attribute labels. The labels are case-sensitive. And at last, for each attribute count specified, *Random Method* examines random combinations. There are two options here, namely, evaluations and runtime. *Evaluation Option* evaluates a given number of random combinations, for each attribute count specified. For each attribute count specified, *Runtime Option* evaluates random combinations for a given amount of time. As the *Exhaustive Method* runs indefinitely for the pair-wise combination for the entire set of 1212 SNPs and the *Forced Method* is the totally irrelevant for our experiment, we used *Random Method* with the option of *Runtime*.

The best single-locus model identified was $X483$, with a training and testing accuracy of 56.61% and 49.75%, respectively but the cross-validation consistency was only 4 out of 10 after running for 5 minutes [Please see Table 7]. The best two-locus model identified was $X275$, and $X483$, with a training and testing accuracy of 61.09% and 55.61%, respectively and cross-validation consistency was 6 out of 10 [Please see Table 8]. After running for 15 minutes, MDR gave the best triple-locus model consisting of $X114$, $X216$, and $X1070$

with a training and testing accuracy of 64.07% and 59.37%, respectively [Please see Table 9]. The cross-validation consistency was 9 out of 10. On the contrary, our feature selection algorithm finds this combination after running for only 0.17 minutes with an accuracy of 73.085% without employing any randomness [Please see Table 6]. The ternary-locus model identified after running for 30 minutes was $X114$, $X315$, $X986$, and X1039 with a training and testing accuracy of 68.51% and 52.42%, respectively. The cross-validation consistency was 5 out of 10 [Please see Table 10]. After running for 60 minutes, MDR gave the best ternary-locus model consisting of $X114$, $X315$, $X986$, and $X1039$ with a training and testing accuracy of 68.65%, and 53.20%, respectively. But the cross-validation consistency was of only 4 out of 10 [Please see Table 11].

Table 7 MDR - Time duration: 5 minutes

Model	Training acc.	Testing acc.	CV cons.
X483	0.5661	0.4975	4/10
X275 X483	0.6104	0.5688	7/10
X93 X275 X407	0.6314	0.5642	6/10
X228 X243 X665 X733	0.6806	0.5014	6/10

Table 8 MDR - Time duration: 10 minutes

Model	Training acc.	Testing acc.	CV cons.
X483	0.5661	0.4975	4/10
X275 X483	0.6109	0.5561	6/10
X114 X216 X1070	0.6407	0.5937	9/10
X114 X315 X986 X1039	0.6842	0.5249	6/10

2.6 Conclusions

A subset of single nucleotide polymorphisms (SNPs) can be used to capture the majority of the information of genotype-phenotype association studies. The primary purpose of this

Table 9 MDR - Time duration: 15 minutes

Model	Training acc.	Testing acc.	CV cons.
X483	0.5661	0.4975	4/10
X275 X483	0.6109	0.5561	6/10
X114 X216 X1070	0.6407	0.5937	9/10
X114 X315 X986 X1039	0.6844	0.5133	6/10

Table 10 MDR - Time duration: 30 minutes

Model	Training acc.	Testing acc.	CV cons.
X483	0.5661	0.4975	4/10
X275 X483	0.6114	0.5555	5/10
X114 X216 X1070	0.6408	0.5810	8/10
X114 X315 X986 X1039	0.6851	0.5242	5/10

Table 11 MDR - Time duration: 60 minutes

Model	Training acc.	Testing acc.	CV cons.
X483	0.5661	0.4975	4/10
X275 X702	0.6125	0.5534	5/10
X114 X216 X1070	0.6409	0.5781	8/10
X114 X315 X986 X1039	0.6865	0.5320	4/10

research is to select a subset of SNPs while maximizing the power of detecting a significant association. From this point of view, we have proposed a number of approaches to find a subset of SNPs from the entire set to classify a set of individuals. Our proposed algorithms are indeed efficient, reliable, and scalable in terms of both accuracy and time complexity. Random projection has been used to project the data onto a lower dimensional space. A subset of attributes is then selected from this low dimensional space. To the best of our knowledge, random projection technique has not been employed before in the area of SNPs analysis. As revealed by our experimental results, these techniques offer the potential of high accuracies while keeping the run times low.

Part IV

Data Mining and Pattern Recognition

Chapter 1

Sequential and Parallel Clustering Algorithms

Conventional clustering algorithms suffer from poor scalability, especially when the data dimension is very large. It may take even days to cluster large datasets. For applications such as weather forecasting, time plays a crucial role and such run times are unacceptable. It is perfectly relevant to get even approximate clusters if we can do so within a short period of time. In this research work we propose a novel deterministic sampling technique that can be used to speed up any sequential or parallel clustering algorithm. We call this technique *DSC (Deterministic Sampling-based Clustering)*. As a case study we consider hierarchical clustering. Our empirical results show that DSC results in a speedup of more than an order of magnitude over exact hierarchical clustering algorithms when the data size is more than 6,000. Also, the accuracy obtained is excellent. In fact, on many datasets, **we get an accuracy that is better than that of exact hierarchical clustering algorithms!**. Even though we demonstrate the power of DSC only with respect to hierarchical clustering, DSC is a generic technique and can be employed in the context of any other clustering

technique (such as k-means, k-medians, etc.) as well. We also show how DSC can be employed in the case of parallel algorithms.

1.1 Introduction

The problem of clustering is to partition a given set of objects into groups (called *clusters*) such that objects in the same group are "similar" to each other. There are numerous ways of defining "similarity" and hence there exist many different versions of the clustering problem. Examples include hierarchical clustering, k-means clustering, k-medians clustering, etc. For each of these versions several efficient algorithms have been proposed in the literature. For example, the best known algorithm for hierarchical clustering takes $O(n^2)$ time on n objects (or points). We live in an era of data explosion and the value of n is typically very large. As a result, even a quadratic time algorithm may not be feasible in practice when the datasets are very large. Another factor that could add to the complexity of clustering is the data dimension. One possible way of speeding up these algorithms is with the employment of sampling. For instance, the CURE algorithm [167] employs random sampling in clustering and the speedups obtained are very good. In this research work we propose a novel deterministic sampling technique that can be used to speedup any clustering algorithm. To the best of our knowledge, deterministic sampling has not been utilized before in clustering.

Let I be a given set of n objects. It helps to assume that the objects are points in $\Re^d$ for some large d. In our technique there are several levels of deterministic sampling. In the first level the input is partitioned into several parts. Each part is clustered. Here one could employ any clustering algorithm. Some number of representatives (i.e., sample points) are chosen from each cluster of each part. These representatives move to the next level as

a deterministic sample. In general, at each level we have representatives coming from the prior level. These representatives are put together, partitioned, each part is clustered, and representatives from the clusters proceed to the next level. This process continues until the number of points (i.e., sample from the prior level) is 'small' enough. When this happens, these points are clustered into k clusters, where k is the target number of clusters. For each of these clusters, a center is identified. Finally, for each input point, we identify the closest center and this point is assigned to the corresponding cluster.

Clearly, the above technique can be employed in conjunction with any clustering algorithm. In this research work we consider hierarchical clustering as a case study. However, the technique is generic. We have tested our technique on many synthetic as well standard benchmark datasets. We achieve a speedup of more than an order of magnitude over exact hierarchical clustering when the data size is more than 6,000. Please note that real-life datasets have millions of points and more. Also, the accuracy obtained is very impressive. In fact, on many datasets, our accuracy is better than that of exact hierarchical clustering algorithms! We also show how to employ DSC to improve the performance of parallel clustering algorithms with hierarchical clustering as a case study.

1.2 Related Works

In this section (due to space constraints) we provide a very brief literature survey. Clustering algorithms fall under different categories such as partitioning methods, hierarchical methods, density-based methods, grid-based methods, and model-based methods. Partitioning based clustering divides the dataset into some user specified number of clusters using centroid or medoid based procedures. In the centroid based algorithms clusters are formed

using the center of gravity of the input points. Medoid based algorithms produce clusters accumulating points closest to the center of gravity. Some notable examples of centroid and medoid based algorithms can be found in [172], [171], [164], [174], and [185]. A hierarchical clustering algorithm partitions the entire dataset into a tree, known as the *dendrogram*, of clusters. Two kinds of hierarchical clustering, namely, agglomerative and divisive are known. Agglomerative methods form the clusters in a bottom-up fashion where each data point starts as a single cluster and these clusters get progressively merged until all the input points form a single cluster. Divisive approach works in a top-down fashion starting with a single cluster containing all the input points. This cluster gets partitioned into smaller and smaller clusters until each input point forms a single cluster. Some of the well known hierarchical clustering algorithms are: Balanced Iterative Reducing and Clustering using Hierarchies BIRCH [186], Clustering Using REpresentatives CURE [167] and CHAMELEON [173]. As mentioned before, CURE employs random sampling.

Density based clustering partitions the dataset into a number of groups based on the density of the input points in a region. Examples include DBSCAN [166] and DENCLUE [169]. Grid-based clustering algorithms run in two steps. In the first step they map the entire dataset into a finite number of hyper-rectangular cells and in the next step they perform some statistical procedures on the transformed or mapped points to find the density of the cells. Adjacent cells are connected to form a single cluster if those cells follow same density distribution. Example density-based clustering algorithms are STatistical INformation Grid-based method STING [184], WaveCluster [181], and CLustering In QUEst CLIQUE [161].

The above algorithms do not assume any hypothesis/model about the data and fall under the category of *exploratory* algorithms. *Confirmatory* or *inferential* algorithms assume a hypothesis/model on the data to be clustered. A number of statistical inferential techniques can be found in the literature. Examples include linear regression, discriminant analysis,

multi-dimensional scaling, factor analysis, principal component analysis, and so on. A survey on inferential clustering can be found in [183]. Some other interesting clustering techniques and algorithms can be found in [163], [175], [162], [182], and [180].

1.3 Background Information

1.3.1 Agglomerative hierarchical clustering

Algorithm 1.1: Agglomerative Hierarchical Clustering

Input: A set of n data points and an integer k
Output: The best k clusters
begin

1. Start with n clusters (nodes) labeled $1, 2, 3, \ldots, n$, where each cluster has one input point.
2. Calculate all pair-wise cluster distances and place them in a $n \times n$ matrix. This matrix is called the dissimilarity matrix.
3. Find the pair of nodes (i.e., clusters) with the minimum cluster distance.
4. Join these two nodes into a new node and remove the two old nodes. Relabel the nodes with consecutive integers.
5. Update the dissimilarity matrix.
6. Repeat steps 3 through 5 until only k clusters are left. Output these k clusters.

Steps involved in any agglomerative clustering procedure are shown in Algorithm 1.1. The distance between two clusters can be defined in a number of ways and accordingly different versions of the hierarchical clustering problem can be obtained. We define below some of these distances. For any two clusters I and J, let $d(I, J)$ stands for the distance between I and J. In step 3 of Algorithm 1.1, let the clusters with the minimum distance be I and J. Also, let the merged cluster in step 4 be Q. For any cluster I, $|I|$ denotes the size

of the cluster I. In the following definitions, L refers to any cluster other than I, J, and Q.

1. Single-link: $d(Q, L) = \min\{d(I, L), d(J, L)\}$. The distance between two clusters A and B is the closest distance between a point in A and a point in B: $d(A, B) = \min_{a \in A, b \in B} d(a, b)$.

2. Complete-link: $d(Q, L) = \max\{d(I, L), d(J, L)\}$. The distance between two clusters A and B is the maximal distance between a point in A and a point in B: $d(A, B) = \max_{a \in A, b \in B} d(a, b)$.

3. Average-link: $d(Q, L) = \frac{|I| \cdot d(I,L) + |J| \cdot d(J,L)}{|I| + |J|}$. The distance between two clusters A and B is the average distance between a point in A and a point in B: $d(A, B) = \frac{1}{|A||B|} \sum_{a \in A, b \in B} d(a, b)$.

4. Centroid-link: $d(K, L) = \frac{|I| \cdot d(I,L) + |J| \cdot d(J,L)}{|I| + |J|} - \frac{|I| \cdot |J| \cdot |d(I,J)|}{(|I| + |J|)^2}$. Here $d(A, B)$ is the distance between the centroids of the clusters in Euclidean space: $d(A, B) = (\parallel \vec{c}_A - \vec{c}_B \parallel)^2$, where $\vec{c}_A$ denotes the centroid of the points in cluster A.

5. Ward-link: $d(K, L) = \frac{(|I| + |L|) \cdot d(I,L) + (|J| + |L|) \cdot d(J,L) - |L| \cdot d(I,J)}{|I| + |J| + |L|}$. Here $d(A, B) = \frac{2|A||B|}{|A| + |B|} \cdot (\parallel \vec{c}_A - \vec{c}_B \parallel)^2$ where $\vec{c}_A$ denotes the centroid of the points in cluster A.

1.3.2 Sampling

The idea of sampling is to pick a subset of the given input, process the subset, and make inferences on the original dataset. Sampling has played a major role in the design of efficient algorithms for numerous fundamental problems in computing such as sorting, selection, convex hull, clustering, rules mining, etc. Both sequential and parallel algorithms have benefited. Random sampling perhaps is the most popular. Deterministic sampling has also been employed for such problems as selection. For a survey on the role of random sampling in knowledge discovery, the reader is referred to [176]. In agglomerative hierarchical clustering, random sampling is exploited in ROCK [168] and CURE [167]. These algorithms randomly choose a subset of the input points and cluster this subset. Each of the other input points

is assigned to the cluster closest to it.

To the best of our knowledge, deterministic sampling has not been employed in the context of clustering before. A major advantage of deterministic sampling over randomized sampling lies in the fact that the analyses done using deterministic sampling always hold. In this research work we propose a technique called DSC (Deterministic Sampling-based Clustering). DSC is based on the scheme proposed in [177]. The scheme of [177] works in the context of out-of-core selection. The problem of selection is to find the i^{th} smallest key from a collection X of n keys. The selection algorithm of [177] works as follows: At the beginning all the keys are considered as live keys. The algorithm then goes through stages of sampling. In the first stage, it divides the collection X into a number of parts such that each part contains M keys, where M is the size of the memory. Each part is then sorted and keys that are at a distance of $\sqrt{M}$ from each other are retained. So, the ranks of the retained keys are $(\sqrt{M}, 2\sqrt{M}, 3\sqrt{M}, \ldots)$. Clearly, the number of keys in the retained set R_1 from the first stage is $= \frac{n}{\sqrt{M}}$. In the next stage, the algorithm again groups the elements of R_1 such that there are M elements in each part, sorts each part, collects only every $\sqrt{M}^{th}$ element in each part. Let the set retained in the second stage be R_2. This process of selecting a subset from one level as a sample to the next level continues until only $\leq M$ elements are left. These elements constitute a deterministic sample from which two elements ℓ_1 and ℓ_2 are picked such that these elements bracket the i^{th} smallest element of X. Followed by this, we eliminate all the keys of X that do not have a value in the interval $[\ell_1, \ell_2]$. This process of sampling and elimination is continued until the number of keys left is small. At that point, the remaining elements are sorted and the element of interest is identified.

1.3.3 Clustering accuracy

Given a clustering algorithm, there are multiple ways to measure the accuracy of clustering. In this research work we use a measure that is very intuitive and has been mentioned in many prior works (see e.g., [167]). Given k clusters corresponding to a given input point set, we first identify the center of each cluster. Then we calculate the distance of each point to the center of the cluster it belongs to. This distance is summed over all the points. If d_{ij} is the distance of the i^{th} point in cluster j to the corresponding center, the clustering accuracy is computed as $\sum_i \sum_j d_{ij}$. Let the clustering accuracy of DSC and any other exact hierarchical clustering algorithm be CA_A and CA_E, respectively. Then, we define the clustering efficiency of our algorithm as $\frac{CA_E}{CA_A} \times 100\%$.

1.4 Our Algorithm

There are several levels of sampling in our technique. The number of points that move from one level as a sample to the next level progressively decreases. When the number of points in some level falls below some threshold for the first time, we cluster those points into k clusters. We identify the centers of these clusters. Each input point u is then assigned to the cluster whose center is closest to the point u.

Let the number of levels in the algorithm be r (i.e., in stage r the number of remaining points falls below a threshold for the first time). In the first stage we have all the n input points. We partition the input set into p_1 parts of equal size. Each part (of size $\frac{n}{p_1}$) is clustered into q clusters using any clustering algorithm. We pick ℓ representatives from each such cluster and these representatives from each cluster of each part move to the second level as a sample. The number of points that move to the second level is $p_1 q \ell$. In the

second stage all of these points are put together, partitioned into p_2 parts of equal size, each part is clustered into q clusters, ℓ representatives are chosen from each cluster, and these representatives move to the third level; and so on.

In general, in level i we partition the points into p_i equal parts, cluster each part into q clusters, pick ℓ points from each cluster, and the picked points move to level $i+1$, for $1 \leq i \leq (r-1)$. A pseudocode for DSC is supplied in Algorithm 1.2. In this pseudocode we have assumed (for simplicity) that $p_1 = p_2 = \cdots = p_r = p$. Also, the parameters q and ℓ have to be chosen to optimize run time and accuracy. In our implementation we have used the following values: $q = k$ and $\ell = 1$.

Algorithm 1.2: Deterministic Sampling-based Clustering (DSC)

Input: A set of n data points; integers p, k, and r.
Output: The best k clusters
begin

1. Divide the data points into p equal sized parts.
2. Cluster each part into q clusters.
3. Deterministically select ℓ representatives from each of the above clusters.
4. Put all of the representatives together.
5. Repeat r times steps 1 through 4.
6. Cluster the remaining points into k clusters and find the center of each of these final clusters.
7. Assign each input point x to that cluster whose center (from among all the cluster centers) is the closest to x.

1.5 Analysis

1.5.1 Time complexity

Let there be r levels of sampling in the algorithm. Note that the standard hierarchical clustering on n points can be done in $O(n^2)$ time. Let the number of parts in level i be p_i, for $1 \leq i \leq r$. In each level and each part assume that there are q clusters and from each cluster we pick ℓ representatives.

In level 1 there are p_1 parts and a total of n points and hence each part has $\frac{n}{p_1}$ points. To cluster each part we spend $O((n/p_1)^2)$ time and hence the total time spent in level 1 is $O\left(\frac{n^2}{p_1}\right)$. From each part of level 1, we pick $q\ell$ representatives and hence the total number of points that move onto level 2 is $p_1 q\ell$.

There are p_2 parts in level 2 and each part has $\frac{p_1 q\ell}{p_2}$ points. As a result, the total time spent in level 2 is $O\left(\frac{(p_1 q\ell)^2}{p_2}\right)$. Proceeding in a similar manner, the total number of points in level j (for $1 \leq j \leq r$) is $p_{j-1} q\ell$ and there are p_j parts in this level. Therefore, the total time spent in level j is $O\left(\frac{(p_{j-1} q\ell)^2}{p_j}\right)$.

Putting together, the total time spent in all the r levels of sampling is $O\left(\frac{n^2}{p_1} + \frac{(p_1 q\ell)^2}{p_2} + \cdots + \frac{(p_{r-1} q\ell)^2}{p_r}\right)$. We can easily ensure that the number of points from one level to the next decreases by at least a constant factor. For example, consider the case where $\ell = c_1$ and $p_i = \frac{n}{2^{i-1} c_2 q}$, for some constants c_1 and c_2 (greater than zero) with $c_2 > c_1$ and for $1 \leq i \leq r$. In this case, this run time simplifies to $O\left(\frac{n^2}{p_1}\right)$. Also, at level r of sampling, we identify k cluster centers and every other point is assigned to one of these clusters based on which of these centers is closest to that point. The total time for this is $O(nk)$. In summary, the total run time of the algorithm is $O\left(\frac{n^2}{p_1} + nk\right)$.

Note: In Algorithm 2 we have assumed that $\ell = c_1$ and $p_i = \frac{n}{2^{i-1} c_2 q}$, for some constants c_1

and c_2 (greater than zero) with $c_2 > c_1$ and for $1 \leq i \leq r$.

1.5.2 Accuracy

The accuracy of any (randomized or deterministic) sampling based clustering algorithm can be established by verifying that the final clusters of interest are well represented in the sample. In particular, if $C_1, C_2, \ldots, C_k$ are the clusters present in the input dataset then there should be enough representation from each of the clusters in the sample. This verification seems to be very intuitive as has been pointed out in the CURE paper [167]. We can use this observation to verify the validity of our algorithm. For simplicity we consider only one level of sampling. Specifically, we partition the input into p equal parts, cluster each part, select representatives from each cluster of each part, put together all the representatives and cluster them into k clusters, find the centers of these clusters, and assign each input point to the cluster whose center is the closest. The analysis can be extended to multiple levels as well.

Average case analysis

Now we show that if the sample size is large enough then each final cluster will have enough representation in the sample with high probability. Please note that our algorithm employs deterministic sampling and the probability we refer to is computed in the space of all possible inputs. In other words, the verification corresponds to the average case performance of the algorithm. Before we present the verification we state the well-known Chernoff bounds.

Chernoff Bounds: If a random vraiable X is the sum of n iid Bernoulli trials with a success probability of p in each trial, the following equations give us concentration bounds of deviation of X from the expected value of np. X is said to be binomially distributed

and this distribution is denoted as $B(n,p)$. By *high probability* we mean a probability that is $\geq (1-n^{-\alpha})$ where α is the probability parameter and is typically a constant ≥ 1. The first equation is more useful for large deviations whereas the other two are useful for small deviations from a large expected value.

$$Pr(X \geq m) \leq (np/m)^m e^{m-np} \tag{1.5.1}$$

$$Pr(X \leq (1-\epsilon)np) \leq exp(-\epsilon^2 np/2) \tag{1.5.2}$$

$$Pr(X \geq (1+\epsilon)np) \leq exp(-\epsilon^2 np/3) \tag{1.5.3}$$

for all $0 < \epsilon < 1$.

Accuracy Verification: Let I be a given set of n points to be clustered and let the final list of clusters be $C_1, C_2, \ldots, C_k$. Let the number of points in C_i be n_i, for $1 \leq i \leq k$. Consider a simple algorithm where we deterministically pick a sample, cluster the sample into k clusters, and assign every input point x to the cluster whose center (from among all the cluster centers) is the closest to x. Let S be a deterministic sample of size m from I. Also, let X_i be the number of points of C_i in S, for $1 \leq i \leq k$. If we assume that each input permutation is equally likely, then X_i is binomially distributed as $B\left(m, \frac{n_i}{n}\right)$. $E[X_i] = \frac{mn_i}{n}$. Using Chernoff bounds equation 1.5.2, $Pr\left[X_i \leq (1-\epsilon)\frac{mn_i}{n}\right] \leq \exp\left(\frac{-\epsilon^2 mn_i}{2n}\right)$. For $\epsilon = 1/2$, $Pr\left[X_i \leq \frac{mn_i}{2n}\right] \leq \exp\left(\frac{-mn_i}{8n}\right)$. This probability will be $\leq n^{-\alpha}$ when $m \geq \frac{8\alpha n \log_e n}{n_i}$. For this value of m, $Pr[X_i \leq 4\alpha \log_e n] \leq n^{-\alpha}$. Let $n_{min} = \min_{i=1}^{k} n_i$. In summary, when $m \geq \frac{c\alpha n \log_e n}{n_{min}}$ (for some constant $c > 8$), every C_i will have a good representation in S and hence the clusters output will be correct. For example, if $n_{min} = \sqrt{n}$, then it suffices for m to be $\geq 8\alpha\sqrt{n}\log_e n$.

1.6 Simulation Environment

We have evaluated the performance of DSC via rigorous simulations. Both run time and accuracy are considered. Synthetic as well as standard benchmark datasets have been employed for testing. A description of these datasets follows.

We have generated synthetic datasets based on both uniform and skewed distributions. All the datasets are from a high-dimensional space. Our datasets based on uniform distribution are generated by picking each coordinate value of each point uniformly randomly from the interval [0, 200]. To generate skew-distributed datasets, at first we generated datasets part-by-part and finally put them together. For example, a skew-distributed dataset could consist of 10 parts where each part is uniformly distributed in a different interval. The following tables [Please, see Table 1.6.1 and Table 1.6.2] describe our synthetic and benchmark datasets. Benchmark datasets have been downloaded from [165].

Table 1.6.1: Synthetic Datasets

Name	Size	Attributes	Distribution
U1	11k	200	Uniform
U2	12k	200	Uniform
U3	13k	200	Uniform
U4	14k	200	Uniform
U5	15k	200	Uniform
S1	11k	200	Skewed
S2	12k	200	Skewed
S3	13k	200	Skewed
S4	14k	200	Skewed
S5	15k	200	Skewed

Table 1.6.2: Benchmark Datasets

Name	Size	Attributes	# of clusters
Thyroid	215	5	2
Wine	178	13	3
Yeast	1484	8	10
Aggregation	788	2	7
D31	3100	2	31
Flame	240	2	2
R15	600	2	15
Pathbased	300	2	3

1.7 Simulation Results

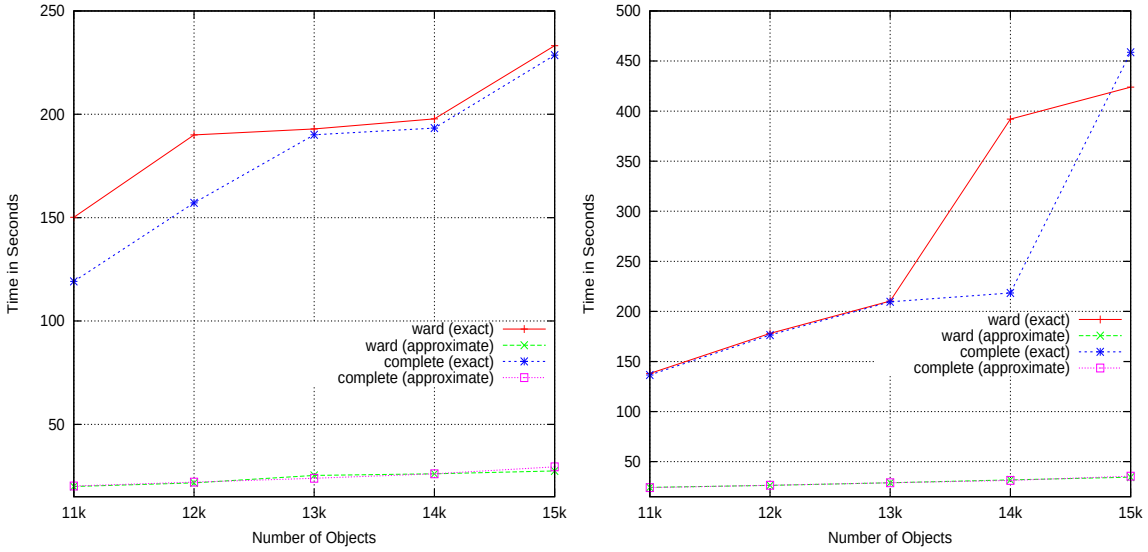
In this section we present our experimental results. All the programs have been run on an Intel Core i5 2.3GHz machine with 4GB of RAM.

1.7.1 Synthetic datasets

In our implementation of DSC we have employed one level of sampling. In particular, we partitioned the entire dataset into groups of size 500 each and clustered each group into 10 clusters. From each such cluster we picked one representative. These representatives were then put together and clustered into 10 clusters. Finally, each input point was assigned to the cluster with the closest center.

We have computed the clustering efficiencies as the average over all the synthetic datasets (uniformly distributed and skew-distributed, respectively). Also, we have applied different clustering methods [Please see Table 1.7.1] such as *ward*, *complete*, and *average*. In the case of skew-distributed datasets the average clustering efficiencies found by applying different clustering methods are in the range of [99%, 101%]. On the other hand, the efficiency exceeds 100% on uniform-distributed datasets. As the size of the datasets and/or the dimension increases, our algorithm outperforms exact algorithms, in terms of run time, by more than an order of magnitude [Please see Figure 1.7.1]. To demonstrate scalability of our proposed algorithm we have generated 5 big datasets occupying a large collection of objects. Each object consists of 2 attributes. We could not able to run the exact algorithms on these big datasets using our Intel Core i5 2.3GHz machine with 4GB of RAM. On the contrary our proposed algorithm is able to perform clustering on these datasets without stalling the machine [Please see Figure 1.7.2].

We have also compared our algorithm with other well known clustering algorithms such



(A) Uniform data having 200 attributes (B) Skew data having 200 attributes

FIGURE 1.7.1: Time to find 10 clusters from each dataset by applying various exact and approximate methods.

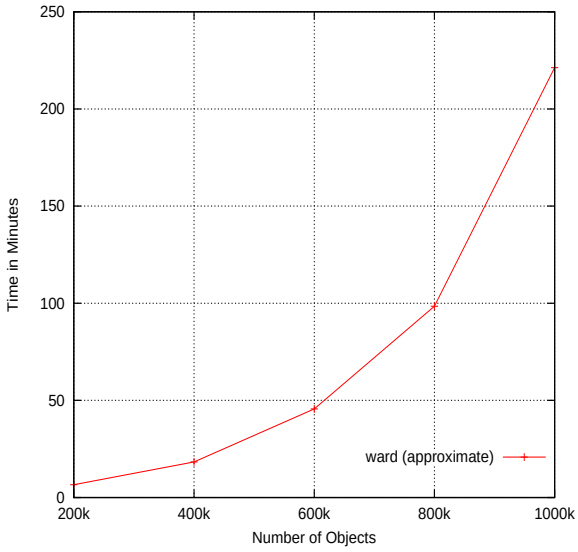


FIGURE 1.7.2: Time to find 10 clusters from each of the big datasets by applying ward (approximate) method.

as k-means, Partitioning Around Medoids (PAM), and Density-Based Spatial Clustering of Applications with Noise (DBSCAN). The datasets considered here are generated by picking

TABLE 1.7.1: Efficiency - Synthetic Datasets

Clustering method	Uniformly distributed datasets	Skew-distributed datasets
ward	100.42%	100.46%
complete	100.45%	100.22%
average	100.00%	99.26%

each coordinate value of each point uniformly randomly from the interval [0, 200]. Each point consists of 200 attributes and each of the algorithms is tuned to find 10 clusters. As the size of the datasets and/or the dimension increases, our algorithm outperforms all of the aforementioned algorithms, in terms of run time, by more than an order of magnitude and the average clustering efficiencies found by applying different clustering methods are in the range of [99%, 100%] [Please see Table 2.6.4].

1.7.2 Benchmark datasets

We have computed clustering efficiency separately for each of the benchmark datasets. For each dataset we generated the same number of clusters as shown in Table 1.6.2. A comparison has been made with various exact algorithms such as *ward*, *complete*, and *average*. From the results [Please see Table 1.7.3], we infer that the accuracy obtained by DSC is very competitive with those of exact algorithms. Also, for D31 dataset DSC is around 4 times faster than exact algorithms. When the data size is less than one thousand DSC is not faster than exact algorithms.

Table 1.7.2: A Comparison

		k-means	**ward**		**complete**		**average**	
Objects	Dim	Time (s)	Efficiency	Time (s)	Efficiency	Time (s)	Efficiency	Time (s)
10k	200	23.49	99.70%	18.59	99.74%	18.27	99.13%	17.93
20k		65.46	99.79%	36.04	99.76%	36.64	99.19%	36.02
50k		235.74	99.87%	94.98	99.87%	98.48	99.23%	97.16
100k		612.85	99.91%	218.96	99.91%	227.33	99.28%	237.47
		PAM	**ward**		**complete**		**average**	
Objects	Dim	Time (s)	Efficiency	Time (s)	Efficiency	Time (s)	Efficiency	Time (s)
10k	200	121.75	99.96%	18.59	100.00%	18.27	99.39%	17.93
		DBSCAN	**ward**		**complete**		**average**	
Objects	Dim	Time (s)	Efficiency	Time (s)	Efficiency	Time (s)	Efficiency	Time (s)
10k	200	359.62	100.00%	22.58	100.00%	22.16	100.00%	21.88

TABLE 1.7.3: Efficiency - Benchmark Datasets

Method	Thyroid	Wine	Yeast	Aggregation	D31	Flame	R15	Pathbased
ward	100.92%	98.80%	103.80%	96.89%	92.32%	97.26%	90.16%	98.71%
complete	101.72%	99.63%	116.51%	95.76%	86.79%	110.62%	81.62%	99.83%
average	101.41%	99.99%	122.12%	95.76%	94.36%	95.78%	91.21%	101.88%

1.8 Parallel Algorithms

When the data size is very large, parallelism becomes inevitable. We live in an era of data explosion. Given that most of the clustering algorithms available in the literature take at least quadratic time, it becomes essential for us to develop parallel clustering algorithms. In this section we show how to parallelize our deterministic sampling based clustering technique.

Several parallel models of computing have been introduced in the literature. These models differ in the inter-processor communication mechanism. We can broadly categorize parallel models into two, namely, fixed connection machines and shared memory models (also known as Parallel Random Access Machines (PRAMs)). A fixed connection machine can be represented as a directed graph where each node corresponds to a processing elements and the edges correspond to communication links. Examples include the mesh, the hypercube, butterfly, etc. On the other hand, in a PRAM communication happens by writing into and reading from common memory cells. Several variants of a PRAM can also be conceived of depending on how read and write conflicts are resolved. A model that does not permit concurrent reads or writes is called the Exclusive Read Exclusive Write (EREW) PRAM. A variant where concurrent reads are permitted and concurrent write are prohibited is called the Concurrent Read Exclusive Write (CREW) PRAM. In a Concurrent Read Concurrent Write (CRCW) PRAM, both concurrent reads and concurrent writes are permitted. There are several versions of a CRCW PRAM depending on how write conflicts are resolved. Examples include the Common CRCW PRAM (where a concurrent write is permitted only if all the conflicting processors have the same message to write), Arbitrary CRCW PRAM (in which when there is a write conflict one of the conflicting messages is written), Priority CRCW PRAM (where write conflicts are resolved based on a priority), etc. More details on parallel models can be found in any text on algorithms (see e.g., [170]).

Before we describe how to parallelize the sampling technique we briefly summarize some previous parallel algorithms for hierarchical clustering.

1.8.1 Parallel hierarchical clustering

Parallel algorithms for hierarchical clustering have been given in [178] for several models of computing. We state some of them in this section.

1. There exists a parallel algorithm for hierarchical clustering on n points that takes $O(\log n)$ time using $\frac{n^2}{\log n}$ CRCW PRAM processors. This algorithm is randomized and the run time bound holds with a high probability. The basic idea behind this algorithm is to reduce the clustering problem to those of finding a minimum spanning tree in a complete graph and finding connected components in a forest.

2. Hierarchical clustering on n points can be done on an array with reconfigurable optical buses (AROB) in $O(\log^2 n)$ cycles using n^2 processors. AROB is a fixed connection machine and a cycle can be thought of as $O(1)$ time.

3. If the input points are from an Euclidean space of constant dimension, then hiearchical clustering can be done in an expected $O(\log n)$ time using n CRCW PRAM processors. Here the expectation is over the space of all possible inputs. The same algorithm runs in $O(\log^2 n)$ cycles on a $1 \times n$ AROB.

1.8.2 New parallel algorithms

In this section we present parallelizations of our sampling based clustering paradigm. Recall that in our sampling strategy, there are several levels of sampling. The number of points that move from one level as a sample to the next level progressively decreases. When the

number of points in some level falls below some threshold for the first time, we cluster those points into k clusters. We identify the centers of these clusters. Each input point u is then assigned to the cluster whose center is closest to the point u.

Let the number of levels be r. Points in level i are partitioned into p_i parts. Each part is clustered into q parts and ℓ representatives are picked from each part. These representatives are the points that move on to the next level. For simplicity consider the case where $\ell = c_1$ and $p_i = \frac{n}{2^{i-1}c_2 q}$, for some constants c_1 and c_2 (greater than zero) with $c_2 > c_1$ and for $1 \leq i \leq r$. Note that with these choices, a constant fraction of points move from one level to the next.

Consider the CRCW PRAM model. Consider the case where the number of processors $P = p_1$. The following algorithm implements our sampling strategy.

1. In the first level we have p_1 parts. Each part can be assigned to a single processor. Each processor then performs a clustering using the algorithm in [178]. Specifically, each part is clusterd into q clusters.

2. From each cluster of each part ℓ representatives are picked. These representatives move to the next level. These representatives can be written into successive cells of common memory without any conflicts.

3. The above two steps are repeated until the number of points remaining is less than a threshold.

4. The remaining points are clustered into k clusters and the center of each cluster is found. Each such center is given a unique integer as its ID.

5. For each input point we figure out the closest center and associate it with this center.

6. We sort the input points based on their center IDs. This completes the clustering task.

Theorem 1.8.1. The above algorithm runs in a randomized $O(\log^2 n)$ time using $\frac{n^2}{p_1 \log n} + \frac{nk}{\log^2 n}$ CRCW PRAM processors. If $q = \Theta(k)$, then the number of processors needed is $O\left(\frac{nk}{\log n}\right)$.

Proof: The size of each part in level 1 is $\frac{n}{p_1}$. Thus clustering of each part can be done in $O\left(\log \frac{n}{p_1}\right) = O(\log n)$ time using $\frac{n^2}{p_1^2 \log n}$ CRCW PRAM processors using the algorithm in [178]. Thus clustering in step 1 can be completed in $O(\log n)$ time given $\frac{n^2}{p_1 \log n}$ CRCW PRAM processors. In step 2, the representatives can be picked in $O(1)$ time.

Note that the points that move from one level to the next is at most a constant fraction. In the worst case there could be $\Theta(\log n)$ levels. Clearly, the time spent in each level is $O(\log n)$ and hence the total time spent in steps 1 and 2 across all the levels is $O(\log^2 n)$. We do have enough processors to complete the processing at each level in $O(\log n)$ time.

We could terminate the process of sampling and representative selection when the number of remaining points is $\frac{n}{\sqrt{p_1}}$ or less. In this case, step 3 will take $O(\log n)$ time, using $\frac{n^2}{p_1 \log n}$ CRCW PRAM processors.

To perform step 5, we can compute the distance between each input point and each of the k cluster centers. This can be done in $O(1)$ time using nk processors. Thus this step can also be done in $O(\log^2 n)$ time using $\frac{nk}{\log^2 n}$ processors (with an application of the slow-down lemma).

To output the clusters, we can sort the input points on their center IDs. Note that each ID is an integer in the range $[1, k]$. Therefore, we can apply the integer sorting algorithm of [179]. In this case, the run time will be $O(\log n)$ with high probability using $\frac{n}{\log n}$ CRCW PRAM processors.

Put together, the total run time of the algorithm is $O(\log^2 n)$ and the processor bound

is $\frac{n^2}{p_1 \log n} + \frac{nk}{\log^2 n}$.

Clearly, when $q = \Theta(k)$, the value of p_1 becomes $\Theta\left(\frac{n}{k}\right)$. In this case, the processor bound becomes $O\left(\frac{nk}{\log n}\right)$. □

In practice, we have to typically perform only a constant number of levels of sampling. In fact in all the experimental results we have reported in this research work we have used only one level of sampling. Even this has yielded very good results. We can specialize Theorem 1.8.1 for this case and get the following theorem.

Theorem 1.8.2. When the number of levels of sampling is $O(1)$, our parallel algorithm runs in $O(\log n)$ time using $\frac{n^2}{p_1 \log n} + \frac{nk}{\log n}$ CRCW PRAM processors. When $q = \Theta(k)$, this processor bound becomes $O\left(\frac{nk}{\log n}\right)$. □

Our deterministic sampling strategy can also be applied to parallel clustering algorithms on other models of computing. For example, we could obtain the following theorem for the AROB model by parallelizing the algorithms given in [178].

Theorem 1.8.3. Hierarchical clustering on n points can be done in $O(\log^3 n)$ cycles using $\frac{n^2}{p_1} + nk$ AROB processors. If the number of levels of sampling is $O(1)$, then the run time can be reduced to $O(\log^2 n)$ cycles keeping the processor bound the same. When $q = \Theta(k)$, then the processor bound becomes $O(nk)$.

1.9 Conclusions

Sampling is a powerful technique that has been applied to solve many fundamental problems of computing efficiently. Random sampling is much more popular than deterministic sampling. In the context of clustering, random sampling has been successfully applied in a number of algorithms such as CURE. To the best of our knowledge, deterministic sampling

has not been employed before for designing clustering algorithms. In this research work we have presented a novel deterministic sampling technique called DSC that can be used to speedup any clustering algorithm. As a case study, we have demonstrated the power of DSC in the context of hierarchical clustering. We have tested the performance of DSC on both synthetic and benchmark datasets. DSC achieves impressive accuracies. On many datasets, DSC achieves better accuracies than exact algorithms! Moreover, the speedups obtained are equally impressive. In particular, DSC is faster than exact algorithms by more than an order of magnitude.

We have also presented parallel clustering algorithms that employ our deterministic sampling paradigm. We thus feel that DSC is a very effective sampling technique. Please note that deterministic sampling is preferable over random sampling since the analysis done using deterministic sampling will always hold (instead of holding on an average or with high probability).

Chapter 2

Novel Randomized Feature Selection Algorithms

Feature selection is the problem of identifying a subset of the most relevant features in the context of model construction. This problem has been well studied and plays a vital role in machine learning. In this research work we present three randomized algorithms for feature selection. They are generic in nature and can be applied for any learning algorithm. Proposed algorithms can be thought of as a random walk in the space of all possible subsets of the features. We demonstrate the generality of our approaches using three different applications. The simulation results show that our feature selection algorithms outperforms some of the best known algorithms existing in the current literature.

2.1 Introduction

Feature Selection is defined as the process of selecting a subset of the most relevant features from a set of features. It involves discarding irrelevant, redundant and noisy features. Feature

selection is also known as variable selection, attribute selection or variable subset selection in the fields of machine learning and statistics. The concept of feature selection is different from feature extraction. Feature extraction creates new features from the set of original features by employing a variety of methods such as linear combinations of features, projection of features from the original space into a transformed space, etc. We can summarize the usefulness of feature selection as follows: (1) Shorter training times: When irrelevant and redundant features are eliminated, the learning time decreases; (2) Improved model creation: The model built is more accurate and efficient; and (3) Enhanced generalization: It produces simpler and more generalized models.

A generic feature selection algorithm employs the following steps: (1) Select a subset of features; (2) Evaluate the selected subset; and (3) If the stopping condition is met then terminate else repeat steps 1 and 2. The algorithm generates candidate subsets using different searching strategies depending on the application. Each of the candidate subsets is then evaluated based on an objective function. In the context of any learning algorithm, the objective function could be the accuracy. Note that for any learning algorithm there are two phases. In the first phase (known as the *training phase*), the learner is trained with a set of known examples. In the second phase (known as the *test phase*), the algorithm is tested on unknown examples. Accuracy refers to the fraction of test examples on which the learner is able to give correct answers. In the feature selection algorithm, if the current subset of features yields a better value for the objective function, the previous best solution is replaced with the current one. If not, the next candidate is generated. This process iterates over the search space until a stopping condition is satisfied. Finally, the best subset is validated by incorporating prior knowledge.

In this research work we introduce a variety of randomized techniques for feature selection. These techniques can be used in the context of any learning algorithm. Consider the space of

all possible subsets of features. We start with a random subset s of the features and calculate its accuracy. We then choose a *random neighbor* s' of s and compute its accuracy. If the accuracy of s' is greater than that of s, we move to the new subset s' and proceed with the search from this point. Otherwise, we proceed our search depending on different techniques proposed in our algorithms. Suppose if the accuracy of s' is smaller than that of s, we can stay with the subset s (with some probability p) or move to the subset s' with probability $1-p$. We proceed with the search from the point we end up with. This process of searching the space is continued until no significant improvement in the accuracy can be obtained. Our randomized search techniques are generic in nature. We have employed it on three different applications and found that those are indeed scalable, reliable and efficient. Note that our algorithms resembles many local searching algorithms (such as Simulated Annealing (SA)). However, our algorithms are much simpler and differs from the others. For example, we do not employ the notion of *temperature* that SA utilizes.

2.2 Related Works

In this section we provide a summary of some well-known feature selection algorithms. These algorithms differ in the way the candidate subsets are generated and in the evaluation criterion used. Some examples of feature extraction methods can be found in [217].

2.2.1 Selection of candidate subset

Subset selection begins with an initial subset that could be empty, the entire set of features, or some randomly chosen features. This initial subset can be changed in a number of ways. In forward selection strategy, features are added one at a time. In backward selection the least

important feature is removed based on some evaluation criterion. Random search strategy randomly adds or removes features to avoid being trapped in a local maximum. If the total number of features is n, the total number of candidate subsets is 2^n. An exhaustive search strategy searches through all the 2^n feature subsets to find an optimal one. Clearly, this may not be feasible in practice [200]. A number of heuristic search strategies have been introduced to overcome this problem. The branch and bound method [207] exploits exhaustive search by maintaining and traversing a tree, but stops the search along a particular branch if a predefined boundary value is exceeded. The branch and bound method has been shown to be effective on many problem instances.

Greedy hill climbing strategies modify the current subset in such a way that results in the maximum improvement in the objective function (see e.g., [210]). Sequential forward search (SFS) [191] [192], sequential backward search (SBS), and bidirectional search [203] are some variations to the greedy hill climbing method. In these methods, the current subset is modified by adding or deleting features. SFS sequentially searches the feature space by starting from the empty set and selects the best single feature to add into the set in each iteration. On the contrary, SBS starts from the full feature set and removes the worst single feature from the set in each iteration. Both approaches add or remove features one at a time. Algorithms with sequential searches are fast and have a time complexity of $O(n^2)$. Sequential forward floating search (SFFS) and sequential backward floating search (SBFS) [208] combine the strategies followed by SFS and SBS. Some feature selection algorithms randomly pick subsets of features from the feature space by following some probabilistic steps and sampling procedures. Examples include evolutionary algorithms [195] [196], and simulated annealing [189]. The use of randomness helps in the avoidance of getting trapped in local maxima.

2.2.2 Evaluation of the generated subset

After selecting the subsets from the original set of features, they are evaluated using an objective function. One possible objective function is the accuracy of the predictive model. Feature selection algorithms can be broadly divided into two categories: (1) wrapper, and (2) filter. In a wrapper method the classification or prediction accuracy of an inductive learning algorithm of interest is used for evaluation of the generated subset. For each generated feature subset, wrappers evaluate its accuracy by applying the learning algorithm using the features residing in the subset. Although it is a computationally expensive procedure, wrappers can find the subsets from the feature space with a high accuracy because the features match well with the learning algorithm. Filter methods are computationally more efficient than wrapper methods since they evaluate the accuracy of a subset of features using objective criteria that can be tested quickly. Common objective criteria include the mutual information, Pearson product-moment correlation coefficient, and the inter/intra class distance. Though filters are computationally more efficient than wrappers, often they produce a feature subset which is not matched with a specific type of predictive model and thus can yield worse prediction accuracies. Some of the filter-based methods are Chi-square [219], correlation-based (e.g., linear and rank), and entropy-based (e.g., information and gain-ratio) [220] filters.

2.3 Background Summary

We demonstrate the applicability of our proposed algorithms using three different applications. The applications of interest are: 1) the prediction of materials properties, 2) analysis of biological data, and 3) data integration. In this section we provide a brief summary on the applications stated above.

2.3.1 Materials property prediction

If one wants to determine properties of a given unknown material, the traditional approaches are lab measurements or computationally intensive simulations (for example using the Density Functional Theory). An attractive alternative is to employ learning algorithms. The idea is to learn the desired properties from easily obtainable information about the material. In this research work we consider an infinite polymeric chain composed of XY_2 building blocks, with X = C, Si, Ge, or Sn, and Y = H, F, Cl, or Br. We are interested in estimating different properties of such chains including dielectric constant and band gap. We assume an infinite polymer chain with a repeat unit containing 4 distinct building blocks, with each of these 4 blocks being any of CH_2, SiF_2, $SiCl_2$, GeF_2, $GeCl_2$, SnF_2, or $SnCl_2$. By plotting the total dielectric constant (composed of the electronic and ionic contributions) and the electronic part of the dielectric constant against the computed band gap, we find some correlations between these three properties. While some correlations are self-evident (and expected)—such as the inverse relationship between the band gap and the electronic part of the dielectric constant, and the large dielectric constant of those systems that contain contiguous SnF_2 units—it is not immediately apparent if these observations may be formalized in order to allow for quantitative property predictions for systems (within this sub-class, of course) not originally considered. For example, can we predict the properties of a chain with a repeat unit containing 8 building blocks (with each of the blocks being any of the aforementioned units)? In Section 2.6, we show that this can indeed be done with high-fidelity using our randomized search method.

We use specific sub-structures, or *motifs* or *scaffolds*, within the main structure to create the attribute vector. Let us illustrate this using the specific example of the polymeric dielectrics created using XY_2 building blocks. Say there are 7 possible choices (or motifs)

for each XY_2 unit: CH_2, SiF_2, $SiCl_2$, GeF_2, $GeCl_2$, SnF_2, and $SnCl_2$. The attribute vector may be defined in terms of 6 fractions, $|f_1, f_2, f_3, f_4, f_5, f_6\rangle$, where f_i is the fraction of XY_2 type or motif i (note that $f_7 = 1 - \Sigma_{i=1}^{6} f_i$). One can extend the components of the attribute vector to include clusters of 2 or 3 XY_2 units of the same type occurring together; such an attribute vector could be represented as $|f_1, \ldots, f_6, g_1, \ldots, g_7, h_1, \ldots, h_7\rangle$, where g_i and h_i are, respectively, the fraction of XY_2 pairs of type i and the fraction of XY_2 triplets of type i. In Section 2.6, we demonstrate that such a motif-based attribute vector does a remarkable job of codifying and capturing the information content of the XY_2 polymeric class of systems, allowing us to train our machines and make high-fidelity predictions.

2.3.2 Gene selection

Gene selection is based on SVMs [14-18]. It takes as input n genes $\{g_1, g_2, g_3, \ldots, g_n\}$, and l vectors $\{v_1, v_2, v_3, \ldots, v_l\}$. As an example, each v_i could be an outcome of a microarray experiment and each vector could be of the following form: $v_i = \{x_i^1, x_i^2, x_i^3, \ldots, x_i^n, y_i\}$. Here x_i^j is the expression level of the j^{th} gene g_j in experiment i. The value of y_i is either *+1* or *-1* based on whether the event of interest is present in experiment i or not. The problem is to identify a set of genes $\{g_i^1, g_i^2, g_i^3, \ldots, g_i^m\}$ sufficient to predict the value of y_i in each experiment. Given a set of vectors, the gene selection algorithm learns to identify the minimum set of genes needed to predict the event of interest and the prediction function. These vectors form the training set for the algorithm. Once trained, the algorithm is provided with a new set of data which is called the test set. The accuracy of gene selection is measured in the test set as a percentage of microarray data on which the algorithm correctly predicts the event of interest. The procedure solely relies on the concept of SVM.

The gene selection algorithm of Song and Rajasekaran [135] is based on the ideas of com-

bining the mutual information among the genes and incorporating correlation information to reject the redundant genes. The Greedy Correlation Incorporated Support Vector Machine (GCI-SVM) algorithm of [135] can be briefly summarized as follows: The SVM is trained only once and the genes are sorted according to the norm of the weight vector corresponding to these genes. Then the sorted list of genes are examined starting from the second gene. The correlation of each of these genes with the first gene is computed until one whose correlation with the first one is less than a certain predefined threshold is found. At this stage this gene is moved to the second place. Now the genes starting from the third gene are examined and the correlation of each of these genes with the second gene is computed until a gene whose correlation with the second gene is less than the threshold is encountered. The above procedure is repeated until end of the list of the sorted genes is reached. In the last stage, genes based on this adjusted sorted genes are selected. GCI-SVM brings the concept of sort-SVM and RFE-SVM [194] altogether which makes it more efficient. In a nutshell, GCI-SVM works as follows:

- Compute the correlation coefficient for each pair of genes.
- Train the SVM using the training data set.
- Sort the genes based on their weight values.
- Go through the sorted genes, pick those genes whose correlation with the previously picked genes is less than a threshold.
- Move in order all picked genes to the front of the sequence; correspondingly, unpicked genes are moved to the end.

We incorporate *gene selection algorithm* in our RFS algorithm to effectively find the best subset of features from the entire set of features. We calculate the accuracy of the selected

subset of features returned by the RFS with the help of GSA and proceeds the search based on the accuracy as stated in Algorithm 2.1 until desired convergence achieved. Please, see Section 2.5 for more details.

2.3.3 Data integration

Data integration involves combining data residing in different sources and providing users with a unified view of these data [188]. As an example, the same person may have health care records with different providers. It helps to merge all the records with all the providers and cluster these records such that each cluster corresponds to one individual. Such an integration, for instance, could help us avoid performing the same tests again and hence save money. Several techniques such as [187], [215], [190], or [216] have been proposed to solve the data integration problem. Tian *et al.* [205] have proposed several space and time efficient techniques to integrate multiple datasets from disparate data sources. They employ agglomerative hierarchical clustering techniques [218] to integrate data of similar types and avoid the computation of cross-products. It can cope up with some common errors committed in input data such as typing distance and sound distance. Furthermore, it can deal with some human-made typing errors e.g., reversal of the first and last names, nickname usage, and attribute truncation.

We incorporate data integration technique of [205] in our RFS algorithm. At each iteration RFS returns a subset of features such as name, sex, postal code, etc. columns from the different databases. Based on the selected features the data integration scheme of Tian *et al.* integrates data of similar types into clusters and calculate the accuracy of the clusters. The RFS then proceeds the search by exploiting the accuracy.

2.4 Our Algorithms

If we can identify a subset of the features that are the most important in determining a property, it will lead to computational efficiency as well as a better accuracy. It is conceivable that some of the features might be hurtful rather than helpful in predictions. Let $\vec{A} = |a_1, a_2, \ldots, a_n\rangle$ be the set of features under consideration. One could use the following simple strategy, in the context of any learning algorithm, to identify a subset of $\vec{A}$ that yields a better accuracy in predictions than $\vec{A}$ itself. For some small value of k (for example 2), we identify all the $\binom{n}{k}$ subsets of $\vec{A}$. For each such subset we train the learner, figure out the accuracy we can get, and pick that subset $\vec{S}$ that yields the best accuracy. Now, from the remaining features, we add one feature at a time to $\vec{S}$ and for each resultant subset, we compute the accuracy obtainable from the learner. Let $\vec{S}'$ be the set (of size $k+1$) of attributes that yields the best accuracy. Next, from the remaining attributes, we add one feature at a time to $\vec{S}'$ and identify a set of size $k+2$ with the best accuracy, and so on. Finally, from out of all of the above accuracies, we pick the best one.

2.4.1 Randomized feature selector (RFS)

We can think of the above simple technique as a greedy algorithm that tries to find an optimal subset of attributes and it may not always yield optimal results. On the other hand, it will be infeasible to try every subset of attributes (since there are 2^n such subsets). We propose the following novel approach instead: Consider the space of all possible subsets of attributes. We start with a random point p (i.e., a random subset of the features) in this space and calculate the accuracy q corresponding to this subset. We then flip an unbiased three sided coin with sides 1, 2, and 3. If the outcome of the coin flip is 1, we choose a random neighbor p' of this point by removing one feature from p and adding a new feature

to p. After choosing p', we compute its accuracy q'. If $q' > q$ then we move to the point p' and proceed with the search from p'. On the other hand, if $q' < q$, then we stay with point p (with some probability u) or move to point p' with probability $(1 - u)$. This step is done to ensure that we do not get stuck in a local maximum. If the outcome of the coin flip is 2, we choose a random neighbor p' by removing one feature from p and compute its accuracy q'. The next steps are the same as stated in the case of 1. Consider the last case where the outcome of the coin flip is 3. We choose a random neighbor p' by adding one feature to p and compute its accuracy q'. The rest of the steps are the same as above. If $q' > q$ then we move to the point p' and proceed with the search from p'. On the other hand, if $q' < q$, then we stay with point p (with some probability u) or move to point p' with probability $(1-u)$. We proceed with the search from the point we end up with. This process of searching the space is continued until no significant improvement in the accuracy can be obtained. A relevant choice for u is $\exp(-c(q - q'))$ for some constant c. In fact, the above algorithm resembles the simulated annealing (SA) algorithm of [198]. Note that our algorithm is very different from SA. In particular, our algorithm is much simpler than SA. Details of our algorithm can be found in Algorithm 2.1.

2.4.2 Randomized feature selector 2 (RFS2)

Another variation of RFS algorithm is RFS2. It is based on pure random walk on the search space. In RFS if $q' < q$, then we stay with point p (with some probability u) or move to point p' with probability $(1 - u)$ as described above. Instead in RFS2 if $q' < q$ we always stay with point p and restart the search from that point. Details of our algorithm can be found in Algorithm 2.2. Please, see line 19 which differentiates RFS2 and RFS.

2.4.3 Randomized feature selector 3 (RFS3)

RFS3 is another variation of RFS algorithm. In RFS3 we iterate the search space from different points in the search space. In each iteration if $q' < q$ we randomly select a brand new subset from search space and proceed the search as described in RFS. Details of our algorithm can be found in Algorithm 2.3. Please, see lines 19-22 which differentiate RFS3 and RFS.

2.5 Analysis of Our Algorithms

In this section we illustrate runtime analysis and convergence proof of RFS algorithm. The same analyses can be applied to RFS2 and RFS3 algorithms with some minor modifications. In RFS algorithm it is easy to see that each run of the **repeat** loop takes $O(n)$ time (please, see pseudo code of Algorithm 2.1). This can be reduced to $O(\log n)$ by keeping $F - F'$ as a balanced tree (such as a red-black tree). An important question is how many runs will be needed for convergence. In this section we answer this question. The analysis is based on representing the steps of the algorithm as a time homogeneous Markov chain. The analysis is similar to the one in [209].

We can conceive of a directed graph $G(V, E)$ for the feature selection problem as follows: Every subset of the n features is a node in G. From any node $p \in V$, there will be edges to its neighbors. Clearly, in the Algorithm RFS, for every node p, there are three kinds of neighbors: Let p' be a neighbor of p. If p' is of the first kind, then the number of features in p and p' will be the same. Thus there are $N_p^1 \leq n$ such neighbors. If p' is of the second type, then p' will have one more feature than p. Finally, if p' is of the third type, then p' will have one less feature than p. As a result, it follows that there will be $\leq 3n$ neighbors for any

Algorithm 2.1: Randomized Feature Selector (RFS)

Input: The set F of all possible features and an Inductive Learning Algorithm $\mathcal{L}$
Output: A near optimal subset F' of features
begin

1. Randomly sample a subset F' of features from F.
2. Run the inductive learning algorithm $\mathcal{L}$ using the features in F'.
3. Compute the accuracy A of the concept C learnt by $\mathcal{L}$.
4. **repeat**
5. Flip an unbiased three sided coin with sides 1, 2, and 3.
6. **if** (the outcome of the coin flip is 1){
7. Choose a random feature f from $F - F'$ and add it to F'.
8. Remove a random feature f' from F' to get F''.
9. } **else if** (the outcome of the coin flip is 2){
10. Choose a random feature f from $F - F'$ and add it to F' to get F''.
11. } **else if** (the outcome of the coin flip is 3){
12. Remove a random feature f from F' to get F''.
13. }
14. Run the inductive learning algorithm $\mathcal{L}$ using the features in F''.
15. Compute the accuracy A' of the concept C' learnt by $\mathcal{L}$.
16. **if** $(A' > A)$ {
17. $F' := F''$ and $A := A'$; Perform the search from F'.
18. } **else**{
19. With probability u perform the search from F' and with probability $1 - u$ perform the search from F'' with $A := A'$.
20. }

until *no significant improvement in the accuracy can be obtained*;
21. Output F'.

Algorithm 2.2: Randomized Feature Selector 2 (RFS2)

Input: The set F of all possible features and an Inductive Learning Algorithm $\mathcal{L}$
Output: A near optimal subset F' of features
begin

```
1   Randomly sample a subset F' of features from F.
2   Run the inductive learning algorithm L using the features in F'.
3   Compute the accuracy A of the concept C learnt by L.
4   repeat
5       Flip an unbiased three sided coin with sides 1, 2, and 3.
6       if (the outcome of the coin flip is 1){
7           Choose a random feature f from F − F' and add it to F'.
8           Remove a random feature f' from F' to get F''.
9       } else if (the outcome of the coin flip is 2){
10          Choose a random feature f from F − F' and add it to F'
            to get F''.
11      } else if (the outcome of the coin flip is 3){
12          Remove a random feature f from F' to get F''.
13      }
14      Run the inductive learning algorithm L using the features in F''.
15      Compute the accuracy A' of the concept C' learnt by L.
16      if (A' > A ){
17          F' := F'' and A := A'; Perform the search from F'.
18      } else{
19          Perform the search from F'
20      }
    until no significant improvement in the accuracy can be obtained;
21  Output F'.
```

node in the graph $G(V, E)$. Let the number of neighbors of p of the second and third types be N_p^2 and N_p^3, respectively.

Algorithm RFS starts from a random node p in G and performs a random walk in

Algorithm 2.3: Randomized Feature Selector 3 (RFS 3)

Input: The set F of all possible features and an Inductive Learning Algorithm $\mathcal{L}$
Output: A near optimal subset F' of features
begin

```
1   Randomly sample a subset F' of features from F.
2   Run the inductive learning algorithm L using the features in F'.
3   Compute the accuracy A of the concept C learnt by L.
4   repeat
5       Flip an unbiased three sided coin with sides 1, 2, and 3.
6       if (the outcome of the coin flip is 1){
7           Choose a random feature f from F − F' and add it to F'.
8           Remove a random feature f' from F' to get F''.
9       } else if (the outcome of the coin flip is 2){
10          Choose a random feature f from F − F' and add it to F'
            to get F''.
11      } else if (the outcome of the coin flip is 3){
12          Remove a random feature f from F' to get F''.
13      }
14      Run the inductive learning algorithm L using the features in F''.
15      Compute the accuracy A' of the concept C' learnt by L.
16      if (A' > A ){
17          F' := F'' and A := A'; Perform the search from F'.
18      } else{
19          Randomly sample a subset F' of features from F.
20          Run the learning algorithm L using the features in F'.
21          Compute the accuracy A of the concept C learnt by L.
22          Perform the search from F'.
23      }
    until no significant improvement in the accuracy can be obtained;
24  Output F'.
```

this graph. The next node visited can be of type 1, 2, or 3 all with equal probability. The next node visited depends only on the current node. We can thus model RFS as a time homogeneous Markov chain. In contrast, the simulated annealing algorithm has been modeled as a time inhomogeneous Markov chain (see e.g., [206] and [209]).

Note that for any two nodes p and p' in G, there is a directed path from p to p' and hence G is strongly connected. For any node p in G, let $q(p)$ be the accuracy of the feature set corresponding to p. If p is any node and p' is a neighbor of p of type k ($1 \leq k \leq 3$), then the transition probability $P_{pp'}$ from p to p' is given by:

$$P_{pp'} = \begin{cases} 0 & \text{if } p' \notin N(p) \& p' \neq p \\ \frac{1}{3N_p^k} \min(1, \exp\{c(q(p') - q(p))\}) & \text{if } p' \text{ is a neighbor of } p \text{ of type } k \end{cases}$$

and

$$P_{pp} = 1 - \sum_{p' \in N(p)} P_{pp'}$$

RFS is said to have converged if the underlying Markov chain had been in a globally optimal state at least once. Let p be the starting node (i.e., start state) of the Markov chain and let p' be a globally optimal state. Then there is a path from p to p' of length $\leq n$.

Let $\Delta = \max_{p \in V,\ p' \in N(p)} \{q(p) - q(p')\}$. Also, let the degree and diameter of $G(V, E)$ be d and D, respectively. Clearly, $d \leq 3n$ and $D \leq n$. If p' is a neighbor of p, then $P_{pp'}$ is $\geq \frac{1}{3d} \exp(-c\Delta)$. We can derive a time bound within which the RFS algorithm will converge as stated in the following Lemma.

Lemma 2.5.1. *If p is any state in V, then the expected number of steps before a global optimal state is visited starting from p is $\leq \left(\frac{1}{3d} \exp(-c\Delta)\right)^{-D}$.*

Proof. Let g be any globally optimal state. Then there exists a directed path from p to g in $G(V, E)$ of length $\ell \leq D$. Let $e_1, e_2, \ldots, e_\ell$ be the sequence of edges in the path. The probability that g is visited starting from p is greater than or equal to the probability that each one of the edges $e_i,\ 1 \leq i \leq \ell$ is traversed in succession. The later probability is at least $\left[(\frac{1}{3d}) \exp(-c\Delta)\right]^\ell \ \geq \left[(\frac{1}{3d}) \exp(-c\Delta)\right]^D$. As a result, the expected number of steps before g is visited is $\leq [3d\ \exp(c\Delta)]^D$. $\square$

Theorem 2.5.1. RFS converges in time $\leq 2m[3d\ \exp(c\Delta)]^D$, with probability $\geq (1 - 2^{-m})$, independent of the start state (for any integer $m \geq 1$).

Proof. Let $E = 2[3d\ \exp(c\Delta)]^D$. We prove by induction (on m) that the probability of a global optimal state g not being visited in mE steps is $\leq 2^{-m}$.

Induction Hypothesis. Independent of the start state, probability that g is not visited in mE steps is $\leq 2^{-m}$.

Base case (for $m = 1$) follows from Lemma 2.5.1 and Markov's inequality.

Induction step. Assume the hypothesis for all $m \leq (r - 1)$. We'll prove the hypothesis for $m = r$. Let $p_E, p_{2E}, \ldots, p_{(r-1)E}$ be the states of the Markov chain during time steps $E, 2E, \ldots, (r - 1)E$, respectively. Let A be the event: g is not visited during the first E steps, and B be the event: g is not visited during the next $(r - 1)E$ steps.

The probability P that g is not visited in rE steps is given by

$$P \ = \ Prob.[B/A] \ \times \ Prob.[A].$$

Since $Prob.[B/A]$ depends only on what state the Markov chain is in at time step E and the time duration $(r - 1)E$, we infer:

$$P = Prob.[A] \sum_{p \in V} Prob.[B/p_E = p] \times Prob.[p_E = p].$$

Applying the induction hypotheis, Prob.[A] is $\leq 1/2$ and Prob.[B/$p_E = p$] is $\leq 2^{-(r-1)}$ for each $p \in V$. Therefore, we have,

$$P \ \leq \ \frac{1}{2}\, 2^{-(r-1)} \ = \ 2^{-r}.\square$$

2.6 Results and Discussion

We have employed our randomized feature selection algorithms on three different application domains. These applications include but not limited to the prediction of properties of materials, processing of biological data, and data integration. Our algorithms are generic and can be used in conjunction with any learning algorithm.

2.6.1 Materials property prediction

We consider polymeric dielectrics created using the XY_2 blocks as described in Section 2.3.1. If we assume that our repeat unit consists of 4 building blocks, and that each building block can be any of 7 distinct units (namely, CH_2, SiF_2, $SiCl_2$, GeF_2, $GeCl_2$, SnF_2, and $SnCl_2$), we have a total of 175 distinct polymer chains (accounting for translational symmetry). Of these, we set 130 to be in the training set, and the remainder in the test set to allow for validation of the machine learning model.

Attribute vectors may be chosen in different ways. Consider the motif-based one as described in Section 2.3.1, i.e., our attribute vector, $\vec{A^i} = |f_1^i, \ldots, f_6^i, g_1^i, \ldots, g_7^i, h_1^i, \ldots, h_7^i\rangle$,

TABLE 2.6.1: KRR and modified KRR (mg-KRR and mo-KRR) schemes

		Bandgap		Electric DC		Total DC	
System	Method	Accuracy	Features	Accuracy	Features	Accuracy	Features
4-Block	KRR	92.98%	20	93.75%	20	96.49%	20
	mg-KRR	93.07%	19	94.22%	11	97.23%	14
	mo-KRR	93.43%	16	94.23%	18	97.63%	14
8-Block	KRR	96.95%	20	90.58%	20	95.81%	20
	mg-KRR	96.95%	20	90.64%	15	95.99%	19
	mo-KRR	97.45%	17	95.17%	12	97.68%	13

where f_j^i, g_j^i and h_j^i are, respectively, the fraction of XY_2 units of type j, the fraction of pair clusters of XY_2 units of type j and the fraction of triplet clusters of XY_2 units of type j. Once our machine has learned how to map between the attribute vectors and the properties using the training set, we make predictions on the test set (as well as the training set). Furthermore, we considered several 8-block repeat units (in addition to the 175 4-block systems), and performed our machine learning scheme.

We have tested the above techniques on the KRR scheme with the systems represented using the motif-based attribute vectors. We refer to the greedy extension as the modified greedy KRR (mg-KRR) approach and the modified optimization version as mo-KRR. mg-KRR and mo-KRR are based on Sequential Forward Search (SFS) and Randomized Feature Selector (RFS) algorithms, respectively. In each iteration of each of the algorithms (e.g. mg-KRR and mo-KRR) the accuracy of the selected subset is measured by employing KRR scheme. Based on the accuracy it proceeds the search until desired accuracy/convergence achieved. An assessment of the improvement in the predictive power when mg-KRR and mo-KRR are used for the three properties of interest (namely, the band gap, the electronic part of the dielectric constant and the total dielectric constant) is presented in Table 2.6.1. As can be seen, the level of accuracy of the machine learning schemes is uniformly good for

TABLE 2.6.2: KRR, modified KRR, distributed KRR, and randomized KRR (mo-KRR, dis-KRR, and ran-KRR) schemes

	Bandgap		Electric DC		Total DC	
Method	Accuracy	Features	Accuracy	Features	Accuracy	Features
KRR	92.85%	162	89.30%	162	87.01%	162
mo-KRR	94.07%	127	92.01%	125	88.57%	127
dis-KRR	93.56%	111	91.18%	118	87.63%	113
ran-KRR	96.48%	105	95.68%	102	94.47%	102

all three properties across the 4-block training and test set, as well as the 8-block test set, indicative of the high-fidelity nature of this approach. In particular, note that the mo-KRR method, in general, leads to better accuracy. More importantly, typically, the number of attribute components decreases significantly. This means a significant reduction in the run times of the algorithm while predicting parameter values for an unknown material.

Furthermore to demonstrate the practical applicability we have tested all the versions of our algorithm under KRR scheme with the systems represented using the motif-based attribute vectors. Each vector contains 162 components of 6 units, 23 pairs, and 133 triplets. We refer to the modified optimization version as mo-KRR (base on RFS algorithm), modified randomized version as ran-KRR (based on RFS2 algorithm), and distributed version of mo-KRR as dis-KRR (based on RFS3 algorithm). In dis-KRR we iterate the search space from different points in the search space. In each iteration we randomly select a brand new subset of components from 162 components and proceed the search by executing mo-KRR. The number of iteration is selected in such a way so that the finishing time of dis-KRR is same as the mo-KRR. When all the iteration exhausts dis-KRR outputs the best one by considering all the iterations. The results are shown in Table 2.6.2. In general all the versions of our algorithm lead to better accuracy with significant decrease of attribute components. Please, note that real datasets are used to perform all the experiments to predict materials property.

TABLE 2.6.3: GSA and modified GSA (GSA and mo-GSA) schemes

		GSA		Modified GSA	
System	Method	Accuracy	Features	Accuracy	Features
Dataset 1	GAUSSIAN	50%	15	54%	10
	LINEAR	49%	15	62%	12
Dataset 2	GAUSSIAN	52%	20	60%	13
	LINEAR	53%	20	65%	13
Dataset 3	GAUSSIAN	49%	25	58%	9
	LINEAR	50%	25	58%	11
Dataset 4	GAUSSIAN	50%	30	59%	13
	LINEAR	56%	30	62%	13

2.6.2 Gene selection

We have used the gene selection algorithm to identify some of the best features that can together identify two groups. The gene selection algorithm has two phases. In the first phase, the algorithm is trained with a training dataset. In this phase the algorithm comes up with a model of concept. In the second phase of the algorithm a test dataset is presented. The model learned in the first phase is used to classify the elements residing in the test dataset. As a result, the accuracy of the model learned can be computed. At first, we generated 4 simulated datasets each having 200 subjects with 15, 20, 25, and 30 features, respectively. Each of the features has been given a random value in the range $[0, 99]$. We then randomly assigned a class label to each of the subjects residing in each dataset. Specifically, each subject is assigned to group 1 with probability $\frac{1}{2}$ and it is assigned to group 2 with probability $\frac{1}{2}$. We trained the classifier using a training set which consists of 50 percent of data from each of group 1 and group 2 (data being chosen randomly). The test set is formed using the other 50 percent from group 1 and group 2, respectively.

At first GSA is trained with the training set and it builds a model of concept using SVMs [135]. We have used LINEAR, and GAUSSIAN RBF kernel functions in SVM to build the

model of concept. Using the test data we have measured the accuracy by employing the model built. We then employ our randomized feature selection algorithm RFS on each of the datasets. We call this scheme as modified GSA (mo-GSA). In each iteration mo-GSA randomly selects a subset of features as described in Algorithm 2.1. The accuracy of the selected subset is measured by GSA. Based on the accuracy mo-GSA proceeds to the search space until desired accuracy/convergence achieved. From the result shown in Table 2.6.3 it is evident that after employing mo-GSA the accuracy has been greatly improved and at the same time the number of features has decreased significantly with respect to GSA.

2.6.3 Data integration

Data integration technique of Tian *et al.* [205] is used to detect similar types of data from a set of databases. To test the performance of our approach, we generated 4 artificial datasets each having 10,000 subjects where each subject has 5 features. The features consist of a person's first name, last name, date of birth, sex, and zip code. In general, each person has multiple records. Since errors are introduced randomly in the features, instances of the same individual may differ from each other. Accuracy of any data integration method is calculated as the fraction of persons for whom all the instances have been correctly identified to be belonging to the same person.

At first we execute the algorithm proposed by [205] directly on each of the datasets and calculate the accuracy as defined above. We call this scheme as Data Integrator (DI). We then employ our randomized feature selection algorithm RFS on each of the datasets. We call this scheme as modified Data Integrator (mo-DI). In each iteration mo-DI randomly selects a subset of features as described in Algorithm 2.1. The accuracy of the selected subset is measured by [205]. Based on the accuracy mo-DI proceeds the search until desired

TABLE 2.6.4: DI and modified DI (DI and mo-DI) schemes

	Data Integrator		Modified Data Integrator	
System	Accuracy	# of Features	Accuracy	# of Features
Dataset 1	46.72%	5	89.71%	2
Dataset 2	85.50%	5	90.31%	3
Dataset 3	85.51%	5	90.32%	4
Dataset 4	85.50%	5	86.61%	3

accuracy/convergence achieved. From the result shown in Table 2.6.4 it is evident that after employing mo-DI the accuracy has been greatly improved and at the same time the number of features has also decreased with respect to DI.

2.6.4 The comparisons

We compared RFS algorithm with both of wrapper and filter-based schemes. At first, we generated a simulated dataset having 200 subjects with 15 features. Each of the features was given a random value in the range $[0, 99]$. We then randomly assigned a class label to each of the subjects residing in the dataset. Specifically, each subject was assigned to class 1 with probability $\frac{1}{2}$ and it was assigned to class 2 with probability $\frac{1}{2}$. We further divided the dataset into two parts namely control and test sets. Control set consists of 50 percent of data from each of group 1 and group 2 (data being chosen randomly). The test set was formed using the other 50 percent from group 1 and group 2, respectively.

The comparison procedure had two phases. In the first phase, the feature selection algorithm of interest selected the best feature subset from the control set. In the second phase the accuracy was calculated with the help of GSA from the test set considering only the subset of features given by the first phase. In the case of filter-based methods, we took the best subset of features in such a way that the size of the subset was identical to the size

TABLE 2.6.5: The Comparisons

Method Name	Class	% Accuracy	# of Features
mo-GSA	*wrapper*	54%	10
Chi-squared Filter	*filter*	48%	10
Linear Correlation-based Filter	*filter*	51%	10
Rank Correlation-based Filter	*filter*	51%	10
Information Gain-based Filter	*filter*	46%	10
Gain Ratio-based Filter	*filter*	46%	10
CFS Filter	*wrapper*	50%	2

of the subset returned by mo-GSA. On the other hand wrapper-based methods gave the best subset from the entire space of features. In our comparisons, the filter-based methods we had employed Chi-square, correlation-based (e.g., linear and rank), and entropy-based (e.g., information and gain-ratio) filters. The wrapper methods used is CFS [221]. This algorithm makes use of best first search for searching the attribute subset space. The information about the implementation details of these algorithms can be found in [191]. The comparison results show that our feature selection algorithm (specifically mo-GSA) outperforms some of the best known algorithms existing in the current literature. Please, see Table 2.6.5 for details. As the datasets and the class label were randomly generated from uniform distribution, the correlations among the subjects in the same class are very low. The accuracies of the selected features given by the feature selection algorithms of interest reflect this notion. As the correlations are very low, the prediction model built by the learning algorithms will also not be sufficiently accurate to classify the subjects from unseen data.

2.7 Conclusions

We have presented three novel randomized search techniques which are generic in nature and can be applied to any inductive learning algorithm for selecting a subset of the most relevant features from the set of all possible features. The proposed schemes fall into the class of wrapper methods where the prediction accuracy in each step is determined by the learning algorithm of interest. To demonstrate the validity of our approaches, we have applied it in three different applications, namely, biological data processing, data integration, and materials property prediction. It is evident from the simulation results shown above that our proposed techniques are indeed reliable, scalable, and efficient. Our experiments also reveal that our feature selection algorithms perform better than some of the best known algorithms existing in the current literature.

Chapter 3

Novel Algorithms for the Discovery of Time Series Motifs

Time series motif mining (TSMM) is an important problem that has numerous applications in image processing, patterns identification, intrusion detection, etc. TSMM problem can be though of as a special case of the closest pair problem. Numerous algorithms have been presented for solving these problems. For instance, for the closest pair problem on n points there exists an $O(n \log n)$ algorithm (when the dimension is a constant). There also exist randomized algorithms with an expected linear time. However these algorithms do not perform well in practice. The algorithms that are employed in practice have a worst case quadratic run time. One of the best performing algorithms for the TSMM problem is MK. In this research work we present an elegant called MPR for the TSMM problem that performs better than MK. Also, we present approximation algorithms for the TSMM problem that are faster than MK by a factor of up to more than 40.

3.1 Introduction

Given a set of n points in any metric space, the problem of finding the closest pair of points is known as the Closest Pair Problem (CPP) and has been well studied. In his seminal paper, Rabin proposed a randomized algorithm with an expected run time of $O(n)$ [121] (where the expectation is in the space of all possible outcomes of coin flips made in the algorithm). Rabin's algorithm used the floor function as a basic operation. In 1979, Fortune and Hopcroft presented a deterministic algorithm with a run time of $O(n \log\log n)$ assuming that the floor operation takes $O(1)$ time [110]. Both of these algorithms assume a constant-dimensional space (and the run times have an exponential dependency on the dimension). Other clasical algorithms include [120, 112]. Yao has proven a lower bound of $\Omega(n \log n)$ on the algebraic decision tree model (for any dimension) [129]. This lower bound holds under the assumption that the floor function is not allowed. One of the major issues with the above algorithms is the fact that their run times are exponentially dependent on the dimension.

Time series motif mining (TSMM) is a crucial problem that can be thought of as the CPP in a large dimensional space. In one version of the TSMM problem, we are given a sequence S of real numbers and an integer ℓ. The goal is to identify two subsequences of S of length ℓ each that are the most similar to each other (from among all pairs of subsequences of length ℓ each). These most similar subsequences are referred to as *time series motifs*. Let C be a collection of all the ℓ-mers of S. (An ℓ-mer is nothing but a contiguous subsequence of S of length ℓ). Clearly, the ℓ-mers in C can be thought of as points in $\Re^{\ell}$. As a result, the TSMM problem is the same as CPP in $\Re^{\ell}$. Any of the algorithms known for the CPP can thus be used to solve the TSMM problem. A typical value for ℓ of practical interest is several hundreds (or more). For these values of ℓ, the above algorithms ([121],[110],[120],[112]) will take an unacceptable amount of time (because of the exponential dependence on the

dimension). Designing an efficient practical and exact algorithm for the TSMM problem remains an ongoing challenge.

Mueen, et al. have presented an elegant exact algorithm called MK for TSMM [116]. MK improves the performance of the brute-force algorithm with a novel application of the triangular inequality. MK is currently the best-performing algorithm in practice for TSMM. A number of probabilistic as well as approximate algorithms are also known for solving this problem (see e.g., [103, 105, 111, 113, 114, 124, 125]). For instance, the algorithm of [105] exploits algorithms proposed for finding (ℓ, d)-motifs from biological data. The idea here is to partition the time series data into frames of certain width. Followed by this, the mean value in each frame is computed. This mean is quantized into four intervals and as a result, the original time series data is converted into a string of characters from an alphabet of size 4. Finally, any (ℓ, d)-motif finding algorithm is applied on the transformed string to identify the time series motifs.

In this research work we present efficient algorithms for Time Series Motif Mining. Our algorithms improve the results reported in several papers including [116], [105], and [101]. We present both exact and approximation algorithms. Our exact algorithm improves the run time of MK by a factor of up to 2. Our approximation algorithm is up to two orders of magnitude faster than MK while maintaining a very high accuracy.

3.2 Exact Time Series Motif Mining Algorithms

The input for the TSMM problem are a sequence $T = a_1, a_2, \ldots, a_n$ and an integer ℓ. The goal is to find two ℓ-mers of T that are the closest to each other (from among all the pairs of ℓ-mers of T). A general version of this problem is one where the input consists of n

points from $\Re^\ell$ and we want to identify the two closest points. A straight forward algorithm will compute the distance between every pair of ℓ-mers and output the pair with the least distance. Since we can compute the distance between two ℓ-mers in $O(\ell)$ time, this simple algorithm for TSMM will run in a total of $O(n^2\ell)$ time.

3.2.1 MK algorithm

The MK algorithm of [116] speeds up the brute force method by pruning off a large number of pairs that cannot possibly be the closest. There are two main ideas used in MK. The first idea in the algorithm is to speedup the computation of distances. Let $x = x_1, x_2, \ldots, x_\ell$ and $y = y_1, y_2, \ldots, y_\ell$ be any two ℓ-mers. To compute the distance between x and y, the algorithm keeps adding $(x_i - y_i)^2$ for $i = 1, 2, \ldots, \ell$. When the sum exceeds δ^2, this pair is immediately dropped (without completing the rest of the distance computation). Here δ is a known upper bound on the distance between the closest pair of ℓ-mers. This technique is known as *early abandoning*. For example, a value for δ could be otained at the beginning of any algorithm using a random sample of ℓ-mers. From thereon, δ could be dynamically updated to be the smallest distance among all the ℓ-mer pairs that have been processed until that point in the algorithm.

The second idea in MK uses the triangular inequality in a novel way. Let x and y be any two ℓ-mers. At any stage in the algorithm, we have an upper bound δ on the distance between the closest pair of ℓ-mers. If $d(x, y)$ can be inferred to be greater than δ, then we can drop the pair (x, y) from future consideration (since this pair cannot be the closest). Ideally, we would like to calculate $d(x, y)$ exactly for every pair of ℓ-mers x and y. But this will take too much time. MK circumvents this problem by **estimating** the distance between x and y via the triangular inequality. In particular, a random reference ℓ-mer r is chosen and the

distance between each ℓ-mer and r is computed. The ℓ-mers are kept in an ascending order of their distances to r. From thereon, $d(r,y) - d(r,x)$ is used as a lower bound on $d(x,y)$. If this lower bound is $> \delta$, then (x,y) is dropped from future consideration.

The above algorithm is generalized to employ multiple reference ℓ-mers. The use of multiple references speeds up the algorithm.

3.2.2 An analysis of the MK algorithm and our new idea

In this section we provide an (informal) analysis of the MK algorithm to explain why the algorithm has a very good performance. Specifically, if we choose multiple random reference points, the algorithm achieves a much better run time than having a single reference point. We explain why this is the case.

For ease of understanding consider the 2D Euclidean space. The analysis can be extended to points in $\Re^\ell$. For any two ℓ-mers x and y, the closer $d(r,y)-d(r,x)$ is to $d(x,y)$, the better will be our estimate and hence the better will be our chance of dropping (x,y) (if (x,y) is not the closest pair). It turns out that the quality of the lower bound $d(r,y) - d(r,x)$ is decided by two factors: 1) the angle $\angle rxy$ and 2) $d(r,x)$. We illustrate this with an example. Let $x = (0,0)$ and $y = (1,0)$. Consider a reference point $r_1 = (1,1)$ (Figure 3.2.3(a)). Note that r_1 is at a distance of $\sqrt{2}$ from x. In this case $d(r_1,x) - d(r_1,y) = 0.414$. Also, $\angle r_1xy = 45°$. Let r_2 be the point we get by keeping the distance between the reference point and x the same, but changing this angle to 30° (Figure 3.2.3(b)). In this case, $d(r_2,x) - d(r_2,y)$ improves to 0.6722. As another example, if the reference point r lies on the perpendicular bisector of x and y, then $d(r,x) - d(r,y) = 0$.

For any two input points x and y, if we pick multiple reference points randomly, then we would expect that at least one of these reference points r will be such that the angle

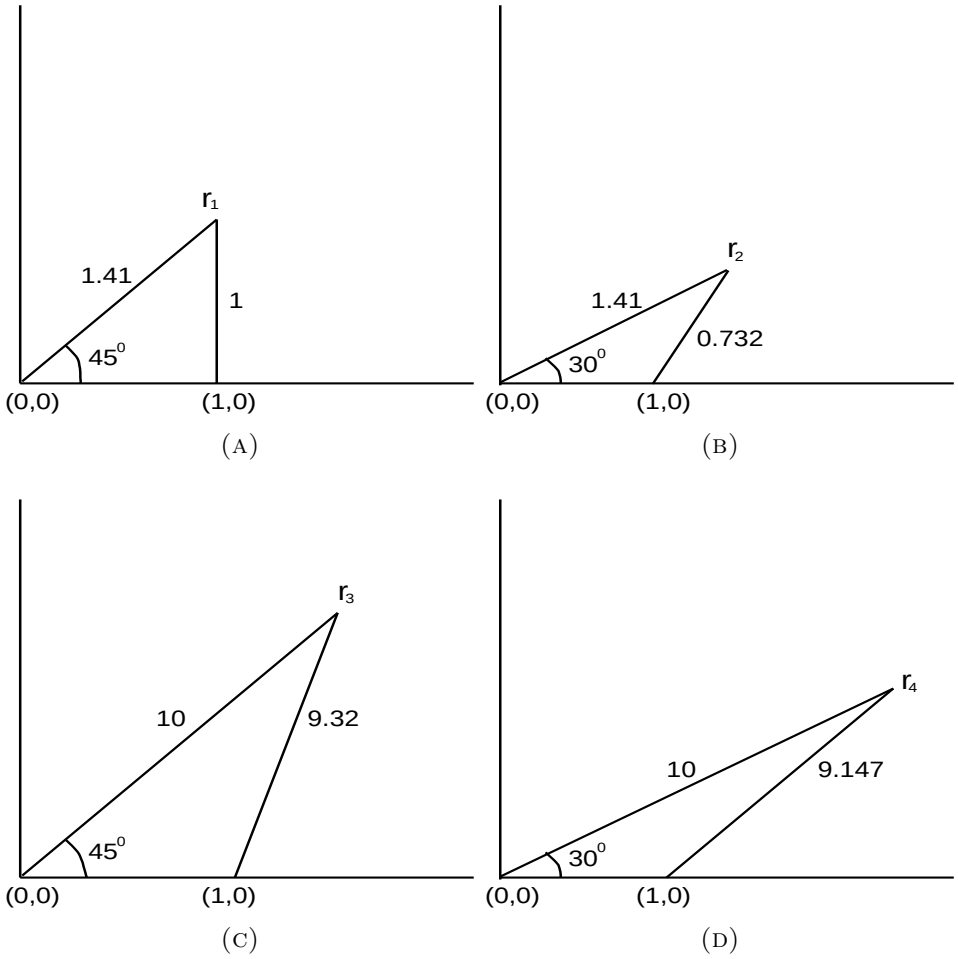


FIGURE 3.2.1: The effect of reference points.

$\angle rxy$ will be such that $d(r,x) - d(r,y)$ will be 'large'. In contrast, if we have only one reference point, for some pairs of points the corresponding angles may be 'good', but for a good percentage of the pairs, the angles may not be 'good'. (A reference point r is ''good' if $d(r,x) - d(r,y)$ is close to $d(x,y)$).

The effect of $d(r,x)$ on $d(r,x) - d(r,y)$ can be seen with the same examples. Consider the example of Figure 3.2.1(a). Assume that we keep the angle the same but increase $d(r_1,x)$ to 10 and get the reference point r_3 (Figure 3.2.1(c)). In this case, $d(r_3,x) - d(r_3,y)$ improves

to 0.6803. Also, in the example of Figure 3.2.1(b), say we keep the angle the same but increase $d(r_2, x)$ to 10 and get the reference point r_4. In this case, $d(r_4, x) - d(r_4, y)$ improves to 0.8526. Of course, if the reference point r lies on the perpendicular between x and y, then however large $d(r, x)$ could be, $d(r, x) - d(r, y)$ will continue to be zero! However, the probability of this happening is low. For a given angle θ, we can compute the limit of $d(r, x) - d(r, y)$ as $d(r, x)$ tends to ∞. For instance when the angle is 45° (Figure 3.2.1(a)), this limit is $\frac{1}{\sqrt{2}} \approx 0.707$.

3.2.3 Our algorithm

Our proposed new algorithm indeed exploits the relationship between $d(r, x)$ and $d(r, x) - d(r, y)$. In particular, we pick a collection C of random reference points and project each of these points out by multiplying each coordinate value of each point by a factor of f. For example, f could be 10. The rest of the algorithm is the same as MK.

A pseudocode for our algorithm, called *Motif discovery with Projected Reference points (MPR)*, is given below.

Algorithm MPR

Input: $T = a_1, a_2, \ldots, a_n$ and an integer ℓ, where each a_i is a real number (for $1 \leq i \leq n$). Input are also q and f, where q is the number of references and f is the projection factor.
Output: The two closest ℓ-mers of T.

1) Pick q random ℓ-mers of T as references; Project these references by multiplying each element in each ℓ-mer by f. Let these projected references be $r_1, r_2, \ldots, r_q$.
2) Compute the distance between every ℓ-mer of T and every projected reference ℓ-mer.
3) Sort the ℓ-mers of T with respect to their distances to r_1. Let the sorted ℓ-mers be

$p_1, p_2, \ldots, p_{n-\ell+1}$.

4) Let $\delta = \infty$; Let $answer = (0, 0)$;

5) for $i := 1$ **to** $(n - \ell + 1)$ **do**

for $j := (i + 1)$ **to** $(n - \ell + 1)$ **do**

$failure := false$;

for $k := 1$ **to** q **do**

if $d(r_k, p_j) - d(r_k, p_i) > \delta$ **then**

$failure := true$; **exit**;

if $failure$ **then exit else**

Compute $d(p_i, p_j)$;

if $d(p_i, p_j) < \delta$ **then**

$\delta := d(p_i, p_j)$; $answer = (i, j)$;

6) Output (i, j).

Observation: Please note that even though the above algorithm has been presented for solving the TSMM problem, it is straight forward to extend it to solve CPP in $\Re^\ell$.

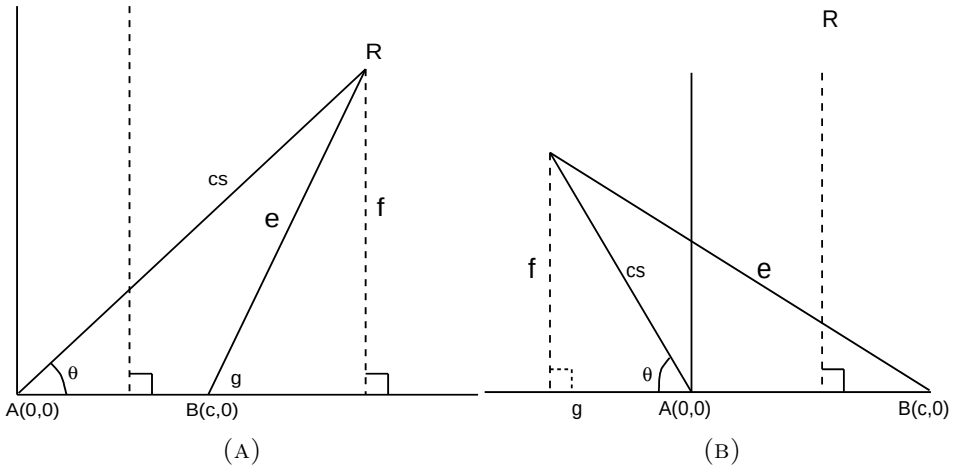


FIGURE 3.2.2: The effect of scaling on reference points.

3.2.4 An analysis of our algorithm

In this section we show why our idea of projecting reference points improves the performance of the algorithm. Let the input points be from $\Re^d$ for some integer d. Consider any two input points A and B. Let R be any reference point. Note that any three points are coplanar. Consider any hyperplane $\mathcal{H}$ containing A, B, and R. The following analysis is for the case that $\mathcal{H}$ passes through the origin. If we multiply every coordinate of R by the same number, then the resultant point will also lie in $\mathcal{H}$. This is because the equation defining $\mathcal{H}$ will be of the form $a_1x_1 + a_2x_2 + \cdots + a_dx_d = 0$. Thus, in order to see how $d(R, A) - d(R, B)$ changes with a scaling of R, it suffices to consider the case that these three points are in 2D.

Without loss of generality let A be $(0, 0)$ and B be $(c, 0)$, for some real number c. There are two cases to consider for the position of R relative to A and B: 1) R is to the right of the perpendicular bisector of A and B; 2) R is to the left of the perpendicular bisector between A and B. These two cases are illustrated in Figures 3.2.2(a) and 3.2.2(b), respectively. Note that when R lies on the perpendicular bisector of A and B, $d(R, A) - d(R, B)$ will be zero.

In case 1, let $d(A, R)$ be cs, s being a scaling factor. $f = cs\sin\theta$, $g = cs\cos\theta - c$, and $e = \sqrt{f^2 + g^2}$. As a result, $e = \sqrt{c^2s^2 + c^2 - 2c^2s\cos\theta} = cs\sqrt{1 + \frac{1}{s^2} - \frac{2\cos\theta}{s}}$. Using the fact that $(1 - u)^n \approx 1 - nu$ (when $nu << 1$), $e \approx cs + \frac{c}{2s} - c\cos\theta$. Thus, $d(A, R) - d(R, B) \approx c\cos\theta - \frac{c}{2s}$. Clearly, when c and θ are the same, the value of $d(R, A) - d(R, B)$ increases when s increases.

In case 2, let $d(A, R)$ be cs, for a scaling factor of s. Clearly, $f = cs\sin\theta$, $g = cs\cos\theta$. Thus, $e = \sqrt{f^2 + (g + c)^2}$. Also, $e = \sqrt{c^2s^2 + c^2 + 2c^2s\cos\theta} = cs\sqrt{1 + \frac{1}{s^2} + \frac{2\cos\theta}{s}}$. Using the approximation mentioned in case 1, we see that $e \approx cs + \frac{c}{2s} + c\cos\theta$. Therefore, $d(R, B) - d(R, A) \approx \frac{c}{2s} + c\cos\theta$. In this case, when c and θ are the same, the value of $d(R, B) - d(R, A)$ increases when s decreases.

But for a given reference point, and two input points A and B, we do not know which of the two cases will hold. But we can expect that half of the randomly chosen reference points will fall under case 1 and the other half will be expected to fall under case 2. If we only employ a scaling factor s that is greater than one, then, for an expected half of the reference points we expect to see an improvement in the estimate of a lower bound for $d(A, B)$. This explains why our algorithm performs better than MK.

The above analysis can also be used to better understand the MK algorithm.

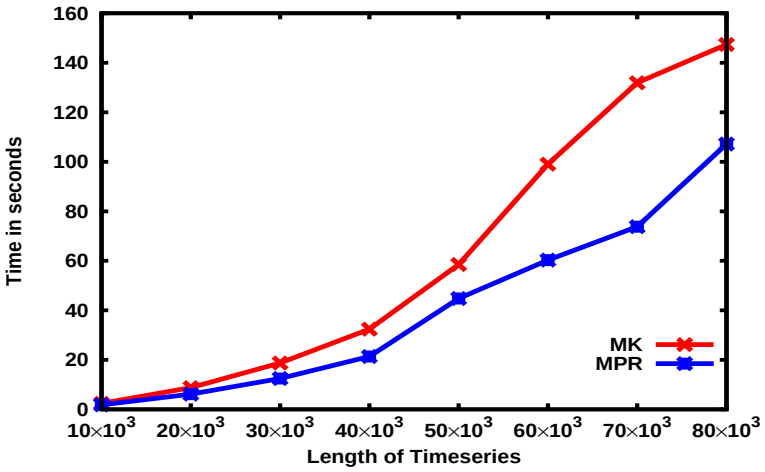
3.2.5 Fixed radius nearest neighbors problem (FRNNP)

In this problem we are given n points $a_1, a_2, \ldots, a_n$ in $\Re^\ell$ and a radius R (which is a real number) and the problem is to identify the R-neighborhood of each input point. If p is an input point, its R-neighborhood is defined to be the set of all input points that are within a distance of R from p. FRNNP has numerous applications. One of the applications of vital importance is that of molecular simulations.

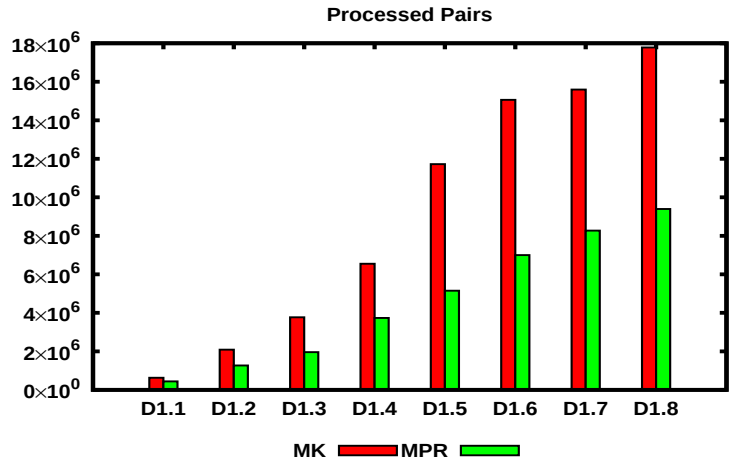
We can modify MPR to solve this problem as well. The modified version is given below. Let $N(i)$ denote the R-neighborhood of $a_i, 1 \leq i \leq n$.

TABLE 3.2.1: Number of pairs and runtime comparisons in Euclidean space. CPU times are given in seconds.

		MK		MPR		Gain	Speed up
Dataset	Size	Processed Pairs	CPU Time	Processed Pairs	CPU Time	Processed Pairs	Time
D1.1	10×10^3	628,856	2.41	438,363	1.82	1.43	1.32
D1.2	20×10^3	2,088,182	8.76	1,266,334	6.16	1.65	1.42
D1.3	30×10^3	3,770,808	18.74	1,959,506	12.47	1.92	1.50
D1.4	40×10^3	6,550,980	32.39	3,734,578	21.35	1.75	1.52
D1.5	50×10^3	11,721,674	58.53	5,149,979	44.77	2.28	1.31
D1.6	60×10^3	15,061,807	99.01	6,997,972	60.28	2.15	1.64
D1.7	70×10^3	15,594,331	131.91	9,391,459	73.80	1.66	1.79
D1.8	80×10^3	17,780,625	147.40	8,269,054	107.19	2.15	1.38



(A) CPU times consumed by MK and MPR methods



(B) Number of pairs processed by MK and MPR methods

FIGURE 3.2.3: Performance evaluations between MK between MPR methods.

Algorithm MPR-FRNNS

Input: $T = a_1, a_2, \ldots, a_n$ and R, where each a_i is a point in $\Re^\ell$ (for $1 \leq i \leq n$) and R is a real number. Input are also q and f, where q is the number of references and f is the projection factor.

Output: $N(i)$, for $1 \leq i \leq n$.

TABLE 3.2.2: Number of pairs as a function of number of references. Number of time series is 50k with 1,024 attributes.

References	Method	Processed Pairs	CPU Time
10	MK	9,094,227	47.75
	MPR	6,573,553	39.58
20	MK	4,363,302	49.62
	MPR	2,227,327	36.79
30	MK	2,004,017	37.97
	MPR	1,035,769	29.84
40	MK	1,678,898	47.70
	MPR	889,405	41.40
50	MK	1,319,278	40.26
	MPR	790,229	35.69

1) Pick q random points of T as references; Project these references by multiplying each coordinate of each reference point by f. Let these projected references be $r_1, r_2, \ldots, r_q$.

2) Compute the distance between every point of T and every projected reference point.

3) Sort the points of T with respect to their distances to r_1. Let the sorted points be $p_1, p_2, \ldots, p_n$.

4) for $i := 1$ **to** n **do** $N(i) := \emptyset$;

5) for $i := 1$ **to** n **do**

  for $j := (i+1)$ **to** n **do**

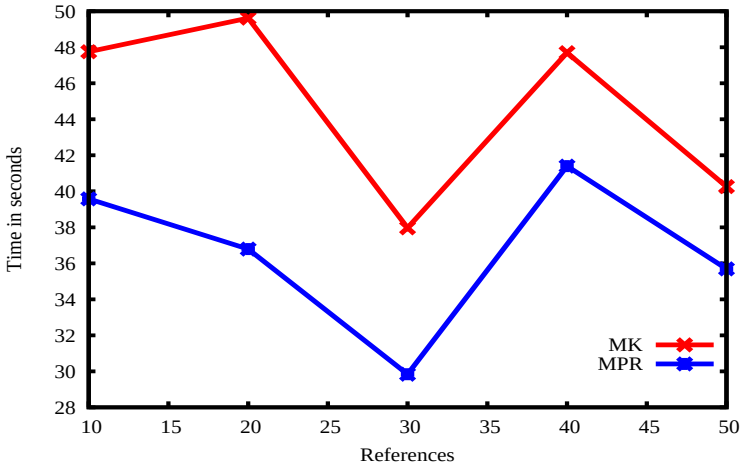
    $failure := false$;

    for $k := 1$ **to** q **do**

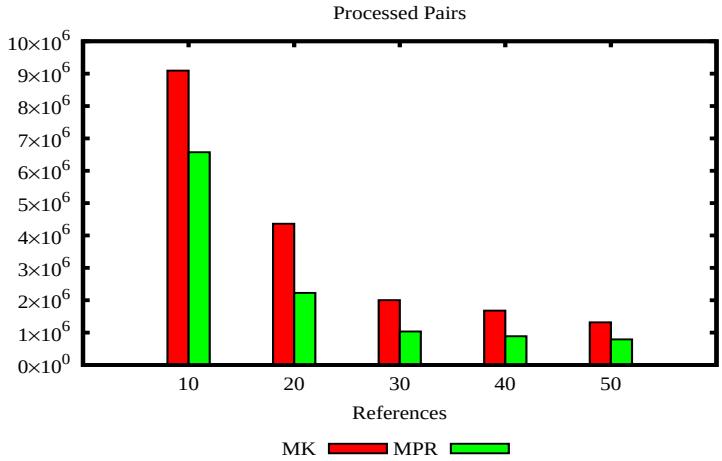
      if $d(r_k, p_j) - d(r_k, p_i) > R$ **then**

        $failure := true$; **exit**;

    if $failure$ **then exit else**



(A) CPU times consumed by MK and MPR



(B) Number of pairs processed by MK and MPR

FIGURE 3.2.4: Performance evaluations between MK and MPR with respect to the number of references.

Compute $d(p_i, p_j)$;

if $d(p_i, p_j) \leq R$ **then** add j to $N(i)$ and

add i to $N(j)$;

6) Output $N(i)$, for $1 \leq i \leq n$.

3.2.6 An experimental comparison of MK and MPR

A typical algorithm in the literature for CPP has two phases. In the first phase pairs of points that cannot possibly be the closest are eliminated. In the second phase distance is computed between every pair of points that survive the first phase. The time spent in the first phase is typically very small and hence is negligible (compared to the time spent in the second phase). Also, the time needed to process the pairs in the second phase is linear in the number of surviving pairs. As a result, it suffices to report the number of pairs (to be processed in the second phase) as a measure of performance (see e.g., [101]). In this research work also we use this measure of performance throughout.

We have experimentally compared the performance of MK and MPR on different data sets. The machine we have used has an Intel(R) Core(TM) i3-M350 2.27 GHz CPU with 4GB RAM running Windows 7 (64 bit). As mentioned in [116], random walk data set is the most difficult case for time series mining algorithms since the probability of the existence of very close motif pairs is very low. We have also used the same data for our comparison. In particular, we have used 8 different random walk data sets of sizes ranging from 10K to 80K. Our algorithm performs better than MK for higher motif lengths. Both the algorithms have been run 5 times and the averages computed. We do not perform any comparison with the brute force method as that has already been done in [116].

In Table 3.2.1 we show the number of pairs processed (in the second phase) in MK and MPR. The size (i.e., the length) of the time series data varies from 10K to 80K, the motif length being 1024. The following parameter values have been used: $q = 1$ and $f = 10$. From this table we see that MK processes around $2\times$ the number of pairs processed by MPR. Figure 3.2.3 presents a graphical plot of the runtime requirements of MK and MPR

algorithms. This figure shows that the run time of MK is around 1.5× the run time of MPR. This improvement is quite significant for the following reason: There are two ideas used in MK, namely, early abandoning and the use of random reference points. As the authors point out in [116], the difference between using early abandoning alone and both the ideas is small, especially on random walk data. Table 3.2.2 and Figure 3.2.4 show how the number of pairs reduces with an increase in the number q of the reference points for MK and MPR algorithms.

3.3 Approximate Time Series Motif Mining Algorithms

In this research work we offer two approximation algorithms for the TSMM problem. These are called ATSMM1 and ATSMM2. These algorithms employ quantization and random sampling. We provide details on them next. The expected run times of these algorithms are also analyzed. Experimental evaluations indicate that our algorithms are significantly faster than the brute force and MK algorithms.

3.3.1 Our algorithm ATSMM1

We start by obtaining an upper bound on the distance between the closest pair of motifs. We can obtain such a bound with random sampling for example. For instance we can randomly pick $\sqrt{n}$ points, compute the closest distance between any pair in this sample, and use this distance as δ. Each attribute value is quantized using δ.

If the input time series sequence is of length n, we think of the input as consisting of $N = (n - \ell + 1)$ points in $\Re^\ell$. I.e., each ℓ-mer of the input sequence is a point in $\Re^\ell$. We can also think of each of the ℓ dimensions as an attribute.

Let a be one of the attributes. Let min_a and max_a be the minimum and maximum values for this attribute across all the motifs. Let $\delta' = \sqrt{2}\frac{\delta}{\sqrt{\ell}}$. Consider partitioning the values along this attribute into intervals of size $2\delta'$ each. Also change the interval $[min_a, max_a]$ to $[r + min_a, max_a]$ where r is uniformly random in the range $[0, \delta']$. Partition this range into intervals of size $2\delta'$ each. (The reason for this will be apparent in the analysis). Label these intervals with successive integers starting from 1. Each motif gets an integer index along this attribute depending on the motif's value for this attribute. If the motifs m_1 and m_2 have a distance of $\leq \delta'$, then the probability that they will fall in the same interval (i.e., will have the same index) is $\geq \frac{1}{2}$.

In a similar manner, assign an index for each motif along each attribute. Each motif gets ℓ integer indices. After calculating these indices, the algorithm proceeds as follows. There will be k stages in the algorithm (for some suitable value k). In each stage we get some candidate pairs. We identify the best among these. At the end of k stages we output the best pair among all the candidates processed. Candidate pairs in any stage are generated as follows. We randomly pick c attributes. For each motif there will be c indices corresponding to these attributes. We hash the motifs using these indices. Two motifs will fall into the same bucket in the hash table if they match in these c indices. Every pair of motifs in every bucket in the hash table is a candidate pair.

A pseudocode for ATSMM1 follows:

0) Pick a random sample of $\sqrt{n}$ motifs, and identify
   the distance δ between the closest pair in this
   sample using the brute-force algorithm;
1) **for** each attribute a **do**
   Compute the minimum and maximum values

for this attribute;

Let these be min_a and $\max_a$ respectively;

2) **for** each motif q **do**

for each attribute a **do**

Compute an index I_q^a for motif q along the attribute a by partitioning the range $[r+ min_a, max_a]$ into windows of width $2\delta'$ each;

Here $\delta' = \sqrt{2}\frac{\delta}{\sqrt{\ell}}$, and r is a random number in the range $[0, \delta']$;

Generate a priority queue Q with p keys all being equal to ∞;

for $i = 1$ to k **do**

Pick c attributes randomly and hash the motifs based on their indices for these attributes;

for each pair m of motifs in each bucket **do**

if the Euclidean distance (across all the attribute indices) between the motifs in m is less than the largest key in Q

then Delete the largest key from Q and insert m into Q with the Euclidean distance as the key;

3) Identify and output the best pair (in terms of Euclidean distance) in Q;

3.3.2 An analysis of our algorithm

We present an analysis for ATSMM1 and a similar analysis applies for ATSMM2 as well. If (m_1, m_2) is the closest pair of motifs, what can we say about the distance between these motifs along each of the ℓ attributes? Clearly, the expected distance between m_1 and m_2 along any attribute is $\leq \frac{\delta}{\sqrt{\ell}}$. This means that there is at least one attribute along which the distance between m_1 and m_2 is no more than $\frac{\delta}{\sqrt{\ell}}$. Also, we can see that there are at least $\frac{\ell}{2}$ attributes along which the distance between m_1 and m_2 is no more than $\delta' = \sqrt{2}\frac{\delta}{\sqrt{\ell}}$.

We infer that if the distance between m_1 and m_2 is $\leq \delta$, and if we pick one of the attributes randomly, then the probability that m_1 and m_2 will have the same index for this attribute is $\geq \frac{1}{4}$. If we pick c attributes randomly, the probability that m_1 and m_2 will have the same indices along all of these attributes if $\geq \left(\frac{1}{4}\right)^c$.

If we repeat the above process of sampling k times, then the probability that m_1 and m_2 will have the same indices in at least one of these stages of sampling is $\geq 1 - \left[1 - \left(\frac{1}{4}\right)^c\right]^k\Big]$. We want this probability to be high. In this research work by high probability we mean a probability of $\geq (1 - n^{-\alpha})$ where n is the input size and α is a probability parameter (typically assumed to be a constant ≥ 1). $[1 - \left(\frac{1}{4}\right)^c]^k$ will be $\leq n^{-\alpha}$ if $k \geq 4^c \alpha \log_e n$, using the fact that $(1 - y)^{1/y} \leq 1/e$ (for any $0 < y < 1$).

The above proof demonstrates that, for a fixed value of c, if k is chosen appropriately, then with high probability the closest pair will be identified. We can also estimate the expected run time using the following analysis.

Let p_1 be the largest probability that any pair will have the same index along a randomly chosen attribute. Let $M = (m_1, m_2)$ be this pair. Let p_2 be the second largest probability that any pair will have the same index along a randomly chosen attribute. Let $M' = (m_3, m_4)$ be this pair. Probability that m_1 and m_2 have the same index along all the c attributes chosen

in a stage of the algorithm is p_1^c. Thus the probability that m_1 and m_2 do not fall into the same bucket in a given stage is $(1-p_1^c)$. As a result, the probability that motifs of M do not fall into the same bucket in z successive stages is $(1-p_1^c)^z$. This probability is $\leq \exp(-zp_1^c)$ using the fact that $(1-y)^{1/y} \leq 1/e$ for any $0 < y < 1$. This probability will be $\leq n^{-\alpha}$ if $z \geq \frac{\alpha \log n}{p_1^c}$.

We want to ensure that the number of pairs generated in each stage is not too large. One possibility is to let the expected number of pairs generated in each stage be $O(n)$. The probability that m_3 and m_4 fall into the same bucket in any specific stage is $\leq p_2^q$. If this probability is $\leq \frac{1}{n}$, then the expected number pairs generated in any iteration will be $\leq n$. This happens if $c = \frac{\log n}{\log(1/p_2)}$. For this value of c, the value of z becomes $\alpha \log n \; n^{\log p_1 / \log p_2}$.

Given that the expected time we spend in hashing in each iteration and the time for generating the pairs is $O(n)$, it follows that the expected run time of the algorithm is $O\left(n^{1+\frac{\log p_1}{\log p_2}} \log n\right)$. Thus we get the following Theorem:

Theorem 3.3.1. The expected run time of the algorithm ATSMM1 is $O\left(n^{1+\frac{\log p_1}{\log p_2}} \log n\right)$. □

3.3.3 Our algorithm ATSMM2

Note that the analysis of ATSMM1 assumes that we compute the Euclidean distance between every pair of motifs. The computation of Euclidean distances is very costly especially when the dimension is very high. Instead if we can compute Hamming distances using the integer indices of the attributes, significant speedups can be obtained (especially if we utilize bit level operations). Our algorithm ATSMM2 indeed employs this approximation in addition to quantization and random sampling. As the experimental results (in Section 3.3.4) show, this approximation yields very good accuracies.

ATSMM2 is an out-of-core algorithm. To start with the input is in the disk. There are three steps in the algorithm and in each step we do one pass through the input data and hence there are a total of three passes through the data.

In each pass the algorithm incrementally retrieves information embedded in the dataset and after the final pass it outputs a pair of motifs. We claim that this pair of motifs is the closest pair with a very high probability. The details of our algorithm are provided next.

First pass

If the input sequence is of length n, we think of the input as consisting of $N = (n - \ell + 1)$ points in $\Re^{\ell}$. We can also think of each of the ℓ dimensions as an attribute. In the first pass we record the minimum and maximum values along each of the ℓ dimensions (or attributes). These values will be used to represent each coordinate of each motif in the Hamming space.

Second pass

We normalize each of the ℓ attributes in the second pass over the entire dataset using the (min, max) pairs. Normalization is the process of scaling any data so that it falls within a specified range. There are many methods of normalization, such as min-max normalization, z-score normalization, normalization by decimal scaling, etc. In this context we have followed min-max normalization technique.

Assume that the minimum and maximum values of an attribute a are given by min_a and max_a. Min-max normalization maps a value of an attribute $v(a)$ to $v'(a)$ in the new range $[new_{min_a}, new_{max_a}]$ by computing:

$$v'(a) = \frac{v - min_a}{max_a - min_a}(new_{max_a} - new_{min_a}) + new_{min_a} \tag{3.3.1}$$

ATSMM transforms the value of each attribute $v(a)$ of any motif to $v'(a)$ using the above formulation and encodes each transformed attribute into bits. User defines the number of bits to be used to encode each attribute. Suppose the user defines x bits for each attribute. The range of $(new_{min_a}, new_{max_a})$ will be $[0, 2^x - 1]$. It is easy to see that $v'(a)$ will be always in the range 0 to $2^x - 1$. After encoding all the attributes of a particular motif into bits, we concatenate the resulting bits to make an array of bits. We follow the same procedure stated above for every motif to transform it into a bit array. Note that each motif is represented as a bit array with ℓx bits.

Next, we randomly sample a subset of the motif attributes and hash the motifs based on this subset. Two motifs will be hashed into the same bucket if they have the same (quantized) values for the randomly chosen attributes. If two motifs fall into the same bucket in the hash table, this is a candidate pair. We keep a priority queue Q that stores the best p pairs that have been encountered thus far (for some relevant p). The key used for any pair in Q will be the Hamming distance between them (across all the attributes). The Hamming distance (across all the atributes) between each candidate pair will computed and inserted into Q if this Hamming distance is less than the largest key in Q.

We repeat this process of sampling and hashing k times (for some suitable value of k). In each stage of sampling the candidate pairs generated are used to update Q.

The closest pair in terms of Hamming distance may not necessarily be the closest pair in the original Euclidean space. This is the reason why we keep the priority Q.

Third pass

In the third pass we compute the Euclidean distance between every pair of motifs found in Q and output the best pair. Please, note that the original dataset always resides in the disk.

A pseudocode for ATSMM2 follows.

1) **for** each attribute a **do**

Compute the minimum and maximum values

for this attribute;

Let these be min_a and $\max_a$ respectively;

2) **for** each motif q **do**

Normalize each attribute of q using Equation 3.3.1;

Generate a priority queue Q with p keys

all being equal to ∞;

for $i = 1$ to k **do**

Pick c attributes randomly and hash the motifs

based on these attributes;

for each pair m of motifs in each bucket **do**

if the Hamming distance (across all the

attributes) between the motifs in m is less

than the largest key in Q

then Delete the largest key from Q and

insert m into Q with the Hamming

distance as the key;

3) Identify and output the best pair

(in terms of Euclidean distance) in Q;

TABLE 3.3.1: Runtime comparisons between Brute-force and ATSMM2. Min Dist refers to the distance of the closest pair. CPU times are given in seconds.

		Brute-force		ATSMM2		
Dataset	Time series	Min Dist	CPU Time	Min Dist	CPU Time	Avg. Rank
D2.1	2×10^3	226.76	2.80	226.76	1.70	1.00
D2.2	4×10^3	221.82	15.20	221.82	1.78	2.20
D2.3	6×10^3	224.09	36.03	224.09	1.90	2.60
D2.4	8×10^3	220.83	38.07	225.83	2.23	4.40
D2.5	10×10^3	221.39	89.73	224.37	2.49	3.00
D2.6	20×10^3	219.62	373.88	222.12	4.63	3.00
D2.7	40×10^3	215.17	1,497.25	218.26	12.36	4.00
D2.8	60×10^3	212.98	3,379.05	215.77	24.69	2.60
D2.9	80×10^3	212.44	5,038.03	215.53	43.25	3.00

TABLE 3.3.2: Performance of ATSMM2 in larger datasets. CPU times are given in seconds.

Dataset	Time series	Length	CPU Time
D3.1	1×10^6	512	292.06
		1,024	470.22
		2,048	836.74
D3.2	2×10^6	512	773.71
		1,024	1,327.31
		2,048	2,312.11
D3.3	3×10^6	512	1,559.81
		1,024	2,376.43
		2,048	4,040.52
D3.4	4×10^6	512	2,347.49
		1,024	3,795.44
		2,048	6,568.21
D3.5	5×10^6	512	3,486.84
		1,024	5,590.73
		2,048	9,920.47

3.3.4 An experimental evaluation of ATSMM2

We have performed rigorous experimental evaluations to test the scalability and effectiveness of ATSMM2. Since our algorithm may not always output the closest pair, we wanted to check the quality of output from our algorithm. To measure this quality, we have used the brute

TABLE 3.3.3: Performance of ATSMM2 with respect to the number of attributes. CPU times are given in seconds.

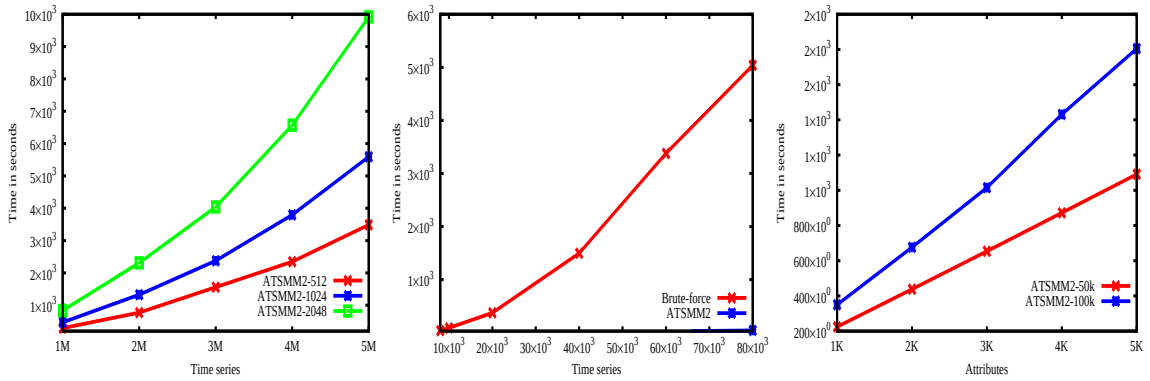
Dataset	Time series	Attributes	CPU Time
D4.1	5×10^4	10×10^3	224.45
		20×10^3	438.04
		30×10^3	653.41
		40×10^3	872.07
		50×10^3	1,091.04
D4.2	1×10^5	10×10^3	349.60
		20×10^3	676.24
		30×10^3	1,014.83
		40×10^3	1,431.29
		50×10^3	1,804.99

TABLE 3.3.4: Runtime comparisons between MK and ATSMM2. CPU times are given in seconds.

		MK	ATSMM2
Dataset	Time series	CPU Time	CPU Time
D5.1	2×10^3	9.70	22.41
D5.2	4×10^3	35.95	22.69
D5.3	6×10^3	83.96	23.60
D5.4	8×10^3	152.30	25.28
D5.5	10×10^3	234.08	27.73
D5.6	20×10^3	945.97	42.32
D5.7	40×10^3	3,844.77	88.42

force algorithm to identify the top pairs (the closest, the second closest, the third closest, etc.). We used these outputs to identify the rank of the best output from ATSMM2. The results are reported in Table 3.3.1. As is clear, for many of the cases, ATSMM2 finds the closest pair and for most of the cases ATSMM2 finds one of the top 3 closest pairs of motifs. Each dataset in Table 3.3.1 is generated randomly using uniform distribution. Each motif has 200 floating point attributes ranging from 0.00 to 50.00. 3 bits were used to encode each attribute. ATSMM2 picked 9 attributes randomly in each stage of sampling and the number of stages was 100. Please, see Figure 3.3.1(b) for runtime comparisons.

We ran ATSMM2 on a number of bigger datasets (see Table 3.3.2). Please, note that



(A) CPU times consumed by ATSMM2 with respect to various number of time series and attributes

(B) CPU times consumed by ATSMM2 and Brute-force methods

(C) CPU times consumed by ATSMM2 on large number of attributes

FIGURE 3.3.1: Performance evaluations of ATSMM2.

we were not able to run MK on these datasets. The dataset in Table 3.3.2 contains time series data of length 1×10^6 to 5×10^6. Each motif had 512 to 2048 floating point attributes. The value of each attribute ranged from 0.00 to 100.00 and was randomly populated using a uniform distribution. 5 bits were used to encode each attribute. ATSMM2 picked 15 attributes randomly in each stage of sampling and the number of stages was 100. Please, see Figure 3.3.1(a) for visual details. We also ran ATSMM2 with higher number of attributes. Please, see Table 3.3.3. The number of attributes ranged from 10×10^3 to 50×10^3. Attributes had values ranged from 0.00 to 100.00 and was randomly populated using a uniform distribution. 5 bits were used to encode each attribute and the algorithm picked 15 attributes randomly in each stage of sampling and the number of stages was 100. Figure 3.3.1(c) represents the pictorial representation of attribute vs. runtime. It is evident that the runtime linearly increases with respect to the increase of attributes. All the experiments were done on an Intel Westmere compute node with 48 GB of RAM. The operating system running was Red Hat Enterprise Linux Server release 5.7 (Tikanga).

We have also compared our algorithm ATSMM2 with MK. The number of time series ranged from 1×10^3 to 10×10^3 having 1024 attributes. Attributes were generated using uniform distribution. The values of the attributes ranged from 0.00 to 100. ATSMM2 picked 9 attributes randomly in each stage of sampling and the number of stages was 100. Please, see Table 3.3.4 for performance evaluations in detail. Please note that we could not run either the brute force algorithm or MK on large input data sets since they took too much time.

3.4 Conclusions

In this research work we have presented two novel algorithms for the discovery of time series motifs. We have compared our algorithms with the previous best algorithms for time series motif mining. Our exact algorithm is faster than MK by a factor of up to 2. Our approximate algorithm is faster than MK by a factor of up to more than 40. As the data size increases, the difference between the run times of MK and ATSMM2 becomes more.

Part V

Conclusions and Future Directions

In this thesis we have worked on a number of problems related to big data analytics. Problems that we have focused on include metagenomic phylogenetic clustering, spliced junctions discovery, scaffolding, biological sequence compression, error correction, genotype-phenotype correlations, GWAS, clustering, closest pair problem, and feature selection. We have presented novel algorithms for solving these problems. Our algorithms perform better than the best prior algorithms.

In future we intend to work on these problems to obtain even better algorithms. We also plan to expand our research efforts by considering other related problems. The expectation is that most of our effort will be on the invention of novel algorithms and their implementations. We plan to work closely with domain experts (such as biologists). Given that big data arise in every walk of life, we hope to work with scientists from different areas of science and engineering.

References

[1] S.-M. Ahn, T.-H. Kim *et al.*: **The first Korean genome sequence and analysis: full genome sequencing for a socio-ethnic group**, *Genome Res.*, 19:1622-1629, 2009.

[2] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman: **Basic local alignment search tool**, *J Molecular Biology*, 215(3):403-10, 2004.

[3] M.C. Brandon, D.C. Wallace, and P. Baldi: **Data structures and compression algorithms for genomic sequence data**, *Bioinformatics*, 25:1731-1738, 2009.

[4] M.D. Cao, T.I. Dix, L. Allison, and C. Mears: **A simple statistical algorithm for biological sequence compression**, *IEEE Data Compression Conference* (DCC), pp. 43-52, 2007.

[5] S. Christley, Y. Lu, C. Li, and X. Xie: **Human genomes as email attachments**, *Bioinformatics*, 25:274-275, 2009.

[6] S. Deorowicz, D. Agnieszka, and S. Grabowski: **Genome compression: a novel approach for large collections**, *Bioinformatics*, 29(20):1-7, 2013.

[7] S. Deorowicz and S. Grabowski: **Robust relative compression of genomes with random access**, *Bioinformatics*, 27(21):2979-2986, 2011.

[8] M.H.-Y. Fritz, R. Leinonen, G. Cochrane, and E. Birney: **Efficient storage of high throughput DNA sequencing data using reference-based compression**, *Genome Res*, 21:734-740, 2011.

[9] S.W. Golomb: **Run-length encodings**, *IEEE Transactions on Information Theory*, 12(3):399-401, 1966.

[10] D. Huffman: **A method for the construction of minimum-redundancy codes**, *Proceedings of the Institute of Radio Engineers*, pp. 1098-1101. 1952.

[11] S. Kurtz, A. Phillippy, A.L. Delcher, M. Smoot, M. Shumway, C. Antonescu, and S.L. Salzberg: **Versatile and open software for comparing large genomes**, *Genome Biology*, 5(2):R12, 2004.

[12] S. Levy *et al.*: **The diploid genome sequence of an Asian individual**, *Nature*, 456:60-66, 2008.

[13] A. Moffat: **Implementing the PPM data compression scheme**, *IEEE Trans. Commun.*, 38(11):1917-1921, 1990.

[14] I. Ochoa, M. Hernaez, and T. Weissman: **iDoComp: a compression scheme for assembled genomes**, *Bioinformatics*, 31(5):626-633, 2014.

[15] D. Pavlichin, T. Weissman, and G. Yona: **The Human Genome Contracts Again**, *Bioinformatics*, 29(17):2199-2202, 2013.

[16] E. Peter: **Universal codeword sets and representations of the integers**, *IEEE Trans. Information Theory*, 21(2):194-203, 1975.

[17] A.J. Pinho, D. Pratas, and S.P. Garcia: **GReEn: a tool for efficient compression of genome resequencing data**, *Nucleic Acids Res.*, 40:e27, 2012.

[18] S. Saha and S. Rajasekaran: **ERGC: an efficient referential genome compression algorithm**, *Bioinformatics*, 31(21):3468-3475, 2015.

[19] K. Shanika, P. Simon, and Z. Justin: **Optimized relative lempel-ziv compression of genomes**, 34^{th} *Australasian Computer Science Conference*, 113:91-98, 2011.

[20] Z.D. Stephens *et al.*: **Big Data: Astronomical or Genomical?**, *PLoS Biol.*, 13(7):e1002195, 2015.

[21] C. Wang and D. Zhang: **A novel compression tool for efficient storage of genome resequencing data**, *Nucleic Acids Res*, 39:E45-U74, 2011.

[22] J. Ziv and A. Lempel: **A universal algorithm for sequential data compression**, *IEEE Trans Inf Theory*, 23:337-343, 1977.

[23] J.K. Bonfield and M.V. Mahoney: **Compression of FASTQ and SAM format sequencing data**, *PLoS One*, 8:e59190, 2013.

[24] T. Bose, M.H. Mohammed, A. Dutta, and S.S. Mande: **BIND - An algorithm for loss-less compression of nucleotide sequence data**, *J Biosci*, 37:785-789, 2012.

[25] M. Burrows and D. J. Wheeler: **A block-sorting lossless data compression algorithm**, *SRC Research Report*, 1994.

[26] X. Chen, S. Kwong, and M. Li: **A compression algorithm for DNA sequences and its applications in genome comparison**, *Genome Informat Ser.*, 10:51-61, 1999.

[27] X. Chen, M. Li, B. Ma, and J. Tromp: **DNACompress: fast and effective DNA sequence compression**, *Bioinformatics*, 8:1696-1698, 2002.

[28] A.J. Cox, M.J. Bauer, T. Akobi, and G. Rosone: **Large-scale compression of genomic sequence databases with the Burrows-Wheeler transform**, *Bioinformatics*, 28:1415-1419, 2012.

[29] S. Deorowicz and S. Grabowski: **Compression of DNA sequence reads in FASTQ format**, *Bioinformatics*, 27:860-862, 2011.

[30] S. Grumbach and F. Tahi: **Compression of DNA sequences**, *IEEE Data Compression Conference* (DCC), Snowbird, Utah, pp. 340-350, 1993.

[31] S. Grumbach and F. Tahi: **A new challenge for compression algorithms**, *Genet Seq Inform Process Manag*, 30:875-886, 1994.

[32] F. Hach, I. Numanagic, C. Alkan, and S.C. Sahinalp: **SCALCE: boosting sequence compression algorithms using locally consistent encoding**, *Bioinformatics*, 28:3051-3057, 2012.

[33] D.C. Jones, W.L. Ruzzo, X. Peng, and M.G. Katze: **Compression of nextgeneration sequencing reads aided by highly efficient de novo assembly**, *Nucleic Acids Res*, 40:e171, 2012.

[34] C. Kingsford and R. Patro: **Compression of short-read sequences using path encoding**, *bioRxiv*, 2014.

[35] G. Korodi, I. Tabus, J. Rissanen, and J. Astola: **DNA sequence compression - based on the normalized maximum likelihood model**, *IEEE Sign Process Mag* 24:47-53, 2007.

[36] S. Kuruppu, B. Beresford-Smith, T. Conway, and J. Zobel: **Iterative dictionary construction for compression of large DNA data sets**, *IEEE-ACM Trans Computat Biol Bioinformatics*, 9:137-149, 2012.

[37] B. Langmead, C. Trapnell, M. Pop, and S.L. Salzberg: **Ultrafast and memory-efficient alignment of short DNA sequences to the human genome**, *Genome Biology*, 10:R25, 2009.

[38] H. Li and R. Durbin: **Fast and accurate short read alignment with Burrows-Wheeler transform**, *Bioinformatics*, 25:1754-1760, 2009.

[39] M.H. Mohammed, A. Dutta, T. Bose, S. Chadaram, and S.S. Mande: **DELIMINATE-a fast and efficient method for loss-less compression of genomic sequences**, *Bioinformatics*, 28:2527-2529, 2012.

[40] A.J. Pinho, P.J.S.G. Ferreira, A.J.R. Neves, and C.A.C. Bastos: **On the representability of complete genomes by multiple competing finite-context (Markov) models**, *PLoS One*, 6:e21588, 2011.

[41] A.J. Pinho and D. Pratas: **MFCompress: a compression tool for FASTA and multi-FASTA data**, *Bioinformatics*, 30:117-118, 2014.

[42] N. Popitsch and A.V. Haeseler: **NGC: lossless and lossy compression of aligned high-throughput sequencing data**, *Nucleic Acids Res*, 41:e27, 2013.

[43] S.C. Sahinalp and U. Vishkin: **Efficient approximate and dynamic matching of patterns using a labeling paradigm**, 37^th *Annual Symposium on Foundations of Computer Science*, pp. 320-328, 1996.

[44] W. Tembe, J. Lowey, and E. Suh: **G-SQZ: compact encoding of genomic sequence and quality data**, *Bioinformatics*, 26:2192-2194, 2010.

[45] M. Chaisson, P. Pevzner, and H. Tang: **Fragment assembly with short reads**, *Bioinformatics*, 20:2067-2074, 2004.

[46] P. A. Pevzner, H. Tang, and M.S. Waterman: **An Eulerian path approach to DNA fragment assembly**, *Proceedings of the National Academy of Sciences of the United States of America*, 98:9748-9753, 2001.

[47] J. Butler *et al.*: **ALLPATHS: de novo assembly of whole-genome shotgun microreads**, *Genome Res*, 18:810-820, 2008.

[48] H. Shi, B. Schmidt, W. Liu, and W. Mller-Wittig: **Accelerating Error Correction in High-Throughput Short-Read DNA Sequencing Data with CUDA**, *IEEE International Symposium on Parallel & Distributed Processing*, pp. 1-8, 2009.

[49] D.R. Kelley, M.C. Schatz, and S.L. Salzberg: **Quake: quality-aware detection and correction of sequencing errors**, *Genome biology*, 11:R116, 2010.

[50] X. Yang, K.S. Dorman, and S. Aluru: **Reptile: representative tiling for short read error correction**, *Bioinformatics*, 26:2526-2533, 2010.

[51] P. Medvedev, E. Scott, B. Kakaradov, and P. Pevzner: **Error correction of high-throughput sequencing datasets with non-uniform coverage**, *Bioinformatics*, 27(13):i137-i141, 2011.

[52] Y. Liu, J. Schrder, and B, Schmidt: **Musket: a multistage k-mer spectrum-based error corrector for Illumina sequence data**, *Bioinformatics*, 29(3):308-315, 2013.

[53] L. Ilie and M. Molnar: **RACER: Rapid and accurate correction of errors in reads**, *Bioinformatics*, 29(19):2490-2493, 2013.

[54] Y. Heo, X.L. Wu, D. Chen, J. Ma, and W.M. Hwu: **BLESS: bloom filter-based error correction solution for high-throughput sequencing reads**, *Bioinformatics*, 30(10):1354-1362, 2014.

[55] M.T. Tammi *et al.*: **Correcting errors in shotgun sequences**, *Nucleic Acids Res.*, 31(15):4663-4672, 2003.

[56] S. Batzoglou *et al.*: **ARACHNE: a whole-genome shotgun assembler**, *Genome Res.*, 12:177-189, 2002.

[57] L. Salmela and J. Schroder: **Correcting errors in short reads by multiple alignments**, *Bioinformatics*, 27:1455-1461, 2011.

[58] W-C. Kao, A.H. Chan, and Y.S. Song: **ECHO: a reference-free short-read error correction algorithm**, *Genome research*, 21:1181-1192, 2011.

[59] J. Schroder *et al.*: **SHREC: a short-read error correction method**, *Bioinformatics*, 25:2157-2163, 2009.

[60] L. Ilie, F. Fazayeli, and S. Ilie: **HiTEC: accurate error correction in high-throughput sequencing data**, *Bioinformatics*, 27:295-302, 2011.

[61] L. Salmela: **Correction of sequencing errors in a mixed set of reads**, *Bioinformatics*, 26:1284-1290, 2010.

[62] J. Buhler and M. Tompa: **Finding motifs using random projections**, 5^{th} *Annual International Conference on Computational Molecular Biology* (RECOMB), pp. 69-76, 2011.

[63] D. Hernandez, P. Francois, L. Franois, M. sters, and J. Schrenzel: **De novo bacterial genome sequencing: millions of very short reads assembled on a desktop computer**, *Genome Research*, 18(5):802-809, 2008.

[64] D.R. Zerbino, and E. Birney: **Velvet: Algorithms for de novo short read assembly using de Bruijn graphs**, *Genome Research*, 18:821-829, 2008.

[65] R. Li *et al.*: **De novo assembly of human genomes with massively parallel short read sequencing**, *Genome Research*, 20:265-272, 2010.

[66] A. Smith *et al.*: **Using quality scores and longer reads improves accuracy of solexa read mapping**, *BMC Bioinformatics*, 9:128-135, 2008.

[67] C. Trapnell, L. Pachter and S.L. Salzberg: **TopHat: discovering splice junctions with RNA-Seq**, *Bioinformatics*, 25:1105-1111, 2009.

[68] B. Langmead, C. Trapnell, M. Pop, and S.L. Salzberg: **Ultrafast and memory-efficient alignment of short DNA sequences to the human genome**, *Genome Biol.*, 10:R25, 2009.

[69] K.F. Au, H. Jiang, L. Lin, Y. Xing, and W.H. Wong: **Detection of splice junctions from paired-end RNA-seq data by SpliceMap**, *Nucleic Acids Res.*, 38:4570-4578, 2010.

[70] K. Wang *et al.*: **MapSplice: accurate mapping of RNA-seq reads for splice junction discovery**, *Nucleic Acids Res.*, 38:e178, 2010.

[71] D.W. Bryant Jr, R. Shen, H.D. Priest, W.K. Wong, and T.C. Mockler: **Supersplat-spliced RNA-seq alignment**, *Bioinformatics*, 26:1500-1505, 2010.

[72] M.T. Dimon, K. Sorber, and J.L. DeRisi: **HMMSplicer: a tool for efficient and sensitive discovery of known and novel splice junctions in RNA-Seq data**, *PLoS One*, 5:e13875, 2010.

[73] Q. Pan, O. Shai, L.J. Lee, B.J. Frey, and B.J. Blencowe: **Deep surveying of alternative splicing complexity in the human transcriptome by high-throughput sequencing**, *Nat. Genet.*, 40:1413-1415, 2008.

[74] H. Jiang and W. Wong: **Seqmap: mapping massive amount of oligonucleotides to the genome**, *Bioinformatics*, 24:2395-2396, 2008.

[75] A. Ameur, A. Wetterbom, L. Feuk, and U. Gyllensten: **Global and unbiased detection of splice junctions from rna-seq data**, *Genome Biology*, 11:R34, 2010.

[76] D. Kim, G. Pertea, C. Trapnell, H. Pimente, R. Kelley, and S.L. Salzberg: **TopHat2: accurate alignment of transcriptomes in the presence of insertions, deletions and gene fusions**, *Genome Biology*, 14:R36, 2013.

[77] S. Saha and S. Rajasekaran: **NRRC: A Non-referential Reads Compression Algorithm**, 11^{th} *International Symposium on Bioinformatics Research and Applications* (ISBRA), pp. 297-308, 2015.

[78] V. Vapnik: **The Nature of Statistical Learning Theory**, *Berlin: Springer-Verlag*, 1995.

[79] C. Cortes and V. Vapnik: **Support vector networks**, *Mach Learn.*, 20:1-25, 1995.

[80] H. Li, J. Ruan, and R. Durbin: **Mapping short DNA sequencing reads and calling variants using mapping quality scores**, *Genome Res.* 18:1851-1858, 2008.

[81] C. Trapnell *et al.*: **Transcript assembly and quantification by RNA-Seq reveals unannotated transcripts and isoform switching during cell differentiation**, *Nat. Biotechnol.*, 28:511-515, 2010.

[82] D.W. Thomas and N. Serban: **Fast and SNP-tolerant detection of complex variants and splicing in short reads**, *Bioinformatics*, 26:873-881, 2010.

[83] A. Dobin *et al.*: **STAR: ultrafast universal RNA-seq aligner**, *Bioinformatics*, 29:15-21, 2013.

[84] R.G. Gregory *et al.*: **Comparative analysis of RNA-Seq alignment algorithms and the RNA-Seq unified mapper (RUM)**, *Bioinformatics*, 27:2518-2528, 2011.

[85] M.L. Metzker: **Sequencing technologies - the next generation**, *Nat Rev Genet.*, 11(1):31-46, 2010.

[86] M. Pop and S.L. Salzberg: **Bioinformatics challenges of new sequencing technology**, *Trends Genet.*, 24:142-149, 2008.

[87] K.A. Frazer, S.S. Murray, N.J. Schork, and E.J. Topol: **Human genetic variation and its contribution to complex traits**, *Nature Rev. Genet.*, 10:241-251, 2009.

[88] M.J. Chaisson, D. Brinza, and P.A. Pevzner: **De novo fragment assembly with short mate-paired reads: does the read length matter?**, *Genome Res.*, 19:336-346, 2009.

[89] M. Nagarajan, T.D. Read, and M. Pop: **Scaffolding and validation of bacterial genome assemblies using optical restriction maps**, *Bioinformatics*, 24(10):1229-1235, 2008.

[90] F. Sanger and A.R. Coulson: **A rapid method for determining sequences in DNA by primed synthesis with DNA polymerase**, *J. Mol. Biol.*, 94(3):441-448, 1975.

[91] F. Sanger, S. Nicklen, and A.R. Coulson: **DNA sequencing with chain-terminating inhibitors**, *Proc. Natl. Acad. Sci*, 74(12):5463-5467, 1977.

[92] R. Staden: **A strategy of DNA sequencing employing computer programs**, *Nucleic Acids Research*, 6(7):2601-2610, 1979.

[93] S. Anderson: **Shotgun DNA sequencing using cloned DNase I-generated fragments**, *Nucleic Acids Research*, 9(13):3015-3027, 1981.

[94] D. Nathans and H.O. Smith: **Restriction endonucleases in the analysis and restructuring of DNA molecules**, *Annu. Rev. Biochem*, 44:273-293, 1975.

[95] S. Anderson: **Optical mapping: a novel, single-molecule approach to genomic analysis**, *Genome Res.*, 5:1-4, 1995.

[96] F.W. Engler *et al.*: **Locating sequence on fpc maps and selecting a minimal tiling path**, *Genome Res.*, 13:2152-2163, 2003.

[97] A. Ben-Dor *et al.*: **The restriction scaffold problem**, *J. Comput. Biol.*, 10:385-398, 2003.

[98] S. Reslewic *et al.*: **Whole-genome shotgun optical mapping of rhodospirillum rubrum**, *Appl. Environ. Microbiol.*, 71:5511-5522, 2005.

[99] J.T. Simpson and R. Durbin: **Efficient de novo assembly of large genomes using compressed data structures**, *Genome Res.*, 22(3):549-556, 2012.

[100] S. Kurtz *et al.*: **Versatile and open software for comparing large genomes**, *Genome Biology*, 5:R12, 2004.

[101] P. Achlioptas, B. Schölkopf, and K. Borgwardt: **Two-locus association mapping in subquadratic runtime**, *ACM SIGKDD international conference on Knowledge discovery and data mining* (KDD), 2011.

[102] C.E. Aston *et al.*: **Oligogenic combinations associated with breast cancer risk in women under 53 years of age**, *Human Genetics*, 116(3):208-221, 2005.

[103] P. Beaudoin, M. van de Panne, P. Poulin, and S. Coros: **Motion-Motif Graphs**, *Symposium on Computer Animation*, 2008.

[104] M.S. Charikar: **Similarity estimation techniques from rounding algorithms**, *Proc. ACM Symposium on Theory of Computing* (STOC), 2002.

[105] B. Chiu, E. Keogh, and S. Lonardi: **Probabilistic discovery of time series motifs**, *Proc. of the* 9^{th} *International Conference on Knowledge Discovery and Data mining* (KDD), pp. 493-498, 2003.

[106] J.H. Cho, *et al.*: **Identification of novel susceptibility loci for inflammatory bowel disease on chromosomes 1p, 3q, and 4q: evidence for epistasis between 1p and IBD1**, *Proceedings of the National Academy of Sciences of the United States of America*, 95(13):7502-7507, 1998.

[107] T.H. Consortium: **A second generation human haplotype map of over 3.1 million SNPs**, *Nature*, 449(7164):851-861, 2007.

[108] H.J. Cordell: **Detecting gene-gene interactions that underlie human diseases**, *Nat Rev Genet*, 10(6):392-404, 2009.

[109] N.J. Cox *et al.*: **Loci on chromosomes 2 (NIDDM1) and 15 interact to increase susceptibility to diabetes in mexican americans**, *Nature Genetics*, 21(2):213-215, 1999.

[110] S. Fortune and J.S. Hopcroft: **A note on Rabin's nearest-neighbor algorithm**, *Information Processing Letters*, 8(1):20-23, 1979.

[111] T. Guyet, C. Garbay and M. Dojat: **Knowledge construction from time series data using a collaborative exploration system**, *Journal of Biomedical Informatics*, 40(6):672-687, 2007.

[112] S. Khuller and Y. Matias: **A Simple Randomized Sieve Algorithm for the Closest-Pair Problem**, *Information and Computation*, 188(1):34-37, 1995.

[113] J. Meng, J. Yuan, M. Hans, and Y. Wu: **Mining Motifs from Human Motion**, *Proc. of EuroGraphics*, 2008.

[114] D. Minnen, C.L. Isbell, I. Essa, and T. Starner: **Discovering Multivariate Motifs using Subsequence Density Estimation and Greedy Mixture Learning**, 22^{nd} *Conf. on Artificial Intelligence* (AAAI), 2007.

[115] R. Motwani and P. Raghavan: **Randomized Algorithms**, *Cambridge University Press*, 1995.

[116] A. Mueen, E. Keogh, Q. Zhu, S. Cash, and B. Westover: **Exact Discovery of Time Series Motifs**, *SIAM International Conference on Data Mining* (SDM), 2009.

[117] S.K. Musani, D. Shriner, N. Liu, R. Feng, C.S. Coffey, N. Yi, H.K. Tiwari, and D.B. Allison: **Detection of gene x gene interactions in genome-wide association studies of human population data**, *Human Heredity*, 63(2):67-84, 2007.

[118] R. Nakamichi, Y. Ukai, and H. Kishino: **Detection of closely linked multiple quantitative trait loci using a genetic algorithm**, *Genetics*, 158(1):463-475, 2001.

[119] R. Paturi, S. Rajasekaran, and J. Reif: **The light bulb problem**, *Information and Computation*, 117:187-192, 1995.

[120] F. Preparata and M. Shamos: **Computational Geometry**, *Springer Verlag*, 1986.

[121] M. Rabin: **Probabilistic Algorithms**, *Algorithms and Complexity, Recent Results and New Directions*, Academic Press, pp 21-39, 1976.

[122] S. Rajasekaran and S. Saha: **Efficient Algorithms for the Two Locus Association Problem in GWAS and Related Problems**, 25^{th} *ACM International Conference on Information and Knowledge Management* (CIKM), Indianapolis, USA, pp. 24-28, 2016.

[123] S. Rajasekaran and S. Saha: **Efficient Algorithms for the Three Locus Problem in Genome-wide Association Study**, *IEEE International Conference on Data Mining* (ICDM), Indianapolis, USA, pp. 12-15, 2016.

[124] S. Rombo and G. Terracina: **Discovering representative models in large time series databases**, 6^{th} *International Conference on Flexible Query Answering Systems*, pp. 84-97, 2004.

[125] Y. Tanaka, K. Iwamoto, and K. Uehara: **Discovery of time-series motif from multi-dimensional data based on MDL principle**, *Machine Learning*, 58(2-3):269-300, 2005.

[126] S. Tata: **Declarative Querying For Biological Sequences**, *Ph.d. Thesis*, The University of Michigan, 2007.

[127] Y. Wang, X. Liu, K. Robbins, and R. Rekaya: **AntEpiSeeker: detecting epistatic interactions for case-control studies using a two-stage ant colony optimization algorithm**, *BMC Bioinformatics*, 3:117-117, 2010.

[128] J. Xu *et al.*: **Interaction effect of PTEN and CDKN1B chromosomal regions on prostate cancer linkage**, *Human Genetics*, 115(3):255-262, 2004.

[129] A.C. Yao: **Lower Bounds for algebric computation trees with integer inputs**, *SIAM J. Comput.*, 20:4655-4668, 1991.

[130] X. Zhang, S. Huang, F. Zou, and W. Wang: **TEAM: efficient two-locus epistasis tests in human genome-wide association study**, *Bioinformatics*, 26(12):i217-227, 2010.

[131] X. Zhang, F. Zou, and W. Wang: **Fastanova: an efficient algorithm for genome-wide association study**, 14^{th} *ACM SIGKDD international conference on Knowledge discovery and data mining* (KDD), Las Vegas, Nevada, USA, pp. 821-829, 2008.

[132] Wikipedia: **Single-nucleotide Polymorphism** [http://en.wikipedia.org/wiki/Single-nucleotide_polymorphism]

[133] D.N. Cooper, B.A. Smith, H.J. Cooke, S. Niemann, and J. Schmidtke: **An estimate of unique DNA sequence heterozygosity in the human genome**, *Hum Genet*, 69:201-205, 1985

[134] F.S. Collins, M.S. Guyer, and A. Charkravarti: **Variations on a theme: cataloging human DNA sequence variation.** *Science*, 278:1580-1581, 1997.

[135] M. Song and S. Rajasekaran: **A greedy correlation-incorporated SVM-based algorithm for gene selection**, *Advanced Information Networking and Applications Workshops*, 1:657-661, 2007.

[136] D. Achlioptas: **Database-friendly random projections: Johnson-Lindenstrauss with binary coins**, *J Comput Syst Sci*, 66(4):671-687, 2003.

[137] M.D. Ritchie *et al.*: **Multifactor-dimensionality reduction reveals high-order interactions among estrogen-metabolism genes in sporadic breast cancer**, *Genet*, 69:138-147, 2001.

[138] H.K. Tabor, N.J. Risch, and R.M. Myer: **Candidate-gene approaches for studying complex genetic traits: practical considerations**, *Nat Rev Genet*, 3(5):391-397, 2002.

[139] Hodgkinson *et al.*: **Addictions biology: haplotype-based analysis for 130 candidate genes on a single array**, *Alcohol Alcohol*, 43(5):505-515, 2008.

[140] Y. Saeys *et al.*: **A review of feature selection techniques in bioinformatics**, *Bioinformatics*, 23(19):2507-2517, 2007.

[141] T. Mitchell: **Machine Learning**, *McGraw Hill*, New York, 1997.

[142] M. Waddell, D. Page, F. Zhang, and B. Barlogie: **Predicting cancer susceptibility from single-nucleotide polymorphism data: A case study in multiple Myeloma**, *BIOKDD*, Chicago, 2005.

[143] B.N. Goertzel, C. Pennachin, L.S. Coelho, B. Gurbaxani, E.M. Maloney, and J.F. Jones: **Combination of single nucleotide polymorphisms in neuroendocrine ef-**

fector and receptor genes predict chronic fatigue syndrome, *Pharmacogenomics*, 7:475-483, 2006.

[144] J. Listgarten *et al.*: **Predictive models for breast cancer susceptibility from multiple single nucleotide polymorphisms**, *Clin Cancer Res*, 10:2725-2737, 2004.

[145] G. Üsünkar, S. Özögür-Akyüz, G.W. Weber, C.M. Friedrich, and Y.A. Son: **Selection of representative SNP Sets for genome-wide association studies: A metaheuristic approach**, *Optimization Lett*, 6(6):1207-1218, 2012.

[146] Z. Meng, D.V. Zaykin, C.F. Xu, M. Wagner, and M.G. Ehm: **Selection of genetic markers for association analyses, using linkage disequilibrium and haplotypes**, *Am J Hum Genet*, 73:115-130, 2003.

[147] B. Horne and N.J. Camp: **Principal component analysis for selection of optimal SNP-sets that capture intragenic genetic variation**, *Genet Epidemiol*, 26:11-21, 2004.

[148] Y. Lee, Y. Lin, and G. Wahba: **Multicategory support vector machines, theory, and application to the classification of microarray data and satellite radiance data**, *J Amer Stat Assoc*, 99(465):67-81, 2004.

[149] T. Joachims: **Transductive inference for text classification using support vector machines**, 16^{th} *International Conference on Machine Learning* (ICML), San Francisco, USA, pp. 200-209, 1999.

[150] C.W. Hsu and C.J. Lin: **A comparison of methods for multiclass support vector machines**, *IEEE Trans Neural Netw*, 13(2):415-425, 2002.

[151] John and M. Stephen: **Interpreting principal component analyses of spatial population genetic variation**, *Nat Genet*, 40:646-649, 2008.

[152] Boas and L. Mary: **Mathematical Methods in the Physical Sciences**, 2^{nd} edn., New York: Wiley, 1983.

[153] H. Abdi and L.J. Williams: **Principal component analysis**, *Comput Stat, Wiley Interdisciplinary Rev*, 2:433-459, 2010.

[154] G. Isabelle, J. Weston, S. Barnhill, and V.N. Vapnik: **Gene selection for cancer classification using support vector machines**, *Mach Learn*, 46:389-422, 2002.

[155] Y. LeCun, J.S. Denker, and S.A. Solla: **Optimum brain damage**, *Advances in Neural Information Processing Systems 2*, Edited by Touretzky, Morgan: Kaufmann, pp 598-605, 1990.

[156] W.B. Johnson and J. Lindenstrauss: **Extensions of lipschitz mappings into a Hilbert space**, *Conference in Modern Analysis and Probability*, Providence: Amer. Math. Soc., pp. 189-206, 1984.

[157] M.D. Ritchie, L.W. Hahn, and J.H. Moore: **Power of multifactor dimensionality reduction for detecting gene - gene interactions in the presence of genotyping error, missing data, phenocopy, and genetic heterogeneity**, *Genet Epidemiol*, 24:150-157, 2003.

[158] M.D. Ritchie, L.W. Hahn, and J.H. Moore: **Multifactor dimensionality reduction software for detecting gene - gene and gene - environment interactions**, *Bioinformatics*, 19:376-382, 2003.

[159] E.R. Martin, M.D. Ritchie, L. Hahn, S. Kang, and J.H. Moore: **A novel method to identify gene-gene effects in nuclear families: the MDR-PDT**, *Genet Epidemiol*, 30:111-123, 2006.

[160] C.S. Coffey *et al.*: **An application of conditional logistic regression and multifactor dimensionality reduction for detecting gene - gene interactions on risk of myocardial infarction: The importance of model validation**, *BMC Bioinformatics*, 5:49, 2004.

[161] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan: **Automatic subspace clustering of high dimensional data for data mining applications**, *ACM-SIGMOD Conf. on the Management of Data*, pp. 94-105, 1998.

[162] S. Basu, I. Davidson, and K. Wagstaff: **Constrained clustering: advances in algorithms**, *Theory and Applications: Data Mining and Knowledge Discovery*, Chapman & Hall/CRC, vol. 3, 2008.

[163] O. Chapelle, B. Schlkopf, and A. Zien, **Semi-Supervised Learning**, *MIT Press*, 2006.

[164] Y.-M. Cheung: **k*-means: a new generalized k-means clustering algorithm**, *Pattern Recognition Letters*, 24:2883-2893, 2003.

[165] Clustering datasets: `http://cs.joensuu.fi/sipu/datasets/`.

[166] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu: **A density-based algorithm for discovering clusters in large spatial data sets with noise**, 2^{nd} *Int. Conf. on Knowledge Discovery and Data Mining*, Portland, OR, USA, pp. 226-231, 1996.

[167] S. Guha, R. Rastogi, and K. Shim: **CURE: An efficient clustering algorithm for large data sets**, *ACM SIGMOD Conference*, 1998.

[168] S. Guha, R. Rastogi, and K. Shim: **ROCK: A robust clustering algorithm for categorical attributes**, *IEEE Conference on Data Engineering*, 1999.

[169] A. Hinneburg and D. Keim: **An efficient approach to clustering in large multimedia data sets with noise**, 4^{th} *International Conference on Knowledge Discovery and Data Mining*, pp. 58-65, 1998.

[170] E. Horowitz, S. Sahni, and S. Rajasekaran, **Computer Algorithms**, *Silicon Press*, 2008.

[171] Z. Huang: **Extensions to the k-means algorithm for clustering large data sets with categorical values**, *Data Mining and Knowledge Discovery*, 2:283-304, 1998.

[172] A.K. Jain, M.N. Murty, and P.J. Flyn: **Data clustering: a review**, *ACM Computing Surveys*, 31(3), 1999.

[173] G. Karypis, E.H. Han, and V. Kumar: **CHAMELEON: A hierarchical clustering algorithm using dynamic modeling**, *Computer* 32(8):68-75, 1999.

[174] H. Kashima, J. Hu, B. Ray, and M. Singh: **K-means clustering of proportional data using L1 distance**, *Internat. Conf. on Pattern Recognition*, pp. 1-4, 2008.

[175] T. Lange, M.H. Law, A.K. Jain, and J. Buhmann: **Learning with constrained and unlabelled data**, *IEEE Comput. Soc. Conf. Comput. Vision Pattern Recognition*, 1:730-737, 2005.

[176] F. Olken and D. Rotem: **Random sampling from databases: a survey**, *Statistics and Computing*, 5(1):25-42, 1995.

[177] S. Rajasekaran: **Selection algorithms for parallel disk systems**, *Journal of Parallel and Distributed Computing*, 64(4):536-544, 2001.

[178] S. Rajasekaran: **Efficient parallel hierarchical clustering algorithms**, *IEEE Transactions on Parallel and Distributed Systems* 16(6), 2005.

[179] S. Rajasekaran and J.H. Reif: **Optimal and sub-logarithmic time randomized parallel sorting algorithms**, *SIAM Journal on Computing*, 18(3):594-607, 1989.

[180] M. Salter-Townshend and T.B. Murphy: **Variational Bayesian inference for the latent position cluster model for network data**, *Computational Statistics and Data Analysis*, 57(1):661, 2013.

[181] C. Sheikholeslami, S. Chatterjee, and A. Zhang: **WaveCluster: A multi resolution clustering approach for very large spatial data set**, 24^{th} *VLDB Conf.*, 1998.

[182] Y.D. Smet, P. Nemery, and R. Selvaraj: **An exact algorithm for the multicriteria ordered clustering problem**, *Omega*, 40(6):861, 2012.

[183] B.G. Tabachnick and L.S. Fidell: **Using multivariate statistics**, *Allyn and Bacon*, Fifth Edition, Boston, 2007.

[184] W. Wang, J. Yang, and R. Muntz: **STING: A statistical information grid approach to spatial data mining**, 23^{rd} *VLDB Conference*, Athens, Greece, 1997.

[185] X. Yi and Y. Zhang: **Equally contributory privacy-preserving k-means clustering over vertically partitioned data**, *Information Systems*, 38(1):97, 2012.

[186] T. Zhang, R. Ramakrishnan, and M. Linvy: **BIRCH: An efficient data clustering method for very large data sets**, *Data Mining and Knowledge Discovery*, 1(2):141-182, 1997.

[187] P. Christen and K. Goiser: **Quality and complexity measures for data linkage and deduplication**, *Quality Measures in Data Mining*, Volume 43, Edited by F. Guillet, Hamilton H. New York: Springer, pp. 127151, 2007.

[188] **Data integration**, *http://en.wikipedia.org/wiki/Data_integration.*

[189] J. Doak: **An Evaluation of Feature Selection Methods and Their Application to Computer Security**, *Technical report*, Univ. of California at Davis, Dept. Computer Science, 1992.

[190] A.K. Elmagarmid, P.G. Ipeirotis, and V.S. Verykios: **Duplicate Record Detection: A Survey**, *IEEE Trans Knowl Data Eng*, 19:116, 2007.

[191] FSelector: **Optimization by simulated annealing**, *http://cran.r-project.org/web/packages/FSelector/FSelector.pdf.*

[192] S. Geman, E. Bienenstock, and R. Doursat: **Neural networks and the bias/variance dilemma**, *Neural Computation*, 4(1):1-58, 1992.

[193] C.-W. Hsu and C.-J. Lin: **A Comparison of Methods for Multiclass Support Vector Machines**, *IEEE Transactions on Neural Networks*, 2002.

[194] G. Isabelle, J. Weston, S. Barnhill, and V.N. Vapnik: **Gene Selection for Cancer Classification using Support Vector Machines**, *Machine Learning*, 46:389-422, 2002.

[195] A. Jain and D. Zongker: **Feature selection: evaluation, application, and small sample performance**, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19:153-158, 1997.

[196] T. Jirapech-Umpai and S. Aitken: **Feature selection and classification for microarray data analysis: Evolutionary methods for identifying predictive genes**, *BMC Bioinformatics*, 6:148, 2005.

[197] T. Joachims: **Transductive Inference for Text Classification using Support Vector Machines**, *International Conference on Machine Learning* (ICML), pp. 200-209, 1999.

[198] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi: **Optimization by simulated annealing**, *Science*, 220(4598):671-680, 1983.

[199] J. Kittler: **Feature set search algorithm**, *Pattern Recognition and Signal Processing*, Sijthoff and Noordhoff, Alphen aan den Rijn, Netherlands, pp.41-60, 1978.

[200] R. Kohavi and G.H. John: **Wrappers for Feature Subset Selection**, *Artificial Intelligence*, 97(1-2):273-324, 1997.

[201] M. Kudo and J. Sklansky: **Comparison of algorithms that select features for pattern classifiers**, *Pattern Recognition*, 33:25-41, 2000.

[202] Y. Lee, Y. Lin and G. Wahba: **Multicategory Support Vector Machines, Theory, and Application to the Classification of Microarray Data and Satellite Radiance Data**, *J. Amer. Statist. Assoc.*, 99(465)67-81, 2004.

[203] H. Liu and H. Motoda: **Feature Selection for Knowledge Discovery and Data Mining**, *Boston: Kluwer Academic*, 1998.

[204] C.S. Liu, G. Pilania, C. Wang, and R. Ramprasad: **How critical are the van der Waals interactions in polymer crystals?**, *J. Phys. Chem. C*, 116:9347, 2012.

[205] T. Mi, S. Rajasekaran, and R. Aseltine: **Efficient algorithms for fast integration on large data sets from multiple sources**, *BMC Med Inform Decis Mak*, 12:59, 2012.

[206] D. Mitra, F. Romeo, and A.S. Vincentelli, **Convergence and Finite-Time Behavior of Simulated Annealing**, *Advances in Applied Probability*, 1986.

[207] P.M. Narendra and K.A. Fukunaga: **Branch and Bound Algorithm for Feature Subset Selection**, *IEEE Trans. Computer*, 26(9):917-922, 1977.

[208] P. Pudil, J. Novovicova and J. Kittler: **Floating search methods in feature selection**, *Pattern Recognition Letters*, 15:1119-1125, 1994.

[209] S. Rajasekaran: **On Simulated Annealing and Nested Annealing**, *Journal of Global Optimization*, 16(1):43-56, 2000.

[210] J.S. Russell and N. Peter: **Artificial Intelligence: A Modern Approach (2nd ed.)**, *Prentice Hall*, Upper Saddle River, NJ, USA, pp. 111-114, 2003.

[211] C. Saunders, A. Gammerman, and V. Vovk: **Ridge Regression Learning Algorithm in Dual Variables**, 15^{th} *International Conference on Machine Learning*, Madison, WI, USA, pp. 515-521, 1998.

[212] Y. Sun, S.A. Boggs, and R. Ramprasad: **The intrinsic electrical breakdown strength of insulators from first principles**, *Appl. Phys. Lett*, 101(13):290-296, 2012.

[213] A.A. Tikhonov and V.Y. Arsenin: **Solutions of ill-posed problems**, *New York: John Wiley*, 1977.

[214] C.C. Wang, G. Pilania, and R. Ramprasad, **Dielectric properties of carbon, silicon and germanium based polymers: A first principles study**, *Phys. Rev. B*, under review.

[215] W.E. Winkler: **Overview of Record Linkage and Current Research Directions**, *[http://www.census.gov/srd/papers/pdf/rrs2006-02.pdf].*

[216] W.E. Winkler: **Improved Decision Rules In The Fellegi-Sunter Model Of Record Linkage**, *Survey Research Methods, American Statistical Association*, Alexandria, VA: American Statistical Association, 1:274279, 1993.

[217] S. Saha *et al.*: **Efficient techniques for genotype-phenotype correlational analysis**, *BMC medical informatics and decision making*, 13:41, 2013.

[218] S. Rajasekaran and S. Saha: **A Novel Deterministic Sampling Technique to Speedup Clustering Algorithms**, *Advanced Data Mining and Applications* (ADMA), Hangzhou, China, pp. 34-46, 2013.

[219] S. Salzberg *et al.*: **Microbial gene identification using interpolated markov models**, *Nucleic Acids Res.*, 26:544548, 1998.

[220] M. Ben-Bassat: **Pattern recognition and reduction of dimensionality**, *P. Krishnaiah and L. Kanal, (eds.) Handbook of Statistics II*, Vol. 1. North-Holland, Amsterdam, pp. 773-791, 1982.

[221] M. Hall: **Correlation-based feature selection for machine learning**, *PhD Thesis*, Department of Computer Science, Waikato University, New Zealand, 1999.