7-19-2016

Computational Methods for Gene Expression and Genomic Sequence Analysis

Nam Sy Vo

COMPUTATIONAL METHODS FOR GENE EXPRESSION AND GENOMIC SEQUENCE ANALYSIS

by

Nam Sy Vo

A Dissertation

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

Major: Computer Science

The University of Memphis

August 2016

Dedication

I would like to dedicate this dissertation to my dear parents, Le Thi Phuong Thao and Vo Quoc Hung, who I am always greatly indebted to, who are always proud of me, always ready to support me everything in my life with an invaluable love; to my dear younger brother, Vo Tuan Linh, who are always beside me from my childhood; to my dear paternal grandparents, Dinh Thi Sen and Vo Dinh Yen, who always encourage me in everything I do; to my dear maternal grandparents, Nguyen Thi Huong and Le Duc Tinh, who are always in my memory with their wish of my successfulness and happiness. They are always the greatest source of motivation and inspiration throughout my life, forever.

Lời đề tặng

Tôi dành tặng luận án này cho bố mẹ tôi, Lê Thị Phương Thảo và Võ Quốc Hùng, những người tôi luôn mang ơn, những người luôn tự hào về tôi, luôn sẵn sàng hỗ trợ tôi mọi điều trong cuộc sống với một tình thương vô giá; cho em trai yêu quý của tôi, Võ Tuấn Linh, người luôn bên cạnh tôi từ thuở ấu thơ; cho ông bà nội thân thương của tôi, Đinh Thị Sen và Võ Đình Yên, những người luôn động viên khuyến khích tôi trong mọi điều tôi làm; cho ông bà ngoại thân yêu đã khuất của tôi, Nguyễn Thị Hương và Lê Đức Tính, những người luôn ở trong ký ức tôi với mong ước lớn lao về thành công và hạnh phúc cho tôi. Tất cả luôn là nguồn động viên và cảm hứng lớn nhất cho tôi trong suốt cuộc đời này, mãi mãi.

Acknowledgements

This dissertation marked a completion of my long-time study at the University of Memphis. It would not have been possible without the help from many people.

First of all, I would like to acknowledge Dr. Vinhthuy Phan, my major advisor, for everything he did for me. He gave me a lot of useful advice, helped me thoroughly understand a lot of problems not only in this dissertation but also in general background, and taught me a lot of necessary things needed to become a successful independent researcher. I would like to thank him for spending hours of his valuable time with terrific patience to support me in solving problems, not only in this dissertation but also in my life in the U.S.. I am deeply thankful for everything he has done for me.

It has been a pleasure to work with my co-authors and labmates Quang Minh Tran, Diem-Trang Pham, Shanshan Gao, Kevin O'Kello, Nobal Niraula, and Sindhu Priya Oggu, who worked with me and supported me a lot during my research towards this dissertation. I would like to send a special thank to Dr. Ramin Homayouni for his collaboration and his support during my study in the university. I would like to thank Dr. Thomas T. Sutter who worked with me in an important part of this dissertation. I would like to thank Ashutosh K. Pandey who supported me in my early work towards this dissertation. My special thanks go to my committee members, Drs. Ramin Homayouni, Vasile Rus, Lan Wang, Ebergener O. George, and Chase Qishi Wu for their time of reading my doctoral dissertation and master's thesis with lots of useful suggestions.

Last but not least, many thanks to my teachers, friends, and relatives, who have taught and helped me a lot on my long-time journey to this stage. A special thank to Dr. Huynh Quyet Thang, my master's thesis advisor, who encouraged me to start doing research when I was in Viet Nam. Especially, I would like to thank Hai Nguyen and his family members who supported me a lot in my early life in the U.S.. I am very grateful to all my real-life-friends and social-network-friends for their support and encouragement on my whole journey. Thank you so much, this dissertation is for all of you.

Abstract

Advances in technologies currently produce more and more cost-effective, high-throughput, and large-scale biological data. As a result, there is an urgent need for developing efficient computational methods for analyzing these massive data. In this dissertation, we introduce methods to address several important issues in gene expression and genomic sequence analysis, two of the most important areas in bioinformatics.

Firstly, we introduce a novel approach to predicting patterns of gene response to multiple treatments in case of small sample size. Researchers are increasingly interested in experiments with many treatments such as chemicals compounds or drug doses. However, due to cost, many experiments do not have large enough samples, making it difficult for conventional methods to predict patterns of gene response. Here we introduce an approach which exploited dependencies of pairwise comparisons outcomes and resampling techniques to predict true patterns of gene response in case of insufficient samples. This approach deduced more and better functionally enriched gene clusters than conventional methods. Our approach is therefore useful for multiple-treatment studies which have small sample size or contain highly variantly expressed genes.

Secondly, we introduce a novel method for aligning short reads, which are DNA fragments extracted across genomes of individuals, to reference genomes. Results from short read alignment can be used for many studies such as measuring gene expression or detecting genetic variants. Here we introduce a method which employed an iterated randomized algorithm based on FM-index, an efficient data structure for full-text indexing, to align reads to the reference. This method improved alignment performance across a wide range of read lengths and error rates compared to several popular methods, making it a good choice for community to perform short read alignment.

Finally, we introduce a novel approach to detecting genetic variants such as SNPs (single nucleotide polymorphisms) or INDELs (insertions/deletions). This study has great significance in a wide range of areas, from bioinformatics and genetic research to medical

field. For example, one can predict how genomic changes are related to phenotype in their organism of interest, or associate genetic changes to disease risk or medical treatment efficacy. Here we introduce a method which leveraged known genetic variants existing in well-established databases to improve accuracy of detecting variants. This method had higher accuracy than several state-of-the-art methods in many cases, especially for detecting INDELs. Our method therefore has potential to be useful in research and clinical applications which rely on identifying genetic variants accurately.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

This dissertation studies several problems in bioinformatics, an interdisciplinary field devoted to the analysis and interpretation of biological data using computational methods. Biological data are data collected from biological sources such as DNA, RNA or proteins. The primary goal of bioinformatics is to increase the understanding of biological processes. This field has evolved extremely rapidly in recent years due to the explosive growth of biological data generated by the scientific community.

Advances in technologies such as microarrays and more recent next-generation sequencing make it possible to provide cost-effective, high-throughput, and large-scale biological data of interests. This huger and huger amount of data facilitates a wide range of bioinformatics research such as gene expression analysis, sequence analysis, structural bioinformatics, systems biology, biological databases and data mining, and many other fields of interests [1, 2, 3, 4]. As a result, there is an urgent need for developing efficient computational methods to tackle the challenges of analyzing these massive data.

In this dissertation, we develop methods and tools to address several important issues in gene expression and genomic sequence analysis, two of the most important areas of bioinformatics. A brief introduction of these areas will be described in Section 1.1. Among many problems in these areas, we focus on the following: (1) representing and analyzing patterns of gene response to multiple treatments, (2) aligning short DNA sequences to reference genomes, and (3) detecting genomic variants. A brief introduction of these problems will be described in Section 1.2. To address these problems, we introduce novel methods and tools which employed techniques from a variety of fields - information retrieval, data mining, and statistics. Main contributions of these methods and their experimental results on both simulated and real data will be described in Section 1.3. Finally, Section 1.4 will give a brief outline of the dissertation.

1.1 Background

1.1.1 Gene Expression Analysis

Gene expression analysis is the study of functional gene products such as functional RNA species or protein products[1]. This includes obtaining, analyzing, and understanding gene expression data. Gene expression data are obtained from measurements of gene expression levels, i.e., relative abundances of RNAs within cells in living organisms, which show the activities of genes and therefore of living organisms. Gene expression levels can be measured by many techniques such as DNA microarrays [5, 6, 7] or more recent RNA-Seq technology [8, 9, 10]. As an illustration, Figure 1 (left) shows two DNA microarrays, from which gene expression values can be obtained. These values then can be represented in some ways, such as heatmaps (Figure 1, right). Gene expression data are then analyzed and interpreted using computational or statistical methods to detect differential expression, to generate gene clusters, to predict sample characteristics, or many other fields of interests [11, 12].

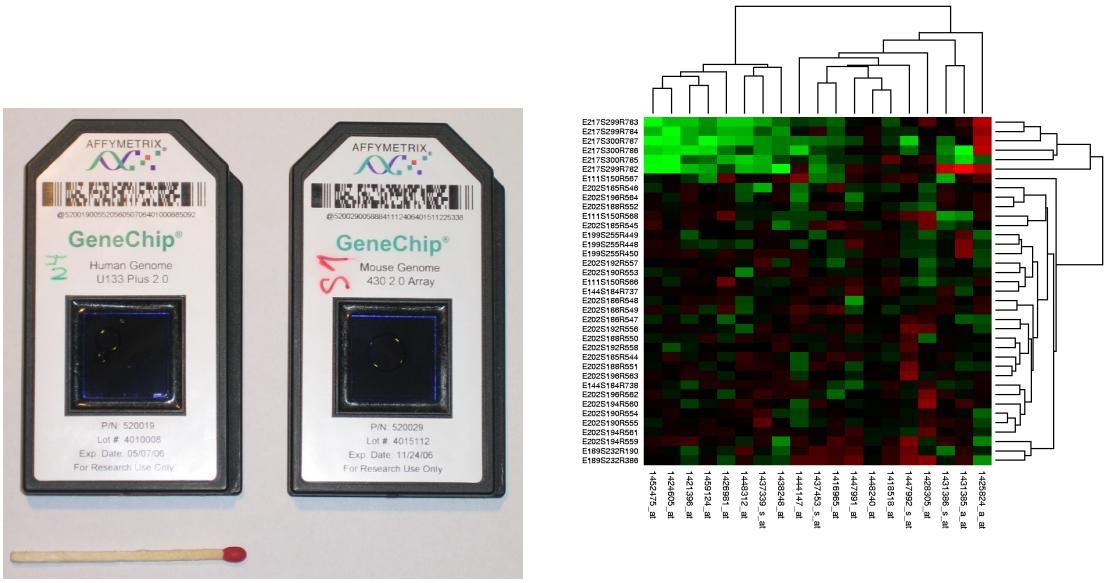

Figure 1: Left: two Affymetrix GeneChips microarrays. Right: a heat map, which represents gene expression values obtained from DNA microarray experiments. (taken from https://en.wikipedia.org/wiki/DNA_microarray, accessed on June 6th, 2016)

[1] http://www.nature.com/subjects/gene-expression-analysis

Gene expression studies typically aim at understanding genetic mechanisms that affect cells at different time points, drug doses, types of drugs, etc., or any combination of these treatments. The analysis of gene expression therefore plays an important role in bioinformatics, biological and medical research. Typically, people apply traditional techniques such as clustering or classification to analyze gene expression data. As showing in Figure 1 (right), we can use Hierarchical Clustering to obtain a number of clusters from microarray data. The hypothesis is, genes sharing the same clusters are also sharing related biological functions. Then the next step is interpreting such clusters to understand their biological meanings. Although advances of technologies (e.g., RNA-Seq) facilitate gene expression studies significantly, there are still many fundamental challenges encountered in analyzing and understanding gene expression data. This dissertation aims to address several challenging problems in this area.

1.1.2 Genomic Sequence Analysis

Genomic sequence analysis is the study of analyzing genomic sequences such as DNA, RNA of individuals of species[2]. This study includes obtaining, analyzing, and understanding genomic sequence data. Genomic sequence data can be obtained from sequencing of DNA using many techniques such as shotgun sequencing [13] or more recent next-generation sequencing [14]. As an illustration, Figure 2 (left) shows a sequencing system, from which extracted fragments of DNA or RNA, so-called *short-reads*, can be obtained. These short-reads then can be analyzed in several ways, such as mapping them to a reference genome from the same species to determine their locations on the reference (Figure 2, right). The short-reads can be analyzed using computational or statistical methods to get the original genomes, to detect genomic variations, to obtain gene expression data, and many other fields of interests [15].

Genomic sequence analysis typically aims at understanding features, function, structure, or evolution of genomes of individuals. The analysis of genomic sequences

[2] http://www.nature.com/subjects/genomic-analysis

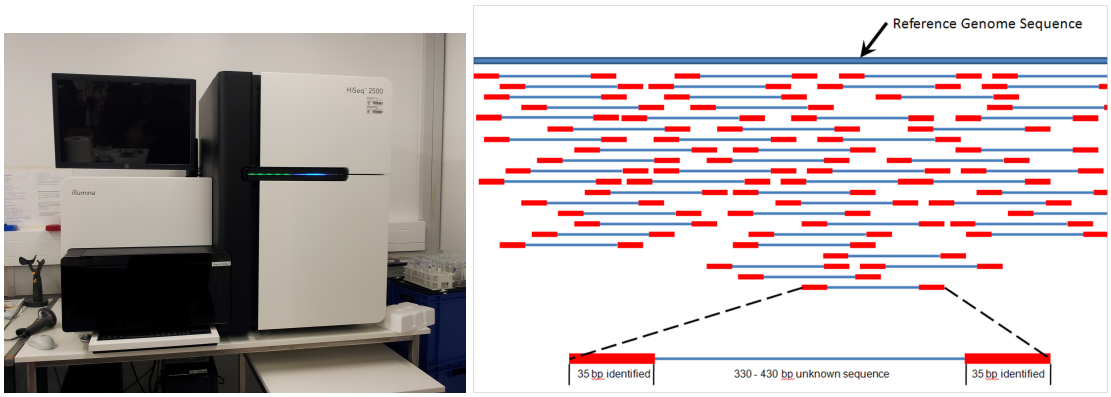

Figure 2: Left: an Illumina HiSeq 2500 sequencer. Right: read mapping, which represents alignment to a reference genome of multiple fragmented and (maybe) overlapping short sequences obtained from DNA sequencing experiments. (taken from https://en.wikipedia.org/wiki/DNA_sequencing, accessed on June 14th, 2016)

therefore plays an important role in bioinformatics, biological, and medical research. Typically, people exploit several methods such as sequence alignment, sequence assembly, searches against biological databases to analyze such data. As showing in Figure 2 (right), we can use sequence alignment techniques to map short-reads to a reference genome sequence. This task is known as *short-read alignment*, one of the most important tasks for downstream analyses such as detecting genomic variants or measuring gene expression levels. For example, the aligned reads then can be used to determine the genomic variations, i.e., the differences between genomes of individuals in population. This task is known as *genomic variant detection*, so-called *variant calling*, which has great significance in genomic research as well as in medical field. For example, one can predict how genomic changes are related to phenotype in their organism of interest, or associate genetic changes to disease risk, medical treatment efficacy or other traits of interest. Although advances of technologies (e.g., next-generation sequencing, or NGS) facilitate genomic sequence analysis significantly, there are still many fundamental challenges remain in analyzing and understanding genomic sequences. This dissertation aims to address several challenging problems in this area.

1.2 Research Problems

Firstly, we focus on the study of representing, analyzing and interpreting patterns of gene response to multiple treatments. While a majority of gene expression studies involve a small number of different types drugs, researchers have designed experiments with hundreds of chemicals at various doses and durations [16, 17]. In such studies, researchers are interested in understanding not only the effects of certain drugs (compared to untreated) but also the differences and similarities among the drugs themselves.

Researchers have introduced a number of *post hoc* approaches to analyze gene expression data with multiple treatments. In this way, only k groups of gene expression values are measured and $\frac{k(k-1)}{2}$ tests are made to compare how a gene responds to all pairs of treatments. Several studies have employed ternary-digits to represent patterns of gene response [18, 19]. An example of ternary-digit pattern is 011022 that represents comparisons of 6 pairs of treatments in which "1" means two treatment are not different, and "2" ("0") means that the first treatment is statistically less than (greater than) the other treatment. Although ternary-digit patterns are concise and have been found useful in calculating the statistical significance of observed response patterns, they are harder to interpret and visualize. Instead of ternary digits, recent studies have employed directed graphs to represent patterns of gene response to all treatment pairs, such as [20, 21]. In this approach, vertices represent treatments and edges represent how a gene responds to all pairs of treatments. The authors showed that this representation made it possible to reason about the accuracy of response as a function of sample size. Moreover, representing patterns of gene response as directed graphs makes it possible to visualize how genes respond to all treatment pairs and to identify effectively primary responses and secondary responses of any particular subset of treatments of interest.

One of the most challenging problems in analyzing gene expression data is that the experiment contains only few samples (or replicates). If the sample size is too small, the true pattern of highly variantly expressed genes cannot be captured accurately and

consequently it is very hard for downstream analyses to predict true patterns of gene response. Researchers have recognized that sufficient large numbers of samples are needed to account for biological variation regardless of the underlying technologies (DNA microarrays or RNA-Seq) [22, 23, 24, 25, 26, 27] and even concluded that sample size should be calculated to meet the objectives of the specific aims of each study [28]. Nevertheless, due to cost, practical experimental designs tend to have small sample sizes that make it hard for conventional methods to predict true patterns of gene response.

This dissertation aims to address the problem of accurately predicting patterns of gene response to multiple treatments in case of small sample size.

Secondly, we focus on the study of determining alignment of short-reads to the reference genome. The basic motivation of this work is that genomes within the same species are very similar, therefore a reference genome can facilitate analysis of genomes of individuals using their short-reads. Alignment of short-reads to reference genomes is a critical step in many applications which rely on the data obtained by NGS technologies. Result from this process can be used for many areas in bioinformatics such as determining gene expression or detecting genomic variants. Mapping DNA or RNA short-reads to a reference genome or transcriptome is the first step of the standard approach to measuring gene expression or identifying genetic variants using NGS data.

Researchers have introduced many approaches and methods along with software packages for the short read alignment. Most of them use *seed-and-extend* strategy to efficiently align reads to the reference. Several data structures such as hash tables, q-grams, suffix trees/suffix arrays, or FM-index [29], a compressed full-text substring index based on the Burrows-Wheeler Transform (BWT) [30], are used to speed up the process of searching for *seeds* [31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43]. For example, SOAP [32] and BFAST [33] use the hash tables while Bowtie [35] and BWA [36] use the FM-index to build a permanent index of the reference. Additional heuristics are also used to enhance efficiency. For example, GASSST [39] uses a filtering technique

to reduce noisy seeds while CUSHAW2 [41] uses several filtering steps to quickly identify true seeds. Alignment algorithms such as the Needleman-Wunsch or the Smith-Waterman (SW) are employed in the *extend* phase to obtain the final alignments.

Nevertheless, there are still many challenges with current methods and tools for accurately aligning short reads to the reference. Current algorithms still struggle to identify correct alignment due to divergence sequencing error, complicated genetic variants such as insertions or deletions or other types of structural variations, repetitive regions of reference genomes, or even misassemblies and incompleteness of the reference. Although advances in technologies produce longer and more accurate reads, it is still difficult to improve accuracy of aligning reads to the reference. For example, the complex genomic variants and the repetitive regions of the reference genomes are still sources of erroneous alignment despite a lot of effort in developing advanced read aligners.

This dissertation aims to address the problem of accurately aligning short reads to the reference genome across a wide range of read lengths and base error rates.

Thirdly, we focus on the study of genomic variant detection, so-called *variant calling*. Detection of genomic variants has great significance in genomic research as well as in medical field. For example, one can predict how genomic changes are related to phenotype in their organism of interest, or associate genetic changes to disease risk, medical treatment efficacy or other traits of interest. With the development of next-generation sequencing technologies, more and more studies now focus on detecting genomic variants using NGS data.

Currently, the typical approach for most methods to detect genomic variants using NGS data is based on analyzing initial aligned reads of unknown genomes to the reference genome, which are produced by an external read aligner. Researchers have developed many variant callers based on statistical methods such as Bayesian method or logistic regression models [44, 45, 46, 47, 48, 49, 50, 51, 52]. Most of variant callers use post-alignment processing strategies to correct erroneous alignment due to

insertions/deletions. Some of recent variant callers employed local-assembly or global-assembly to improve accuracy of variant detection [53, 54, 55].

There are several disadvantages of this conventional approach. First, current aligners typically do not fully take into account the correlation between reads, which does not allow variant callers to take full advantage of input data to support detecting accurate and inaccurate variants. More generally, the alignment step was done separately and independently from variant calling step, which limits incorporating information from one step to another or vice versa. Second, current variant callers typically do not fully take into account knowledge of existing variants, which can help identify common variants more accurately. Consequently, current methods for variant calling are expensive as they require a high sequencing coverage, i.e. large numbers of reads, to be able to detect variants accurately as well as identify many variants as possible. Additionally, the determination of variants, which is performed through many separate steps, requires a careful selection of a diverse set of tools and needs more human intervention, which can be error-prone.

This dissertation aims to address the problem of accurately detecting genomic variants, especially insertions/deletions.

1.3 Main Contributions

Firstly, we have introduced two novel methods to predict true patterns of gene response to multiple treatments in case of small sample size. These methods employed directed graphs and orderable sets, respectively, to represent gene patterns and exploit their certain properties using dependencies of pairwise comparison outcomes to deduce more and better functionally enriched gene clusters in case of small sample size. This work has been introduced in our publications from 2011 to 2014 [56, 57, 58, 59, 60] (the first method), [61, 62] (the second method). Two web tools mDAG[3] and PA[4] were also implemented for analyzing patterns of gene expression with multiple treatments.

[3] cetus.cs.memphis.edu/mdag

[4] cetus.cs.memphis.edu/pa

In the first method, we employed directed graphs as representation of gene expression patterns. We then showed that true patterns of gene expression can be predicted by a pre-cautious use of synthetic data. We exploited the relationship between sample size and a graph property known as *contractibility* together with synthetic replicates to predict true patterns of gene response to treatments. Further, we showed how to use entropy to measure the certainty of such predictions. Using gene expression data from rats' liver cells, we showed that this approach uncovered subtle interactions in response patterns and resulted in more and better functionally enriched gene clusters. Directed graphs can benefit subsequent functional analysis as well.

In the second method, we showed that true patterns of response must be *orderable sets*. This property enables the exploitation of dependencies among outcomes of statistical tests that constitute patterns of gene response. Specifically, instead of obeying the convention of fixing the level of significance α at 0.05, we showed that α might be varied intelligently gene-by-gene to predict true patterns more accurately. This strategy produced clusters that had more than twice the amount of functional enrichment. A comparison to hierarchical clustering showed that our method produced clusters that were much more functionally enriched. Our method also revealed transcription factors grouped together, which were misgrouped by hierarchical clustering, that play an important role in many pharmacological activities. Furthermore, motivated by the fact that a gene might participate in multiple biological functions or be involved in multiple molecular processes, we showed how to resample experimental replicates to predict and assign multiple patterns to each gene. We showed that this approach of assigning multiple patterns to genes further increased functional enrichment many folds.

The above methods are especially useful for gene-expression studies designed to explore functional similarities and differences of similar chemicals. e.g, drug designs. They have been found useful in identifying structure-activity relationships among drugs and differentiating genes sharing similar functional pathways.

Secondly, we have introduced a novel method to improve accuracy of short-read alignment across a wide range of read lengths and base error rates. This method exploited an iterated randomized searching algorithm together with two FM indexes for bidirectional searching to improve alignment performance. The method has been implemented in the tool RandAL[5]. We also have investigated several measures of sequence complexity which can be used to predict performance of short read aligners. This work has been introduced in our publications from 2013 to 2015 [63, 64, 65, 66].

In this method, we employed two FM indices for efficient bidirectional (exact) substring matching between reads and reference genomes. To deal with inexact matching (i.e. allowing gaps), first, we find common substrings, so-called *seeds* between reads and the reference genome. Then, these seeds are extended to complete alignments based on a bounded threshold on the edit distance. We use a special pruning mechanism to shorten vastly the running time of computing edit distances in a vast majority of cases. We also use an iterated randomized algorithm for aligning reads to the reference. This increases the probability of finding seeds quickly and enables us determine methodologically important parameters to speed up the entire alignment process. Our results showed that our algorithm performed consistently well on a wide range of read lengths across several bacterial and eukaryotic genomes. The alignment quality of our method was better or generally as good as that of all compared methods.

Thirdly, we have introduced a novel method to improve accuracy of calling genomic variants, especially insertions/deletions. This method leverages existing genomic variants, such as those provided by the 1000 Genomes Project, during read alignment and variant calling process to improve variant calling performance. We have further improved our previous method of read alignment and integrated it as part of the whole variant

[5] https://github.com/namsyvo/RandAL

calling process. Part of this work has been introduced in our recent publication [67]. The method is implementing in the tool IVC[6].

In this method, we designed a special reference genome called multi-genome, which combines a given variant profile and a reference genome into a unified representation, to facilitate using existing variants. We then design a mixture technique, which combines hashing, modified FM index, and an asymmetric local alignment, to align reads to the reference multi-genome and to call variants simultaneously. We utilize Bayesian method to identify the called variants and calculate their qualities, which indicate the probability of called variants to be correct. Our evaluation on both simulated and real data showed that this approach was able to detect variants, especially insertions/deletions, with low coverage data more accurately than several state-of-the-art methods such as GATK and SAMtools. High accuracy at low coverage can help reduce cost of calling variants significantly. The integration also run the whole variant calling process in a faster manner while still using moderate memory. Finally, this approach helps to reduce errors caused by human intervention.

1.4 Outline of the Dissertation

In the rest of this dissertation, we describe our work in detail. Chapter 2 introduces our work on gene expression pattern analysis. Chapter 3 describes our work on read alignment. Chapter 4 introduces our work on variant calling. Finally, Chapter 5 summaries our contributions as well as describes future directions of the work. Appendices are also included to describe related software packages in more detail.

Although the research problems are related to each other, the chapters of the dissertation are structured in a self-contained way. One can read each Chapter 2, 3, and 4 independently after getting the general description of the research problems from this chapter. Key contributions and future directions of our work can be found in Chapter 5. More details of the tools we have developed can be found in the Appendices.

---

[6] https://github.com/namsyvo/IVC

Chapter 2

Gene Expression Analysis

Gene expression analysis is the study of functional gene products such as functional RNA species or protein products. Gene expression studies typically aim at understanding genetic mechanisms that affect cells at different time points, diseases, drug doses, types of drugs, etc., or any combination of these treatments. These are done by analyzing gene expression data, which are obtained from measurements of gene expression levels. The analysis of gene expression therefore plays an important role in bioinformatics and biomedical research.

Advances of high-throughput technologies such as DNA microarrays and more recent RNA-Seq enable the measurement of gene expressions for a large numbers of genes simultaneously. DNA microarray is a well-established technology which has been widely used for a long time [68, 69, 70]. A DNA microarray is a collection of microscopic DNA spots attached to a solid surface. Each DNA spot contains certain moles (around 10^{12} moles) of a specific DNA sequence, known as *probes*, which can be a short section of a gene. Nucleic acid sequences in the target (of which we need to measure gene expression levels) are put into microarrays for *probe-target hybridization*. Gene expression data are calculated based on detecting and quantifying relative abundance of such nucleic acid sequences. Recently, RNA-Seq technology has emerged as a promising alternative technology. This technology allows us to extract fragments of RNAs from samples. Then the standard approach of measuring gene expression levels using RNA-Seq is mapping these fragments to a reference genome or transcriptome come from the same species. Then gene expression data are calculated based on anayzing reads that are mapped to particular genes in the genome or transcriptome. The RNA-Seq now has become a standard part of many studies in the life sciences research [71, 72].

With the increasing amount of high-throughput gene expression data, there have been a great amount of work dedicated to the generation, understanding and analysis of

such data. This work includes statistical methods that help practitioners calculating sample size, selecting differentially expressed genes, and interpreting results meaningfully; and computational methods that help clustering and classifying gene expression data. Although advances of technologies facilitate gene expression studies significantly, there are still many fundamental challenges encountered in analyzing and understanding gene expression data. Therefore it is still desirable to develop efficient and effective methods for gene expression analysis.

In this chapter, we describe our work in gene expression analysis which focuses on analyzing patterns of gene response to multiple treatments. This work has been published in [56, 57, 58, 59, 60, 61, 62]. In Section 2.1 we describe an overview of the problem, including a formal definition of gene patterns. In Sections 2.2, 2.3, and 2.4, we introduce our methods and tools for representing, analyzing and interpreting such patterns, with the emphasis on predicting true patterns of gene response to treatments in case of insufficient sample size. Finally, Section 2.5 provides conclusion and future directions of our work.

2.1 Analysis of Gene Expression Data with Multiple Treatments

The analysis of gene expression data with multiple treatments is increasingly studied in gene expression analysis. The basic motivation of this problem is that researchers are increasingly interested in experimental designs with varying dose levels and/or multiple pharmacologically related chemicals, especially those that deal with hundreds of chemicals compounds at various doses and durations [16, 17]. In such studies, people are interested in understanding not only the effects of certain drugs (compared to untreated) but also the differences (similarities) among the drugs themselves. Many approaches have been introduced for analyzing, interpreting, and visualizing how genes respond to multiple treatments.

In analyzing gene expression data that contain multiple treatments, one approach is to employ pairwise comparisons to determine the relative orders of genes responding to treatment pairs [18, 73, 19, 20, 21]. Pairwise comparison of gene expression among

selected pairs of genes can be used to create two-gene predictors with simple decision rules for classification of expression profiles [74]. In the simplest case of having two treatment groups (e.g. treatment t versus untreated (control)), pairwise comparisons would identify genes that are up-regulated, down-regulated, or not responded by t. But as the number of different treatments increases, it becomes harder to interpret patterns observed from pairwise comparisons. To represent complex patterns of outcomes produced by pairwise comparisons, researchers have used several representations such as ternary digits [18, 19, 21] or directed graphs [20]. Furthermore, instead of making $\binom{n}{2}$ measurements (n is the number of treatments), researchers have proposed to make only $\Theta(n)$ measurements and use *post hoc* pairwise comparisons to derive gene response to all treatment pairs [18, 75].

Although many techniques and tools have been developed, obtaining reliable results from experiments that contain only few samples is still challenging. Researchers have recognized that sufficient large numbers of samples are needed to account for biological variation regardless of the underlying technologies [23, 25, 22, 24, 26, 6, 27]. One even concluded that sample size should be calculated to meet the objectives of the specific aims of each study [28]. However, due to cost, many practical experimental designs do not have large enough sample sizes, making it difficult for conventional methods to capture accurately gene response of highly variantly expressed genes. Consequently it is very hard for downstream analyses. Therefore it is desirable to develop methods to predict true patterns of gene response for this kind of experiments.

In our work, we focused on the problem of analyzing and predicting true patterns of gene response to multiple treatments. We introduced methods to represent, analyze and interpret such patterns, and most importantly, predict true patterns of gene response in case of having only few samples [57, 58, 59, 61, 62]. We exploited two types of representations, directed graphs and orderable sets, to deal with the challenge of

predicting true patterns of gene response to treatments. Several tools have been developed to facilitate this work and have been introduced as application papers [56, 58, 60].

2.1.1 Using Pairwise Comparisons to Determine Gene Response Patterns

In this study, patterns of genes are defined in terms of how the genes responds to all treatment pairs. In our approach, such patterns are obtained based on determining *relative orders* of gene response to treatments using pairwise comparisons.

Consider a gene g in an experiment with two treatment groups A and B, such as genes in Figure 3. Denote $\{A_1, ..., A_r\}$ and $\{B_1, ..., B_s\}$ be the expression measurements (replicates) of g under A and B, respectively. Then the relative order of g responding to A and B is determined based on comparing these two groups using a pairwise comparison test such as a two-sided Wilcoxon rank-sum test. The outcome of this test can be one of three possibilities:

1. $A \prec B$, when H_0: $\mu_A = \mu_B$ is rejected in favor of H_1: $\mu_A < \mu_B$, with p-value less than α (e.g. 0.05). This means g responds more to B than to A.

2. $B \prec A$, when H_0: $\mu_A = \mu_B$ is rejected in favor of H_1: $\mu_A > \mu_B$, with p-value less than α (e.g. 0.05). This means g responds more to A than to B.

	(Treatment group A)					(Treatment group B)				
Genes	A1	A2	A3	A4	A5	B1	B2	B3	B4	B5
1370911_at	569.112	506.901	400.984	426.499	371.49	390.079	370.69	352.53	320.676	314.665
1387109_at	3931.07	4568.57	5272.9	5289.13	8203.84	4419.44	3455.03	6719.6	3220.68	3391.37
1373312_at	929.204	538.498	608.049	537.233	707.31	480.931	526.941	340.242	504.769	752.605
1375845_at	185.671	216.2	177.421	286.544	146.435	99.5085	67.0297	73.9191	79.9191	104.871
1377869_at	1071.59	1126.87	919.879	1167.29	1259.39	412.692	126.124	712.964	212.517	121.256
1368004_at	335.202	368.368	368.526	354.841	430.516	373.503	356.92	321.257	378.567	376.641
1389253_at	548.446	478.572	473.165	445.512	414.733	410.053	102.592	381.675	309.877	458.157
...	...	...	...	...	...	...	...	...	...	...

Figure 3: An example of gene expression data with 2 treatment groups.

3. $A \sim B$, when H_0: $\mu_A = \mu_B$ is accepted. This means either there is no difference between A and B, or sample size is not sufficient to determine the relative order.

If the experiment has k treatment groups (including *control group*) and each group has multiple samples (replicates), we have $\frac{k(k-1)}{2}$ comparison outcomes, each of which is one of the three possibilities described above. These outcomes determine *the pattern of the gene* g. Note that the determination of such relative orders of treatment pairs with respect to g is independent of expression of other genes. Thus, the pattern of g is determined independently of patterns of other genes.

Note that in this approach, the patterns are determined using *post hoc* comparisons to save cost of analysis. First, significant differentially expressed genes are selected using statistical methods such as ANOVA or Kruskal-Wallis test. Additionally, the gene selection can be adjusted for false discovery rate using methods such as [76]. Then for each of the selected genes, relative orders of gene responses to treatment groups are determined based on pairwise comparisons.

2.2 Analyzing Gene Response Patterns with Directed Graphs

2.2.1 Introduction

The analysis of patterns of gene expression to multiple treatments can be challenging with small sample sizes, because certain responses cannot be statistically ascertained. In this section, we show that when response patterns are represented as directed graphs, additional information about true response patterns might be inferred. We will show that such representation is descriptive, visually expressive, and more importantly, can be used to infer about true patterns of gene responses using synthetic replicates. When sample size of gene expression experiments is too small, this method will help researchers to ascertain certain responses. Most of content of this section has been published in [57, 59].

In this approach, we exploit the relationship between sample size and a graph property known as *contractibility* together with synthetic replicates to predict true patterns

of gene response to treatments. Further, we will show how to use entropy to measure the certainty of such predictions. With microarray data from rats' liver cells, we showed that this approach uncovered subtle interactions in response patterns and resulted in more and better functionally enriched gene clusters. This method is especially useful for gene-expression studies designed to explore functional similarities (and differences) of similar chemicals. It has been found useful in identifying structure-activity relationships among drugs and differentiating genes sharing similar functional pathways. Directed graphs can benefit subsequent functional analysis as well.

We also introduce **PA**, a web-based tool to support this kind of analysis, including linking all genes in such patterns to several external well-known functional analysis tools. This tool has been introduced in our application paper ([58]). It was written in Python, developed based on web framework web2py[1]. It is publicly available for use at http://cetus.cs.memphis.edu/pa.

2.2.2 Method

Using Directed Graphs to Represent Patterns of Gene Response

Given gene expression data of k treatment groups, each having multiple samples, *relative orders* of gene response to treatments can be determined based on the procedure in section 2.1.1. Then, each gene is assigned a directed-graph pattern or *response graph*, in which each vertex represents a treatment group and edges represent the gene response to treatment pairs. These edges correspond to $\frac{k(k-1)}{2}$ comparison outcomes, which were defined in section 2.1.1. Given a pair of vertices A and B, if the gene responds more to A than to B, then the edge $A \rightarrow B$ is established. Conversely, if the gene responds more to B than to A, then the edge $B \rightarrow A$ is established. If there is no difference between A and B, then there is no edge between A and B in the graph.

Figure 4 shows an example of contractible and a non-contractible graph. As shown in [20], contractible patterns are more likely to be accurate, whereas non-contractible

[1] http://www.web2py.com/

patterns are inaccurate due to having too few samples. It turns out this characteristic can be used to infer true patterns of gene response to treatments. Additionally, contractible patterns can be interpreted more easily and more accurately. As an example, genes bearing the pattern shown in Figure 4B are unaffected by treatment 3 (therefore, we can put control and treatment 3 into one group) and down-regulated by treatments 1 and 2; these genes also respond more to treatment 2 than to treatment 1. Whereas with the pattern Figure 4A, we cannot say exactly how genes respond to treatment 2 and 3 compared to control.

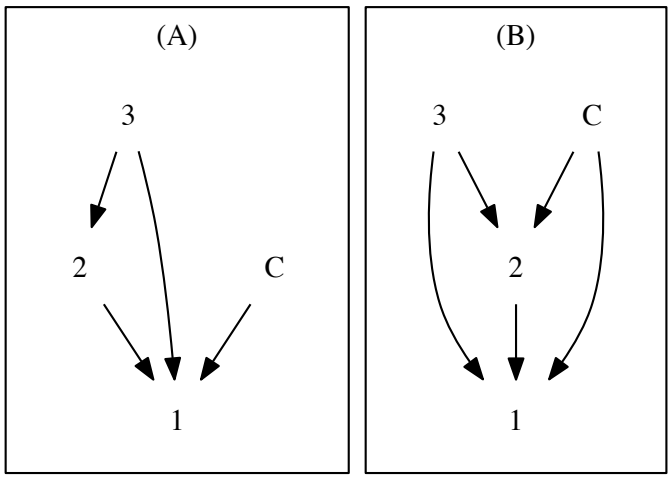

Figure 4: Examples of graph patterns based on control and 3 treatments. (A) a non-contractible graph; (B) a contractible graph, which represent genes unaffected by treatment 3 and down-regulated by treatments 2 and 1.

Predicting True Patterns of Gene Responses

A quick analysis of the non-contractible graph in Figure 5a shows that B being non-distinguishable from both A and C and the fact that $A > C$ are probably due to the lack of samples; as was concluded in a previous work [20]. Further, we see that response

graphs (b), (c), (d), (e) and (f) are all possible contractible graphs that contain (a) as subgraph. Thus, one might conclude that the true response can be one of these.

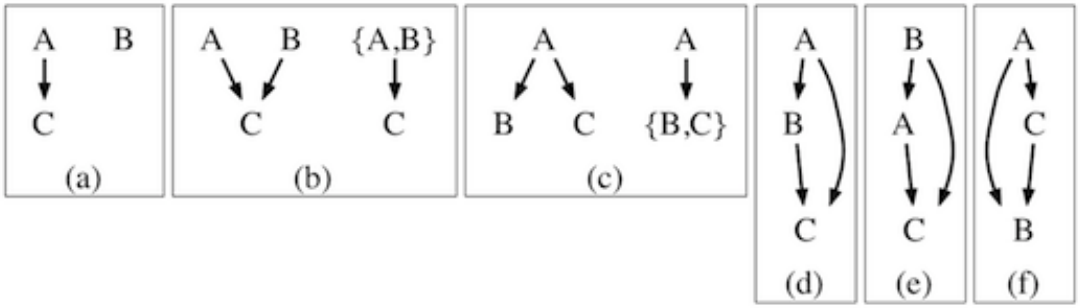

Figure 5: (a) *Non-contractible* pattern of a gene response to 3 treatments A, B, and C. (b-f) five *contractible* patterns that can come from pattern (a).

Our method for predicting true patterns of gene response works as follow. Given a non-contractible graph G, we generate "synthetic" data for non-adjacent vertices (corresponding to $A \sim B$ outcomes) until a contractible super-graph of G emerges. Synthetic data generation is a probabilistic process, so this step is done multiple times. Consequently, several super-graphs of G can be realized. The most dominant (i.e. most frequent) super-graphs are declared most likely candidates of true responses of the gene. One notice is, the answer might be a matter of resolution. For example, an additional 10 replicates, pattern (c) can be observed, and for an additional 100 replicates, pattern (d) can be observed. This would mean that for practical purposes pattern (c) is the gene's true response, and different between (c) and (d) might be practically insignificant.

To generate synthetic data, we rely on recent literatures that shed light on possible underlying distributions of gene expression. [77] showed that certain mouse genes were log normally distributed. [78] studied RNA-seq data and found that the expression levels of a majority of genes in metazoan cells followed a normal distribution or a combination of normal distributions.

Based on these works, we used normal distributions as the underlying distributions of gene expressions. Additionally, we used a goodness-of-fit test (2-sample

Kolmogorov-Smirnov test) to maintain that synthetic and experimental data likely come from the same distributions. We will show that this simple model can produce accurate predictions. This probabilistic strategy of inferring true responses of genes to multiple treatments is described in Algorithm 1.

Algorithm 1: generate-pattern(gene g, int s)

1: **for** $i = 1$ to T, the number of treatment groups of g **do**
2: Let $X_1, \cdots, X_n$ be experimental replicates.
3: Compute sample mean and standard deviation, $\bar{\mu}$ and $\bar{\sigma}$, from $X_1, \cdots, X_n$.
4: Draw $Y_1, \cdots, Y_s$ from the normal distribution $(\bar{\mu}, \bar{\sigma})$.
5: **if** $\{X_1, \cdots, X_n, Y_1, \cdots, Y_s\}$ and $\{X_1, \cdots, X_n\}$ are similarly distributed **then**
6: $R_t = \{X_1, \cdots, X_n, Y_1, \cdots, Y_s\}$
7: **else**
8: $R_t = \{X_1, \cdots, X_n\}$
9: Compute the directed graph G as the result of pairwise comparisons of $R_1, \cdots, R_T$.
10: **return** G

If a response graph is already contractible, no inference is made. For a non-contractible response pattern, let P be the multi-set of all contractible graphs generated by Algorithm 2 as predictions of true responses of a gene g. These graphs contain the original non-contractible response graph (without synthetic replicates) of gene g as subgraph.

Algorithm 2: predict-pattern(gene g, int s)

1: **if** G, the response graph of g, is contractible **then**
2: **return** $\{G\}$
3: $P = \{\}$ {Note: P is a multi-set}
4: **for** $i = 1$ to M **do**
5: $G' =$ generate-pattern(g, s)
6: Add G' to the multi-set P
7: **return** P

Let f_i be the frequency of each $P_i \in P$, and $p_i = \frac{f_i}{|P|}$ be the probability of observing P_i. The entropy of P, $H(P)$ given as $\sum_{P_i \in P} -p_i \log_2 p_i$. $H(P)$ varies between 0 and $\log_2 |P|$. When $H(P) = 0$, P is most certain, consisting of one unique pattern.

When $H(P) = \log_2 |P|$, P are uniformly random. Higher entropy implies less certainty in prediction, and vice versa.

Synthetic replicates are gradually added to real replicates until a dominant contractible pattern has emerged or the number of replicates exceeds a given threshold. This event is quantified as follows.

Lemma 2.2.1. If the entropy of a prediction is less than 1, a dominant pattern has emerged.

Proof. Suppose that $H(P) = \sum_{i=1}^{n} -p_i \log_2 p_i < 1$. We claim that for some j, $\max_i p_i = p_j > 0.5$. If not, then $\forall i, p_i \leq 0.5$, or $-\log_2 p_i \geq 1$. Therefore, $\sum p_i(-\log_2 p_i) \geq \sum p_i = 1$, which is a contradiction. □

2.2.3 Results

When non-contractible patterns are augmented with synthetic replicates to yield contractible patterns, we must ensure that these "predicted" contractible patterns are accurate. It is important to note that predicted patterns are likely super-graphs of originally observed patterns. Thus, our predictions only reveal more refined interactions and do not alter originally observed interactions. First, we will show that our model of generating synthetic replicates can be quite accurate for the purpose of pairwise comparison, under 4 different underlying distributions of gene expression. Further, using a rat microarray data set, we will show that annotating genes with predicted patterns yields more functional information than annotating genes with originally observed patterns.

Accuracy of Predicting A versus B

We tested our method against 4 different models of gene expression distributions: Gaussian, an equal mixture of two Gaussians (to simulate a bimodal distribution), Weibull, and lognormal. These models have been used or recognized as possible distributions of gene expressions [78, 77, 79, 80]. At the core of our method, we make inference about the true outcome of how a gene responds to A versus B, given the

observed outcome $A \sim B$ based on experimental (real) replicates. We wish to assess the accuracy of predicting the outcome of A versus B using synthetic replicates as described in Algorithm 5. The true outcome can be either (i) $A < B$, (ii) $A > B$, or (iii) $A \sim B$.

First, we would like to demonstrate the feasibility of our method making correct predictions under the 4 aforementioned distributions. To do this, we designed distributions A's and B's at varying distances; we used the Hellinger metric defined as $\mathbf{H}(A,B)^2 = 1 - \int_{-\infty}^{\infty} \sqrt{F_A(x)F_B(x)}dx$ and $0 \leq \mathbf{H}(A,B) \leq 1$. For each pair of distributions A and B, define:

- E, the minimum number of samples required to determine $A < B$ with probability of at least 0.95;
- P_n, the total number of real and synthetic replicates needed to predict $A < B$, given n real samples; we estimated P_n for $n = 5, 10$, and 20.

The result is shown in the top figure of Figure 6. Although the results were slightly different for different distributions, generally, as A and B became closer, E and P_n

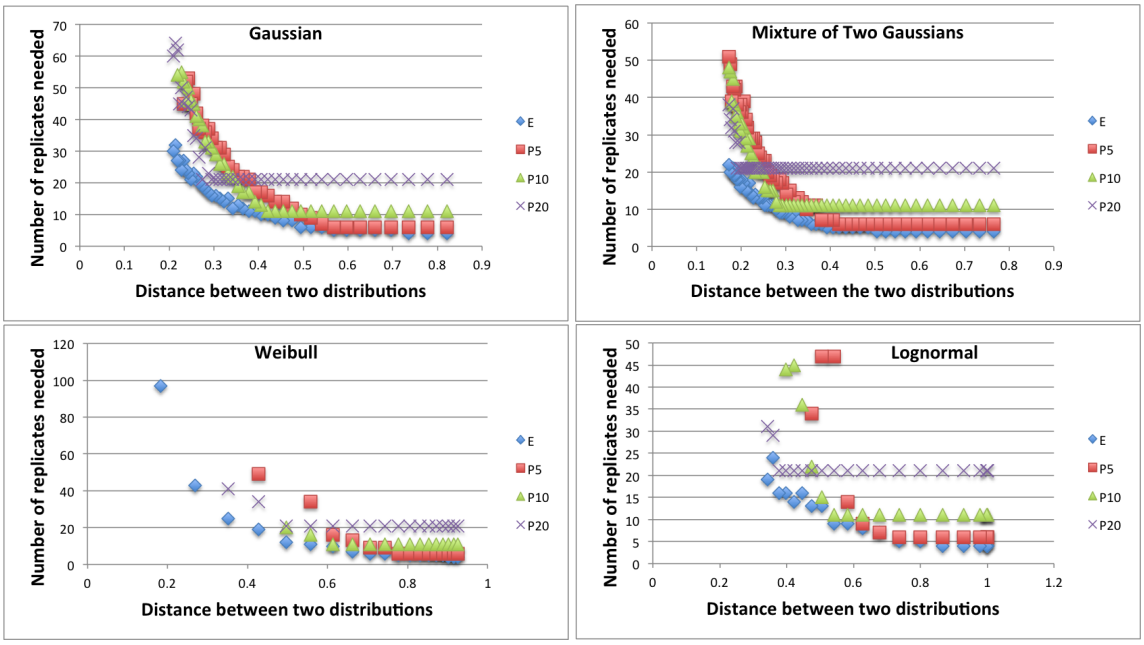

Figure 6: **E** vs. $\mathbf{P}_n$, $n = 5, 10, 20$ for 4 different distributions.

increase, meaning it takes more samples to determine and predict the correct outcome. Further, $E \leq P_n$ for all n's. The crucial point to observe here is that it was possible make the correct prediction of $A < B$ given 20, 10, or even 5 real samples. Even at very close distance of around 0.15 and $E > n$ (i.e. we had fewer real samples than necessary to determine with probability 0.95 that $A < B$), synthetic replicates helped make correct predictions. Although the smaller n was, the more synthetic replicates were needed.

Next, we would like to test for the ability of the method not making the mistake of predicting $A < B$ (or $A > B$), when the two distributions were in fact identical. We saw that in this case the method did not do as well as the other case. To put this into perspective, given $A \sim B$, we must measure both the probabilities of making the correct outcomes of $A < B$ and $A > B$ (*sensitivity*), and of not making the wrong prediction when A and B are identical (*specificity*). Figure 7 shows the trade-off between sensitivity and specificity, averaged across all 4 models of gene-expression distributions at distance 0.5 with $n = 5$ (results are similar at other distances and $n = 10$ or 20). The main point is that as the number of synthetic replicates increased, the probability of correctly predicting $A < B$ (or $A > B$) increased; but the probability of correct predicting $A = B$ decreased. Overall, however, the accuracy remained high, generally above 0.85.

In summary, given 4 quite different underlying distributions of gene expression, the method performed relatively highly accurate of correctly predicting the true outcome of A versus B, given $A \sim B$.

Analysis of Gene Expression Data from Rats

To test the performance of this method on an unknown gene expression distribution (to the best of our knowledge), we used a microarray dataset, in which liver samples of Spague-Dawley rats were treated with either control diet or one of 3 chemopreventive compounds with well understood pharmacological activities: 5,6-benzoflavone (BNF), 3H-1,2-dithiole-3-thione (D3T) and 4-methyl-5-pyrazinyl-3H-1,2-dithiole-3-thione (OLT). This dataset consists of 5 real

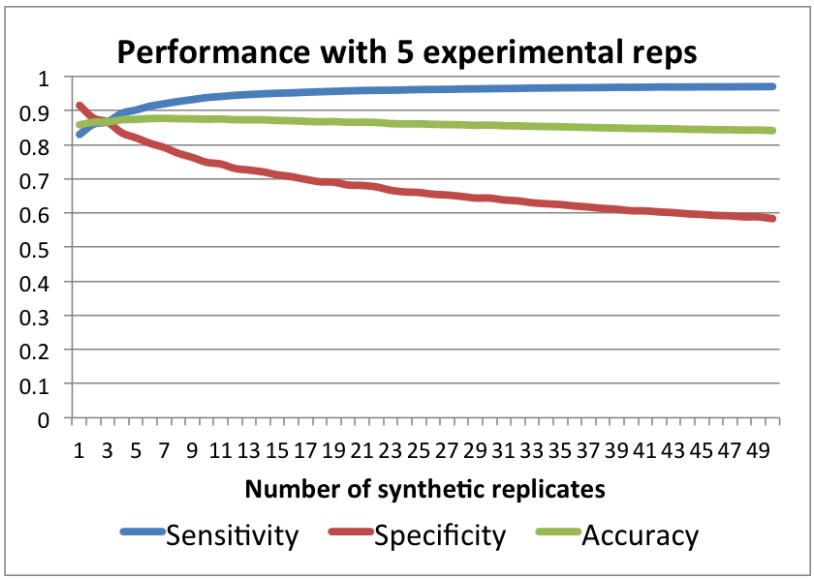

Figure 7: Overall prediction performance for distributions at distance ≈ 0.5.

replicates, based on which 1737 significant genes were selected. From these, 1252 (72%) and 485 (28%) genes were found to have contractible and non-contractible patterns, respectively. In total, 45 contractible and 69 non-contractible patterns were observed. Full description of this data set and the compounds can be found in [20].

BNF is a bifunctional inducer and a ligand to AhR. Upon ligand binding, the AhR is activated and translocates to the nucleus where it forms a heterodimer complex. This complex binds to xenobiotic or dioxin response elements found upstream of several known target genes and acts to enhance the rates of gene transcription in the first pathway. Also, BNF can be metabolized to electrophilic intermediates capable of activating the second pathway. Dithiolethiones such as D3T and OLT are monofunctional inducers, which affect the dissociation of a Cap-N-Collar type leucine zipper transcription factor, Nrf2, from its cytosolic protein partner Keap1. Upon dissociation, Nrf2 translocates to the nucleus where it complexes with small Maf transcription factors affecting gene transcription of the second pathway.

Because the distribution of gene expression is unknown, the only way, and perhaps

ultimately meaningful way of assessing the performance of the method is by functional validation of the observed and predicted patterns, respectively.

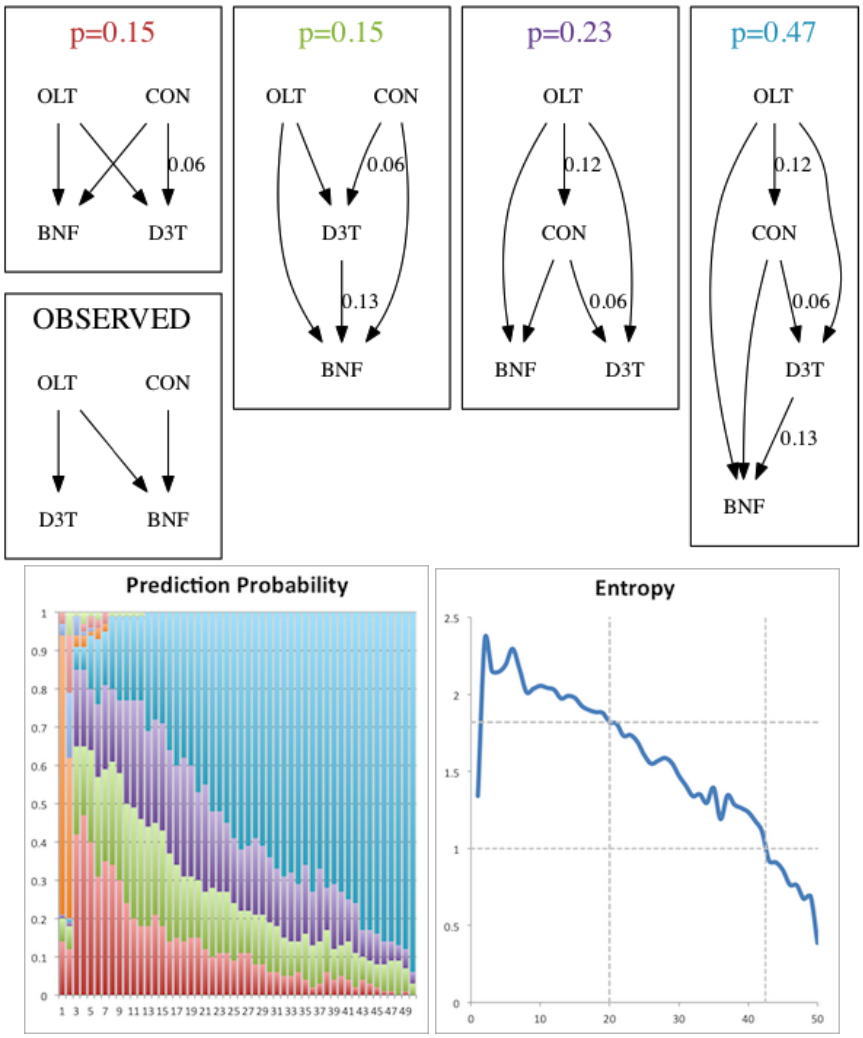

Figure 8: Pattern observed with 5 experimental replicates; 4 patterns predicted with 20 extra synthetic replicates; probability and entropy of predictions as functions of extra synthetic replicates.

First, we would like to demonstrate the utility of this method in analyzing non-contractible patterns. Take for example a non-contractible pattern observed from this data set, and shown in Figure 8. This graph has 3 edges: $CON \rightarrow BNF$, $OLT \rightarrow BNF$, and $OLT \rightarrow D3T$. As more synthetic replicates were added to augment the original pattern, entropy was reduced and predictions became more certain. This is generally expected for any gene. When the entropy gets below 1 (at 42 synthetic replicates), a

dominant pattern is guaranteed to exist. In this case, the pattern, however, began to be dominant at 20 synthetic replicates, with probability 0.47. Since adding synthetic replicates decreases specificity, how do we choose the optimal number of synthetic replicates? Observe that after 20 synthetic replicates, the 4 most probable patterns were already identified and remained unchanged structurally. This suggests 20 synthetic replicates might be a good choice.

The 4 most probable patterns predicted with 20 synthetic replicates are all contractible and contain the observed pattern as subgraph. This means adding synthetic replicates removes ambiguity in non-contractible patterns, but does not alter the original response observed with real data. To distinguish original response and predict response, predicted edges can be annotated with error probabilities. The error probability of predicting $A > B$ is defined to be $p + q$, where p and q are the probabilities of predicting $A > B$ given the true outcomes are $A \sim B$ and $A < B$, respectively. P and q were estimated by averaging over same error probabilities under 4 different models: Gaussian, Mixture, Weibull, and Lognormal.

Additionally, the event of a graph becoming contractible deserves special attention. For example, the edge $CON \rightarrow D3T$ has lowest error probability and appears in all 4 predicted patterns. Its addition makes the observed pattern contractible, suggesting $CON \rightarrow D3T$ is most likely a true response.

The main point here is that this method of prediction does not alter originally observed interactions; all 4 predicted patterns are supergraphs of the observed pattern. The analysis shown here suggests that we can conservatively predicts more refined interactions that were not originally observed, without altering originally observed information based on real data.

Functional Validation of the Predicted Patterns

We would like to assess the biological relevance or meaningfulness of patterns observed with real replicates and patterns predicted with additional synthetic replicates.

We could apply clustering to observed and predicted patterns, separately, and functionally validate each cluster. This would, however, add another level of bias due to the choice of clustering method. Instead, we grouped all genes having same observed patterns into same sets, producing a collection $\mathbf{O}$ of gene sets. Similarly, we grouped all genes having same predicted patterns into the same sets, producing of collection $\mathbf{P}$ of gene sets. Then, we functionally validated $\mathbf{O}$ versus $\mathbf{P}$.

We used a well-known functional tool called DAVID [81], an integrated biological database aimed at systematically extracting biological meaning from large gene or protein lists, to extract highly-enriched clusters from gene lists, each of which consisted of all genes with same pattern. To quantify the degree of biological relevance, we counted the number of functionally enriched clusters and calculated the average enrichment score per enriched cluster. We expect biologically meaningful assignments of patterns to genes will yield many functionally enriched clusters and highly scored cluster.

Figure 9 compares functional enrichment of clusters produced by patterns observed with experimental replicates only versus patterns observed with experimental plus additional synthetic replicates. We can see that, with synthetic replicates, not only were there more enriched clusters, each cluster were also more enriched. Interestingly, few predicted patterns did not appear originally, i.e. when observed with real data. Few genes bearing these "new" patterns were actually found by DAVID to belong to enriched clusters; these are represented by the red bars in the figure. This gives raise to the following hypothesis that 5 experimental replicates were too few for many genes to show subtle interactions to similar chemical compounds. When augmented with synthetic replicates, these interactions emerged.

It is important to observe that adding more synthetic replicates does not necessarily yield more biological meanings. This affirms what we established in a previous section that although adding more synthetic replicates reduces false negatives, it also introduces more false positives. There is a point at which adding more synthetic

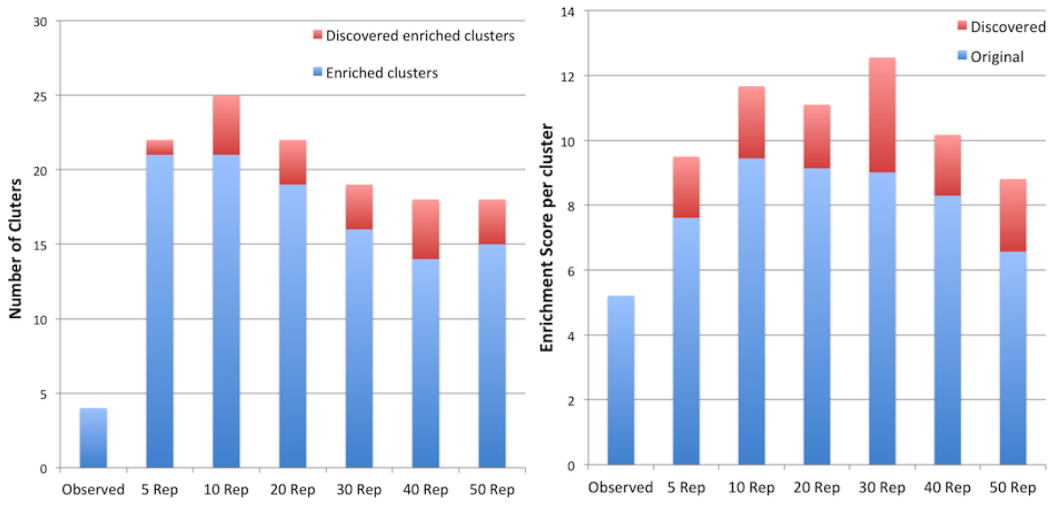

Figure 9: Functional analysis of gene clusters obtained from patterns defined by experimental replicates versus patterns defined by experimental and additional synthetic replicates. Left: number of enriched clusters. Right: averaged enrichment score per enriched cluster. Blue: clusters originally observed with experimental replicates. Red: clusters discovered when synthetic replicates were introduced.

replicates only gets worse results. The figure shows that for this particular data set, we would stipulate that adding about 10 synthetic replicates to the 5 real replicates gives possibly optimal predictions.

2.2.4 Summary

We demonstrated the relatively high accuracy of the method for 4 different models of gene expression distribution, resulting in more enriched clusters and also better functionally-enriched clusters. We showed the method could uncover more subtle interactions that might be missing with a small sample size, which is something traditional clustering methods are unable to do. Such uncovered interactions only add to and do not alter the originally observed interactions.

Although there is not yet a formula to determine precisely the optimal number of synthetic replicates for a specific data set, the proposed method can be used to analyze real data, as demonstrated in Section 2.2.3. One has to experiment with different numbers of synthetic replicates and validate predicted patterns using available tools, revealing the trade-offs between using too few or too many synthetic replicates.

2.3 mDAG: a Web Tool for Analyzing and Visualizing Gene Expression Data

2.3.1 Introduction

We previously introduced a method based on *post hoc* pairwise comparisons to analyze gene expression responses. This method utilized directed graphs to represent gene response to all treatment pairs. It has been found useful in identifying structure-activity relationships among drugs and differentiating genes sharing similar functional pathways. Directed graphs are descriptive, visually expressive and can benefit subsequent functional analysis. This work led to the creation of a software package that allows visualization of gene responses and a visual distinction between possibly accurate response patterns (for genes whose replicates are sufficient) and possibly unreliable response patterns (for genes whose replicate maybe insufficient). To the best our knowledge, there has been no similar tool that assists researchers visualize how genes respond to multiple treatments.

Here we introduce mDAG, a web-based software package based on this method for the analysis, visualization, and interpretation of patterns of responses in gene expression data involving multiple treatments. mDAG allows users to analyze and visualize gene response patterns represented as directed graphs. Genes with same directed-graph response patterns are grouped together and linked to external resources, such as DAVID [81], GeneMANIA [82], and GCAT [83], for functional analyses. Genes with the same directed graph patterns hypothetically share similar biological function, which may be further analyzed using such external tools. To facilitate subsequent functional analysis, several well-known tools have been incorporated into mDAG to help users explore hypotheses about gene function and regulation. This tool is useful for any studies that analyze comparatively response patterns in gene expression data with multiple treatments (chemicals, cell types, etc.). More details of this tool can be found in Appendix A.

2.3.2 Method

Response Graphs

Give gene expression data of k treatment groups, each having multiple replicates, significantly differentially expressed genes are selected. Then, each gene is assigned a directed-graph pattern or *response graph*, in which each vertex represent a treatment group and edges represent how the gene responds to treatment pairs. Edges are defined by $\frac{k(k-1)}{2}$ statistical tests. Given a pair of vertices A and B, a Wilcoxon rank sum test considers replicate data of treatments A and B and determines how the gene responds comparatively to A and B. If the gene is expressed statistically significantly higher under A than B, then the edge $A \rightarrow B$ is established. Conversely, if the gene is expressed statistically significantly higher under B than A, then the edge $B \rightarrow A$ is established. When the test cannot distinguish how the gene responds comparatively to A and B, there is no edge between A and B in the graph.

Assessing Confidence of Observed Patterns

Response graphs capture exactly primary and secondary patterns of gene responses to all treatment pairs. To help users assess confidence in observed patterns, two types of directed graph patterns are identified: *contractible* and *non-contractible*. A graph is said to be contractible if and only if non-adjacent vertices are equivalent, in the sense that their incoming vertices are identical and their outgoing vertices are also identical. We can look at back to Figure 4 in section 2.2.2, which shows examples of contractible and non-contractible graphs. As shown in [20] and validated in [21], contractible patterns are more likely to be accurate, whereas non-contractible patterns are inaccurate due to having too few samples. Contractible patterns are also more easily interpreted. As an example, genes bearing the pattern shown in Figure 4B are unaffected by treatment 3 and down-regulated by treatments 1 and 2; these genes also respond more strongly to treatment 2 compared to treatment 1. Our work [59] further showed the utility of the contractibility of graphs to make inference about patterns of gene response.

2.3.3 Results

Implementation

mDAG is written in Python, developed based on a web framework known as web2py, and support most popular browsers. For database storage, it can be configured to use SQLite, MySQL, Postgres, or a variety of database management systems. mDAG needs a minimum requirement that includes a pre-installed Python 2.6+ and Graphviz, an open source graph visualization software. It has an own scheduler program to manage requests properly as well as exploit computer resources effectively to perform requests. It can be configured as a stand-alone application for personal usage or installed on a server for group usage.

To illustrate the utility of the software, we simulated three sample datasets from a gene expression dataset of rats' liver tissues ([20]). The original dataset includes 12906 genes with control and three treatments and 5 replicates per treatment (including control). The simulated datasets have 8, 10, and 12 treatments (including control), respectively. To be convenient for manipulating with these datasets, we denote control by C and treatments by numbers (1 to 7 for the first, 1 to 9 for the second, and 1 to 11 for the third dataset).

Recognizing Contractible Patterns

To help users interpret patterns of gene responses, the software distinguishes two types of directed graph patterns: contractible and non-contractible patterns. The reason for differentiating these kinds of patterns is that as shown in [20], contractible patterns are more likely to be true patterns and can be interpreted unambiguously.

For example, consider Figure 10, which shows four selected patterns from an analysis with 8, 71, 7, and 23 genes, respectively. We see that pattern A is non-contractible and patterns B, C, and D are contractible. Non-contractible pattern A is ambiguous in that, for example, genes having this pattern respond indistinguishably to both treatment pairs 3 & 5, and 5 & 2, and yet they respond more to treatment 3 than to treatment 2. On the other hand, contractible pattern B captures precisely how genes

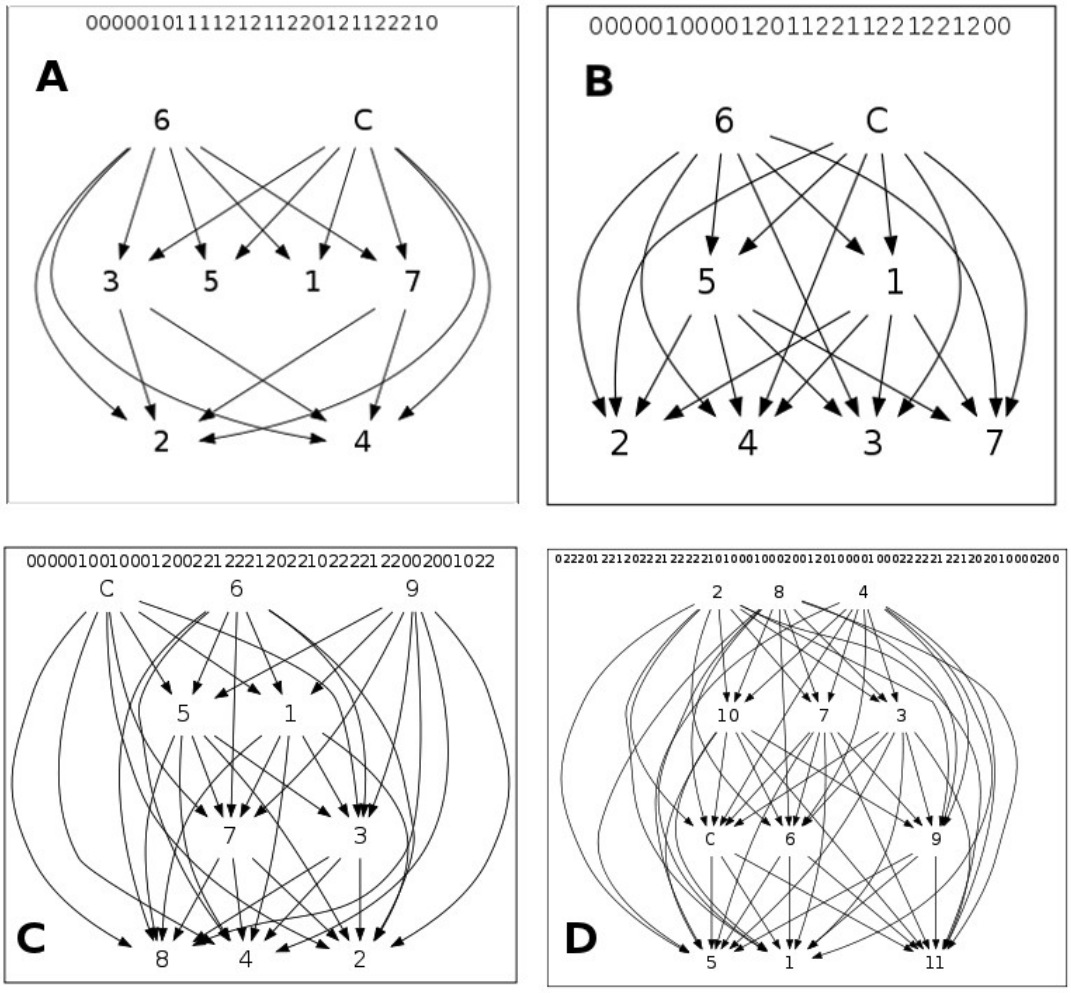

Figure 10: Examples of response graphs: [A] a non-contractible pattern with control and 7 treatments; [B, C, D] contractible patterns with control and 7, 9, and 11 treatments, respectively.

respond to all treatments. The response can be unambiguously linearized as follows: $\{6 \sim C\} > \{5 \sim 1\} > \{2 \sim 4 \sim 3 \sim 7\}$. This means that the 71 genes in pattern B are not affected by treatment 6, and are down-regulated by treatments $5, 1, 2, 4, 3, 7$, but more so by treatments $2, 4, 3, 7$. Further, these genes respond indistinguishably to to treatments 5 & 1, and to treatments 2, 4, 3, & 7. We can similarly and unambiguously interpret contractible patterns C and D.

Filtering to Identify Secondary Responses

Users are able to analyze further gene clusters by filtering patterns based on how genes respond specifically to individual treatments: affected or not affected, up-regulated or down-regulated by a treatment. Users can also filter patterns for a group of genes indicated by Probeset IDs, Gene Symbols, UniGene IDs, or Accession Numbers. Figure 11 shows how to filter patterns with gene response criteria (the left side) and how to filter patterns for a group of genes with proper identifiers (the right side).

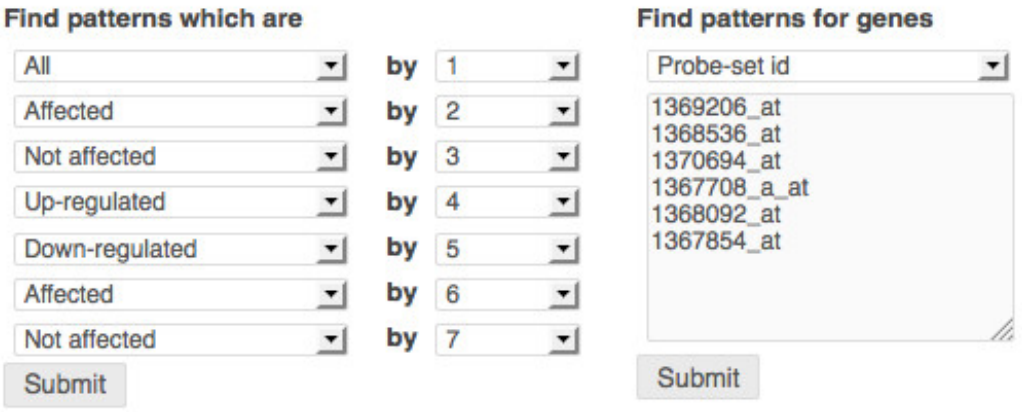

Figure 11: Filtering to identify secondary responses: Left: filtering based on treatment effect: up-regulated, down-regulated, affected, not affected, or any; Right: filtering based on Probset IDs.

Additionally, to facilitate gene cluster analyses, patterns of gene responses are pre-grouped into meta-clusters. A meta-cluster at k-level can be defined as a group of genes that have k groups of equivalent treatments (that can be one or more treatments). Each meta-cluster can consist of several directed graph patterns. Currently, our software shows first level meta-clusters, which includes all genes that have one group of equivalent treatments. For example, patterns in Figures 10A and 10B (see section 2.3.3) share the same meta-cluster "$< \{6 \sim C\}$" (up-regulated by group "$\{6 \sim C\}$"). We can also account for up-down responses for meta-clusters, in this case, pattern in Figure 10B has the meta-cluster "$> \{2 \sim 4 \sim 3 \sim 7\}$" (down-regulated by group "$\{2 \sim 4 \sim 3 \sim 7\}$").

Functional Analyses of Patterns

Genes sharing same directed-graph patterns hypothetically share similar biological function. To facilitate functional analyses, the software incorporates other tools that are well known for this task. Each cluster of genes with same patterns can be analyzed further via three external well-established tools with quite different approaches DAVID [81], GeneMANIA [82], and GCAT [83]. Figure 12 (upper part) shows how to use these tools with our software.

Users are also able to link genes in patterns to NCBI resources by using their identifiers such as Probset IDs, Gene Symbols, UniGene IDs, or Accession Numbers. Figure 12 (lower part) shows how to link to these resources from our software. Tools with well-defined APIs will be continually incorporated into the software. To facilitate subsequence analyses, the software also shows P-value and fold change for each gene. In the future, we will make users be able to set fold change threshold in the software by themselves. In this case, the edge $A \rightarrow B$ ($B \rightarrow A$) will be established if and only if the gene is expressed statistically significantly higher under A (B) than under B (A), and fold change is higher than the given threshold. Otherwise, there is no edge between A and B.

2.3.4 Summary

We introduced a novel web-based software to facilitate comparative gene-expression studies involving multiple treatments. This tool uses directed graphs to represent patterns of gene response to treatments in such a way that these response patterns are descriptive and visually informative. We showed how to use the software to interpret patterns of gene response, filter these patterns to identify secondary responses, and how to perform functional analyses for genes with same patterns. mDAG has features to help users analyze, organize and manage data conveniently and can be easily configured for personal or group usage.

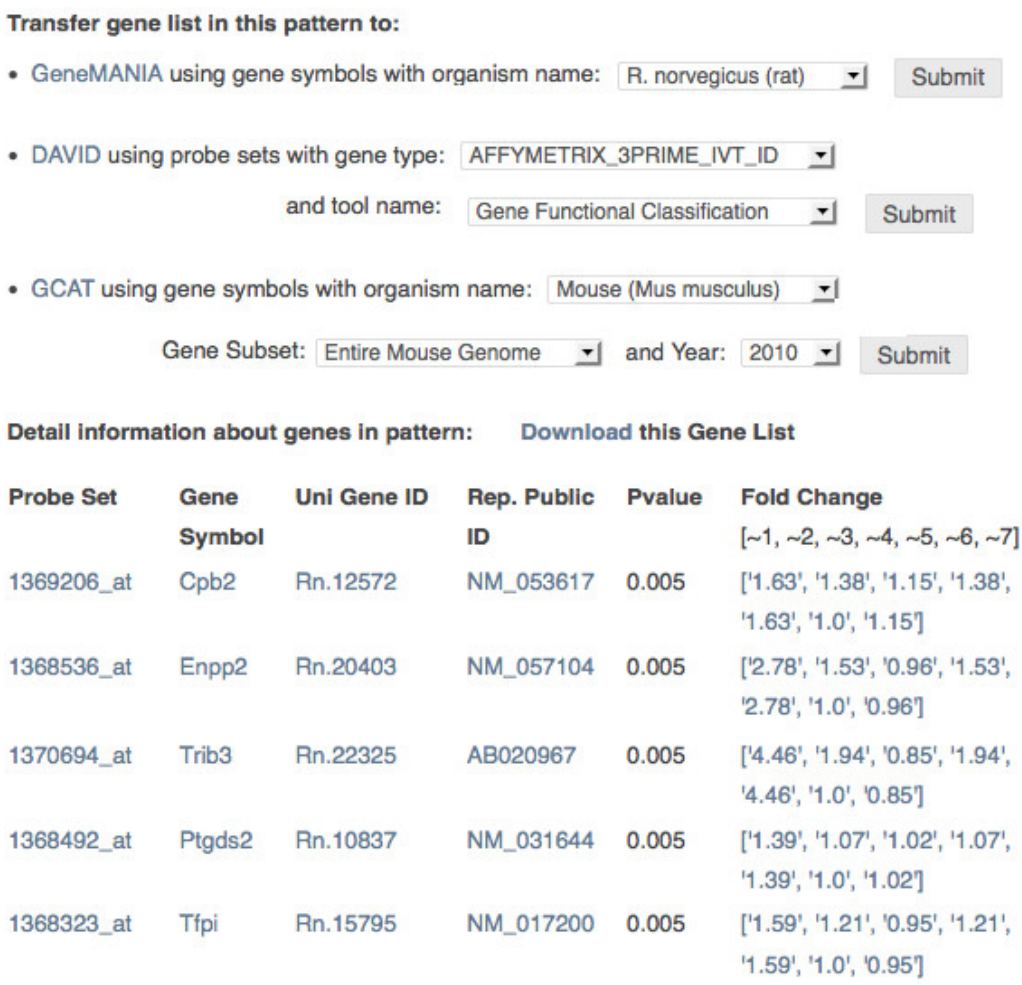

Figure 12: Functional analyses of genes with same patterns: Upper: analyzing biological functions of patterns by transferring their gene lists to proper external tools with several options; Lower: checking information of genes in patterns by linking genes to NCBI resources using their identifiers.

2.4 Exploiting Orderable sets to Predict True Patterns of Gene Response

2.4.1 Introduction

In this section, we introduce a novel method which uses orderable sets to predict true patterns of gene response to treatments in experiments with small sample size. We will show that orderable sets can be exploited to represent patterns of gene response to treatments and more importantly, the dependencies of pairwise comparison outcomes could be exploited effectively to predict true patterns of gene response. Most of the content in this section has been published in ([61, 62]).

In this method, the patterns are determined by relative orders of gene response to treatments, which are obtained using pairwise comparisons using the procedure in section 2.1.1. First, we will show that true patterns of gene response must be *orderable* sets. Then we modify the conventional pairwise comparison tests by increasing the significance level α in a proper manner to deduce orderable patterns, which are shown to be most likely true orders of gene response. We will show that gene lists obtained with this method yield more and better functional enrichment than those obtained with the conventional approach. These results were confirmed using DAVID [2], a well-established tool for extracting biological meaning for gene lists from most major public resources.

Additionally, we will show that gene clusters labelled by patterns produced by our method are much more enriched than those of hierarchical clustering. This method discovered clusters of genes with important biological and pharmacological activities that hierarchical clustering fails to detect, which was confirmed by literature. Moreover, motivated by the fact that a gene can be involved in multiple biological functions, we resampled experimental replicates and predicted multiple response patterns for each gene. Our experiment results using DAVID showed that compared to the conventional method of fixing α and using only experimental samples, this method increased biological enrichment of patterns significantly.

Our approach is promising to design cost-effective experiments with small sample sizes. Using our method, patterns of highly-variantly expressed genes can be predicted by varying α intelligently based on the exploitation of orderable patterns. Furthermore, clusters are labeled meaningfully with patterns that describe precisely how genes in such clusters respond to treatments.

---

[2] http://david.abcc.ncifcrf.gov/

2.4.2 Method

Representing Gene Response Patterns with Orderable Sets

Let P_g be the set of all $\frac{k(k-1)}{2}$ outcomes obtained from pairwise comparisons using the procedure described in section 2.1.1. When the pattern P_g of a gene is observed at a confidence level α, it may not be the true pattern of gene response, especially if sample size is small. To distinguish the true patterns from all patterns, we can use the following observation:

Observation 1. Let Δ be the pattern defined by the following outcomes: $A \prec C, A \sim B, B \sim C$. Then Δ is likely not a true pattern of gene response to the 3 treatments because sample size is insufficient. Moreover, if a pattern contains Δ, then it is likely not a true pattern.

As an illustration, Figure 13A shows the Δ and Figure 13B shows a pattern that cannot be true as it contains Δ. To explore further properties of true patterns, we define:

Definition 1. A pattern P based on n elements $t_1, \cdots, t_n$ is **orderable** if $\forall i, j$ such that $t_i \sim t_j$, $G(t_i) = G(t_j)$ and $L(t_i) = L(t_j)$ where $G(t_i) = \{t_k | t_i \prec t_k\}$ is the set of elements "larger" than t_i, and similarly $L(t_i) = \{t_k | t_k \prec t_i\}$ is the set of elements "smaller" than t_i.

and use the following observation:

Observation 2. A pattern is orderable if and only if it does not contain Δ.

From observations 1 and 2, we can conclude that true patterns are likely orderable. Although this does not imply that orderable patterns must be true response of genes, if we assume that already observed outcomes of types $A \prec B$ are correct (with high probabilities), then additional samples do not change these and therefore true pattern must be an *extension* of the observed pattern, which is defined as:

Definition 2. Q is a **orderable extension** of P if (1) Q is orderable, and (2) $\forall i, j$ if P contains $t_i \prec t_j$, then Q also contains $t_i \prec t_j$.

For instance, the patterns shown in Figure 13C-D are among the orderable extensions of the pattern shown in Figure 13B. These observations turns out can be used to predict true patterns of gene response to treatments. The underlying reasoning of these observations and more details of the method can be found in [62].

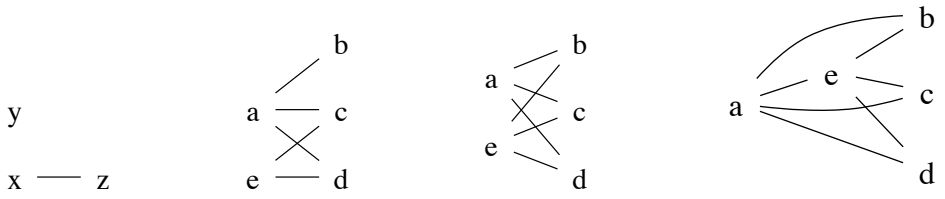

Figure 13: (A) $\Delta = \{x \prec z, x \sim y, y \sim z\}$. (B) A pattern that is not true as it contains Δ (on elements a, e, and b). (C) an orderable pattern. (D) another orderable pattern. Both patterns C and D are *orderable extensions* of the pattern B.

Vary α to Predict True Patterns

The conventional procedure to determine patterns of gene response using pairwise comparisons can be summarized in Algorithm 3, which takes as input the samples of a gene responding to all treatments and a specific confidence level α.

Algorithm 3: Pattern(S, α)

1: Compute p-values computed in comparison tests for all treatment pairs based on samples S
2: Let P be the set of all outcomes computed based on α
3: **return** P

The confidence level α in comparison tests plays an important role in determining the outcomes $A \prec B$ or $B \prec A$ or $A \sim B$ and ultimately whether or not the collective pattern is orderable. As an example, suppose that given 4 hypothetical treatments, the p-values of outcomes for 6 comparison tests are: 0.45 (for the test if $\mu_A < \mu_B$); 0.03 (for the test if $\mu_A < \mu_C$); 0.055 (for the test if $\mu_A < \mu_D$); 0.03 (for the test if $\mu_B < \mu_C$); 0.03 for the test if $\mu_B < \mu_D$; 0.45 (for the test if $\mu_C < \mu_D$). In independent statistical tests, α is

fixed at a conventional small value such as 0.05 to reduce false positives [76]. If we set α at 0.05, then we will get the pattern $\{A \sim B, A \prec C, A \sim D, B \prec C, B \prec D, C \sim D\}$, which is the left pattern in Figure 14. But this pattern is not orderable and cannot be true. If, however, we set α at a slightly higher value at 0.06, we can accept $\mu_A < \mu_D$ (with p-value = 0.055), which will get an orderable pattern in Figure 14.

Since we have established that true patterns must be orderable, it makes sense to vary α conservatively in these dependent tests to achieve orderable patterns, which are more likely to be true patterns of gene response to treatments. Based on this idea, we can devise a procedure to determine an orderable pattern for a given gene, by finding a minimal α that yields a nontrivial orderable pattern. Algorithm 4 takes as input samples (replicates) of all treatment groups and finds an orderable pattern by increasing α (so long as it does not exceed a large value $\alpha_{\max}$, e.g. 0.5) until a non-empty orderable patterns is obtained.

Figure 14: Based on the same hypothetical expression data, (Left) observed pattern is not orderable with $\alpha = 0.05$. (Right) observed pattern is orderable with $\alpha = 0.06$.

Algorithm 4: OrderablePattern(S)

Let L be the list of p-values computed in comparison tests for all treatment pairs based on samples S.
Sort L in increasing order (duplicates removed).
Let m be the largest index of L such that $L[m] < \alpha_{\max}$
for $i = 0$ to m **do**
    $\alpha \leftarrow L[i]$
    P_α be the set of all outcomes computed based on α.
    return P_α if it is orderable & nontrivial
return $\emptyset$

Predict Multiple Patterns for Each Gene

A true pattern must be orderable, but an orderable might not be true. For example, either patterns in Figure 13 can be the true pattern depending on which α is used. When there are many patterns with approximately equal probabilities of being true, it makes more sense to predict multiple patterns for each gene. Further, because a gene might participate in multiple biological functions or molecular processes, assigning multiple patterns to a gene makes sense biologically.

Conventionally, methods for determination of patterns of gene response assign one most-probable pattern to one gene. This strategy however has several problems with respect to biological interpretation of patterns. One of the problems is, a gene can play a role in many biological processes, however the assignment of one pattern to one gene do not capture this situation. To deal with this, one can think about a strategy in which gene would be predicted probabilistically with several potentially patterns. On the other hand, we can perform a many-to-one assignment of patterns to genes. This strategy leads to nice biological interpretation, in which a gene might play multiple roles and participate in different functions.

To predict the most probable patterns for a given gene, we employ the bootstrap method [84, 85] to resample (with replacement) from the set of gene expression replicates to create a large number of datasets for the purpose of approximating the sampling distribution of gene expression. Algorithm 5 computes the set of most probable patterns (and their probabilities) of a gene g using M iterations of bootstrap.

To be conservative in predicting multiple patterns, Algorithm 10 employs a heuristic which stipulates that if a pattern, P, is orderable based on a conventionally low α such as 0.05, then P is likely the true pattern. In this case, the algorithm returns P and its probability of 1. In case where P is not orderable at $\alpha = 0.05$, we generate bootstrap samples for all treatments t_i's that might contribute to yielding orderable patterns. These are treatments involving in outcomes of the type $t_i \sim t_j$ in P. In each iteration k,

bootstrap replicates are used to determine a pattern P_k using either the fixed α (Algorithm 3 in section 2.1) or variable α method (Algorithm 5). After a large number (M) of probabilistic bootstrap experiments, we can determine the set of patterns with their corresponding probabilities.

Algorithm 5: MultiplePatterns(g)

Let $S = \{S_1, \cdots, S_t\}$ be the set of samples of gene g, where S_i is the set of samples of treatment t_i.
$P \leftarrow Pattern(S, 0.05)$ (Algorithm 3)
if P is orderable **then**
    return $(P, 1)$
Let Q be a multiset, initially empty.
for $k = 1$ to M **do**
    for each i such that $\exists j, t_i \sim t_j \in P$ **do**
        replace samples S_i with bootstrap replicates
    Determine pattern P_k based on experimental and bootstrap replicates, using either Algorithm 3 or Algorithm 4.
    $Q = Q \bigcup P_k$
return $\{(P, f_P) \mid P \in Q,\ f_P$ is the frequency of P in $Q\}$

2.4.3 Results

Design of Experiments and Evaluation

To validate our method, we used a gene expression data set came from a controlled study of samples of livers of Sprague-Dawley rats treated with either control diet or one of three chemopreventive compounds with well understood pharmacological activities, 5,6-benzoflavone (BNF), 3H-1,2-dithiole-3-thione (D3T) and 4-methyl-5-pyrazinyl-3H-1,2-dithiole-3-thione (OLT). This dataset had 5 samples in each of 4 treatment groups (including control group). It was available publicly with GEO accession number GSE8880 [20].

There totally 1737 significantly differentially expressed genes were selected using the Kruskal-Wallis procedure. Each gene is placed into a bin labelled by its pattern. When we assign multiple patterns (say $P_1, \cdots, P_k$) to a gene, we will place the gene to bins with labels P_1 or P_2, $\cdots$, or P_k. Thus, each bin contains genes with the same patterns.

Ultimately, the usefulness of a method lies in its ability to predict biological functions. To evaluate our method, we consider genes labelled with the same pattern as a gene cluster. We use DAVID to evaluate functional enrichment of each cluster. DAVID is a resource aimed at systematically extracting biological meaning from large gene lists. DAVID integrates biological information from most major public bioinformatics resources. We used the Gene Functional Classification tool of DAVID to extract highly-enriched clusters from each gene list. To quantify the degree of enrichment, we consider the number of functionally enriched clusters, number of enriched genes, and enrichment score that DAVID returns. All analyses were run with default parameters.

Varying α Results in More Orderable Patterns

With α fixed at 0.05, 1252 (72%) genes acquired 45 orderable patterns and 485 (28%) genes acquired 69 not orderable patterns. We observe that many orderable extensions can be obtained by modestly raising α beyond 0.05. Figure 15a shows that with $\alpha \leq 0.075$, patterns of 84% of genes were orderable; and with $\alpha \leq 0.15$, 97% of genes had orderable patterns. With $\alpha \leq 0.5$, 100% of genes had orderable patterns. There were 55 orderable patterns, which include all 45 orderable patterns observed with $\alpha = 0.05$, plus 10 new patterns.

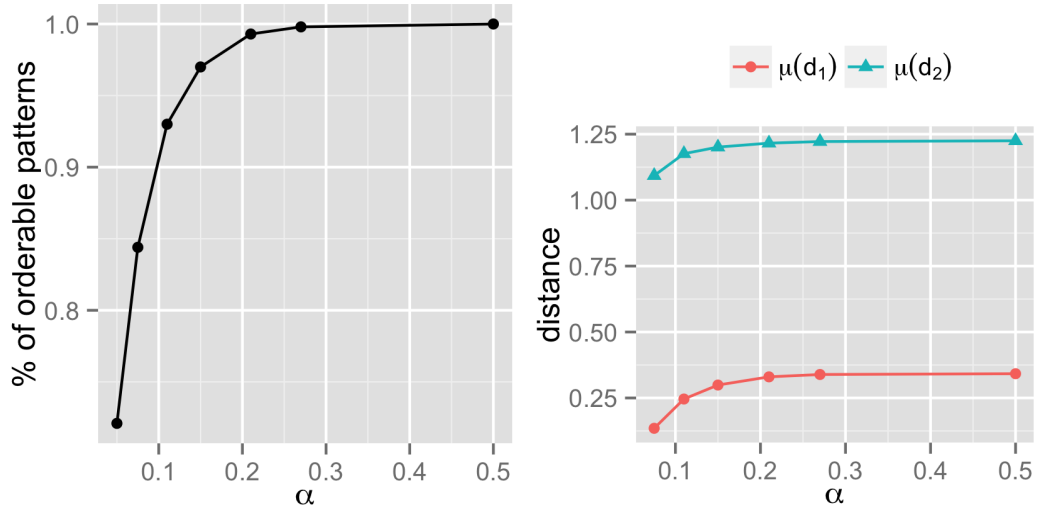

Figure 15: (a) fraction of orderable patterns at increasing values of α. (b) structural difference between patterns acquired at $\alpha = 0.05$ and at higher values.

When the pattern P of a gene at $\alpha = 0.05$ becomes another pattern Q at larger α, by construction, Q is a orderalbe extension of P. We will show quantitatively that the difference between P and Q is not very much. To compare the structural difference between patterns obtained at α fixed at 0.05 and their orderable extensions obtained by a higher α, we define the difference between two pattern P and Q as $d(P,Q) = \sum_{i=1}^{\binom{n}{2}} \delta(p_i, q_i)$, where $\delta(p_i, q_i)$ is 0 if the i^{th} outcome of P and Q is the same, and 1 if it is different. For example, if P and Q are the patterns shown in Fig 13, then $d(P,Q) = 1$ as they differ only in the comparison of a and e.

Figure 15b shows the average structural difference between a pattern observed with α fixed at 0.05 and its orderable extension observed varying α. This difference is denoted in the figure as $\mu(d_1)$. As this number is well below 0.4 for all values of α, we see that on average a pattern is only very slightly different from its orderable extension. This number, however, does not tell the whole story, because 72% of genes acquired orderable patterns at $\alpha = 0.05$ (meaning they are trivially their own orderable extensions). Thus, we proceed to analyze patterns that were not orderable at $\alpha = 0.05$. The structural difference between these patterns and their orderable extensions is denoted in the figure as $\mu(d_2)$. We see that this number is below 1.2 at all values of α. This means that on average adding roughly 1 outcome (of type $A \prec B$) to these patterns would make them orderable.

Varying α Results in Better Enrichment

We have shown that varying α, i.e. allowing different α's for different genes, resulted in more orderable patterns. Since orderable patterns are more likely to be true patterns of gene response to treatments, it is expected that genes grouping in patterns that more likely true would result in more enrichment of the gene set. The enrichment of a gene list is described by DAVID's functional annotations in 3 aspects: (i) the number of enriched clusters (we refer to this in the figures as *E.cluster*), each of which is a subset of the gene list, (ii) the number of genes in each enriched clusters (*E.gene*), and (iii) the enrichment score of each cluster (*E.score*).

Figure 16 shows that gene lists obtained from by varying α for each gene resulted in better enrichment than those obtained by fixing α at 0.05 for all genes. Specifically, with α fixed at 0.05, DAVID found 62 enriched genes in 12 enriched clusters, with a total enrichment score approximately 20.83. Meanwhile, for gene lists obtained with varying α up to 0.15 for each gene, DAVID founded 119 enriched genes in 23 enriched clusters with a total enrichment score of 46.70. Thus, gene lists produced by varying α up to 0.15 for each gene are twice as enriched as fixing α at 0.05.

Figure 16 also shows that setting the maximum α larger than 0.15 did not improve enrichment very much. It seems that 0.15 is the optimal maximum threshold for this data set. For other data sets, it might be necessary to impose an upper bound on α, e.g. 0.15, to reduce false positives. This result suggests that doing so does not affect negatively functional enrichment very much.

An analysis of the results shown in Figure 16 reveals that varying α up to 0.15 yielded four *new* patterns (000001, 022221, 20001, and 201002) with a 26 enriched genes in 5 enriched clusters and total enrichment score approximately 10.37. These patterns were not observed for any genes when α was fixed at 0.05. Thus, varying α not only increased enrichment of existing patterns, it also discovers new enriched patterns.

Assigning Multiple Patterns further Increases Enrichment

The motivation for assigning multiple patterns to a gene is that genes might have multiple biological functions or involve in multiple biological processes. If such is a case, we expect that such genes acquire several patterns with similarly high probabilities. We test this hypothesis by comparing gene-set enrichment of clusters produced by two different schemes: (i) one-pattern assignment scheme, which assigns only one pattern that is determined by real (experimental) replicates , and (ii) multiple-pattern assignment scheme, which assigns possibly multiple patterns, each of which is determined by replicates produced using bootstrap (as discussed in Methods). In comparing these two assignment schemes, we chose to determine patterns by varying α up to 0.15, since the

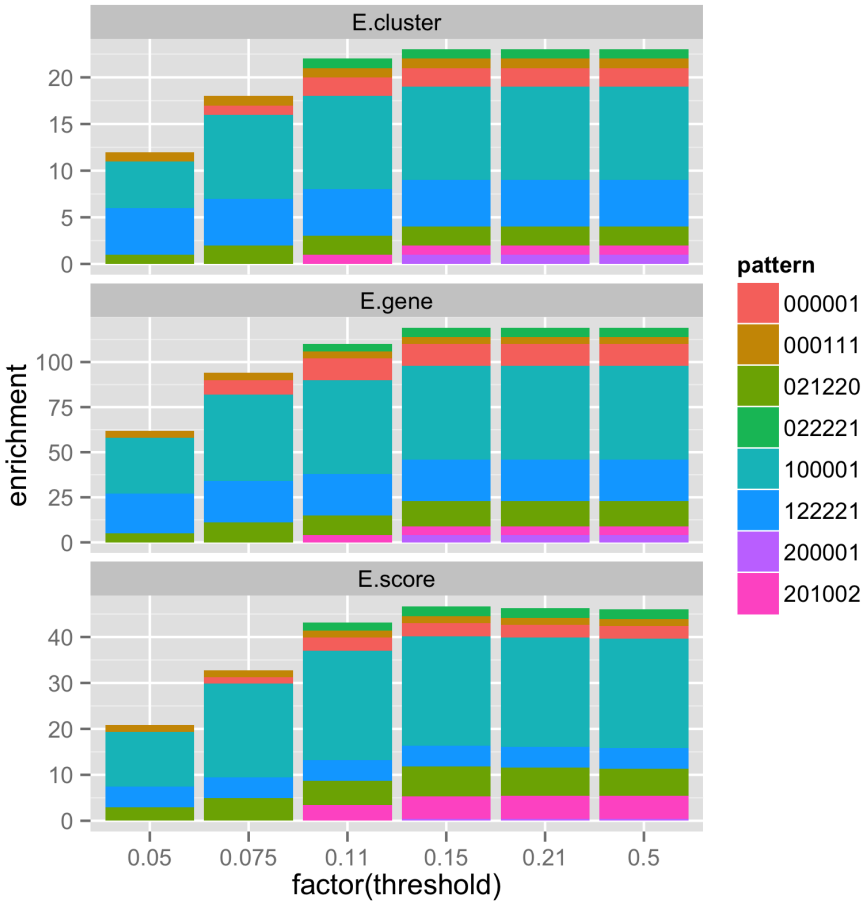

Figure 16: Comparison of gene-set enrichment of clusters produced by fixing α at 0.05 and varying α up to 0.5. The x-axis shows the maximum values of α that can be varied to obtain patterns. The y-axis shows the number of enriched clusters (*E.cluster*), the number of genes in enriched clusters (*E.gene*), and the enrichment score (*E.score*). Each color represents a different pattern, codified by a 6-digit string.

results have shown (as discussed in the previous section) that varying α with the upper bound of 0.15 resulted in most enrichment.

Figure 17 compares gene-set enrichment of the two assignment schemes. In the x-axis, we line up patterns with enriched clusters found by DAVID. Each bar corresponds on a pattern that contains enriched clusters. Patterns assigned to genes by the one-pattern assignment scheme are colored red. Patterns assigned to genes by the multiple-pattern assignment scheme are colored dark turquoise. Figure 17 shows clearly that the multiple-pattern assignment scheme resulted in much more *and* much better enrichment than the one-pattern assignment scheme. Specifically, we observe that

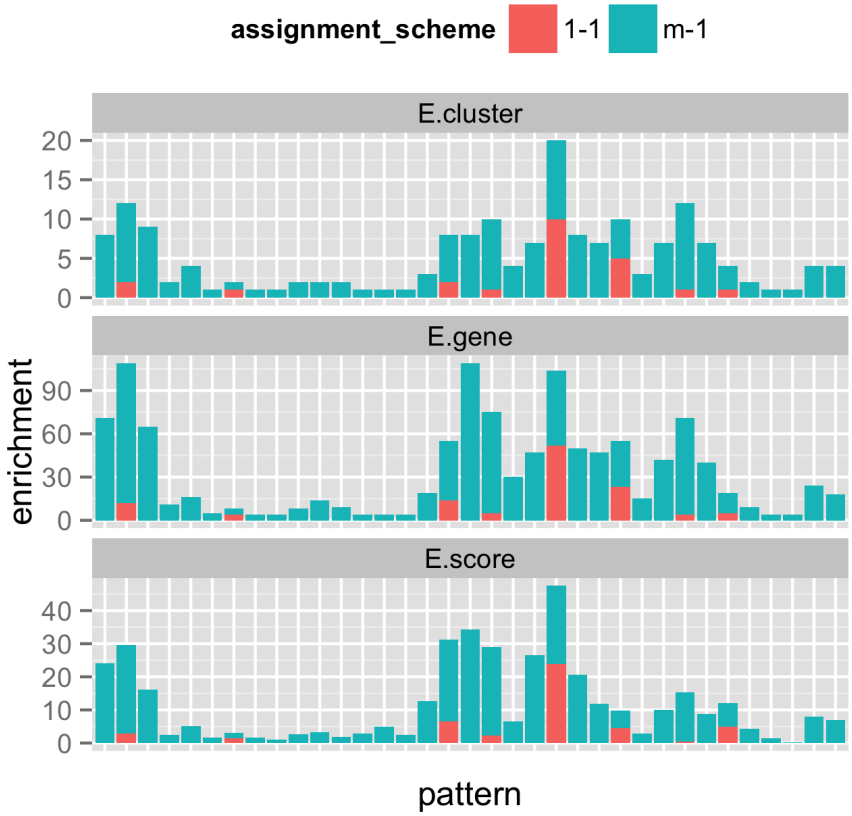

Figure 17: Comparison of gene-set enrichment of clusters produced by one-pattern assignment (1-1) versus multiple-pattern assignment (m-1), with $\alpha \leq 0.15$. The x-axis lines up patterns with enriched clusters found by DAVID. The y-axis shows the values of enrichment in terms of the number of enriched clusters (*E.cluster*), the number of genes in enriched clusters (*E.gene*), and the enrichment score (*E.score*).

- DAVID found enriched gene clusters in many more patterns assigned by multiple-pattern assignment scheme: 8 patterns by the one-pattern assignment scheme versus 35 patterns by the multiple-pattern assignment scheme. Further all of the 8 patterns were contained a subset of the 35 patterns. In other words, the multiple-pattern assignment scheme had *more enrichment* by discovering an additional 25 (3x more) more enriched patterns.

- Looking at each enriched pattern found in both schemes (the columns that contain both red and dark turquoise), we see that the enrichment for the multiple-pattern assignment scheme is much better in all 3 aspects: (i) the number of enriched clusters (*E.cluster*), (ii) the number of genes in enriched clusters (*E.gene*), and (iii)

the enrichment scores for of enriched clusters (*E.score*). In other words, the multiple-pattern assignment scheme had *better enrichment* for the enriched patterns that it shared with the one-pattern assignment scheme.

This comparison shows much promise for the probabilistic approach to assigning multiple patterns to genes. It appears that resampling by bootstrap was helpful in determining true response patterns of genes.

Clusters based on Orderable Patterns are Better Enriched

We may view the placement of genes into bins labelled by patterns as a clustering method because conceptually genes sharing the same patterns should share some biological functions. We shall analyze the predictive power of our method as a clustering method and compare the enrichment of gene clusters produced by this method and gene clusters produced by hierarchical clustering. Although there are better clustering methods for specific scenarios, hierarchical clustering remains popular as a general-purpose clustering technique and as such it is a useful gold standard for comparison. We used Pearson correlation as the distance between two gene expression vectors; each gene expression vector consists of mean expressions of all treatment groups of a gene. Average linkage was used as a measure of similarity between two clusters.

For comparison, it makes more sense to compare hierarchical clustering to our deterministic one-pattern assignment scheme, which assigns only one pattern to each gene, although the probabilistic, multiple-pattern assignment yielded much more enrichment in all aspects than the deterministic one-pattern assignment did. This comparison is more appropriate because hierarchical clustering assigns each gene to only one cluster and in essence is similar to the deterministic one-pattern assignment scheme.

For an appropriate comparison, we chose the hierarchy of clusters (for hierarchical clustering) that gave exactly 78 clusters. Since each hierarchy gives a different number of clusters, one must selected for comparison. And in our approach, an α threshold results in a specific number of clusters for set of differentially expressed genes. Specifically, if we

vary α to maximally 0.15 ($\alpha_{\max} = 0.15$), we get exactly 78 clusters for our dataset. Since we chose this threshold for comparison, we needed to chose an appropriate hierarchy for hierarchical clustering to give the same number of clusters.

As shown in Table 1, DAVID analyzed the set of clusters produced by hierarchical clustering and found 8 clusters that contained a total of 11 enriched clusters (denoted *E.cluster*). One cluster contained 3 enriched clusters; another contained 2 enriched clusters; and each of the other contained 1 enriched cluster. These enriched clusters contained between 4 and 16 genes (denoted *E.genes*). The enrichment score (*E.score*) for each enriched cluster was between 0.58 and 5.82.

Table 1: Functional enrichment of clusters produced by hierarchical clustering (hc) and our method when α is varied up to 0.05 for each gene.

method	cluster-id	E.cluster	E.gene	E.score
hc	c70	3	16	5.8176
hc	c35	2	8	3.8396
hc	c11	1	4	3.0557
hc	c72	1	4	2.6481
hc	c34	1	4	1.2998
hc	c71	1	4	1.0265
hc	c26	1	4	0.8574
hc	c77	1	4	0.5839
$\alpha \leq 0.15$	100001	10	52	23.8291
$\alpha \leq 0.15$	021220	2	14	6.4583
$\alpha \leq 0.15$	201002	1	5	4.9576
$\alpha \leq 0.15$	122221	5	23	4.5289
$\alpha \leq 0.15$	000001	2	12	2.8358
$\alpha \leq 0.15$	022221	1	5	2.1743
$\alpha \leq 0.15$	000111	1	4	1.5216
$\alpha \leq 0.15$	200001	1	4	0.4007

Very interestingly, DAVID also found 8 clusters in the set of clusters produced by our method. These 8 clusters, however, contained a total of 23 enriched clusters; this is more than double the number of enriched clusters for hierarchical clustering. These 23 enriched clusters contained a total of 119 enriched genes; this is also more than double the number of enriched genes for hierarchical clustering. The individual and overall

enrichment scores of enriched clusters in our method are again more than twice those in hierarchical clustering. In summary, our method with $\alpha \leq 0.15$ produced clusters that are more than twice as enriched as those produced by hierarchical clustering.

An important distinction between clustering by comparison-based patterns and by hierarchical clustering is that comparison-based patterns clusters are labelled, while clusters produced by hierarchical clustering (or any other *unsupervised* method) are unlabeled. The labels of comparison-based clusters are patterns that specify precisely how genes placed in such clusters respond to all treatment pairs. Such labelled can be meaningful annotations and carry useful meanings for subsequent analyses. By contrast, for hierarchical clustering, other than the fact that genes in the same clusters are similar based on Pearson correlation (some other distance metric), it is harder to interpret genes that are in the same clusters.

A closer analysis of genes in enriched clusters produced by the two methods shows that 28 out of 48 genes in enriched clusters produced by hierarchical clustering were also in enriched clustering found by our method. Hierarchical clustering discovered 20 genes in enriched clusters that our method did not. On the other hand, our method discover 99 genes in enriched clusters that hierarchical clustering did not.

Biological Associations of Genes in Discovered Patterns

Additional analyses of genes in enriched clusters discovered by our method and missed by hierarchical clustering showed that some of these genes were in fact associated in multiple biological networks and had interesting known pharmacological behaviors. The pattern shown in Figure 18 includes 62 genes, among which 4 transcription factor-encoding genes (Nfe2l2, Klf2, Egr1, and Irf8) belong to an enriched cluster found by DAVID. These 4 transcription factors were grouped together in one cluster by our method, but were placed in different clusters by hierarchical clustering.

Three of the four transcription factors (Nfe2l2, Klf2, and Egr1) have been known to participate in five biological networks (Figure 19): (i) co-expression network (violet

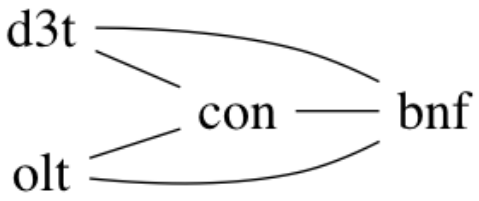

Figure 18: Pattern contains 4 transcription factors Nfe2l2, Klf2, Egr1, and Irf8 in an enriched cluster found by DAVID. These genes participate in multiple biological networks.

edges) supported by 20 publications; (ii) predicted functional network (yellow edges) supported by 13 publications; (iii) protein-protein interaction network (pink edges) supported by 5 data sources, (iv) co-localization network (blue edges) supported by 1 publication, and (v) shared protein domain network (olive edges) supported 2 data sources. These networks were obtained by GeneMANIA [82] based on the current literature and many different sources of functional association data. An interesting finding from these networks is that Egr1, Klf2, and two other genes Sp1 and Egr2 mutually share protein domains.

Among these transcription factors, Nfe2l2 is known to play an important role in many reported pharmacological activities by the studied chemo-prenventive chemical compounds (BNF, D3T, and OLT). Nfe2l2 is a master regulator of the expression of antioxidant response element-dependent genes, which produce proteins responsible for the detoxication of electrophiles and reactive oxygen species[86, 87]. Induction of studied chemical treatments BNF, D3T, and OLT on Nfe2l2 were revealed in many studies. Dewa et al. [88] revealed that BNF induced many Nfe2l2-regulated genes in a study of oxidative stress responses in the livers of rats. Kobayashi et al. [89] showed that Nfe2l2 (together with Keap1) was activated in response to D3T treatment in a study on Zebrafish. Tran et al. [21] also confirmed that many Nfe2l2-dependent genes, particularly detoxifying and antioxidant proteins, respond to D3T and OLT treatments in a study to examine pharmacological structure-activity relationships of these compounds in rat livers. Dong et

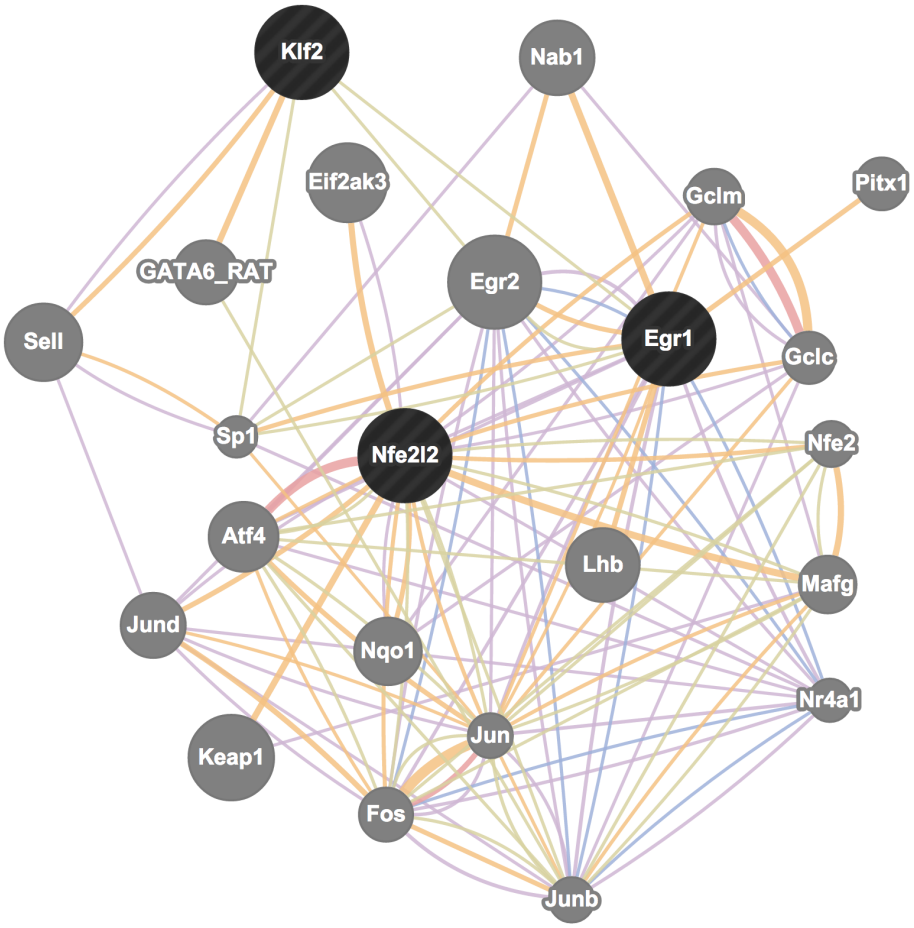

Figure 19: Biological networks that include Nfe2l2, Klf2, Egr1, and Irf8. Edge colors encode different types of networks. Related genes in the networks are in grey circles.

al. [90] demonstrated that D3T can induce Nrf2 activation and prevent ethanol-induced oxidative stress and apoptosis in a study on PC12 cells. The results of these studies support the hypothesis that Nfe2l2 plays an important role in pharmacological activities by the studied chemical treatments BNF, D3T, and OLT.

2.4.4 Summary

We introduce a novel method for the analysis of gene expression patterns in studies involving multiple treatments. We derived crucial properties that *orderable* comparison-based patterns are more likely to be true patterns. This property helps exploit the interdependencies among statistical tests whose outcomes constitute observed patterns. Consequently, we are able to increase α, the threshold used in the statistical tests,

beyond the traditional value of 0.05 to enable a more accurate prediction of true patterns. Objective analyses by DAVID confirmed that increasing α carefully in this way indeed yielded more functional enrichment.

Another novel aspect of this method is in the probabilistic assignment of multiple patterns to each gene. While fuzzy clustering of genes have been introduced, multiple pattern assignment based on bootstrap resampling is a refreshing direction. We further showed that the multiple-pattern assignment scheme increased functional enrichment manyfold. By assigning multiple patterns to a gene, the method discovered new enriched patterns and at the same time increased the enrichment of already discovered clusters.

A comparison to hierarchical clustering shows that the one-pattern assignment scheme produced the same number of enriched clusters, but those clusters were twice as enriched in all three different aspects: the number of enriched groups in each cluster, the number of genes in each enriched group, and the total enrichment score of each group. We compared hierarchical clustering to the one-pattern assignment scheme because both methods place a gene into one cluster, although the multiple-pattern assignment scheme is much better than the one-pattern assignment scheme in terms of producing functionally enriched gene sets. More thorough analysis of 3 transcription factors grouped together in an enriched cluster found by our method and missed by hierarchical clustering showed that these transcription factors participated in multiple biological networks. One particular transcription factor, Nfe2l2, was known to play an important role in many pharmacological activities related to the studied chemo-preventive chemical compounds.

This method is particularly useful in studies with many drugs, with small sample sizes or contain highly variantly expressed genes. If the sample size is sufficient but still small enough that conventional methods might suffer to gauge true response patterns, our approach might be beneficial in its exploitation of the dependencies among interdependent statistical tests. Specifically, the exploitation of orderable patterns will help vary α in an intelligent manner to predict true order of gene response to treatments accurately.

2.5 Conclusion

In this chapter, we have introduced several methods and tools for representing, analyzing, and interpreting gene expression with multiple treatments. We exploited directed graphs and orderable sets, respectively, to represent patterns of gene response and to deal with the challenge of predicting true patterns of gene response to treatments.

Firstly, we showed that by augmenting experimental replicates with synthetic replicates, correct outcomes of pairwise comparison, which can be represented by directed graphs, can be predicted with good accuracy. We then showed by biological analyses that, objectively, predicted patterns resulted in more and better functionally enriched clusters. Secondly, we demonstrated that varying α and assigning multiple patterns to genes ultimately results in gene patterns, which are represented by orderable sets, with more functional enrichment than conventional methods. We then showed that grouping genes with same patterns into clusters will yield much more enriched clusters than hierarchical clustering does. Furthermore, an analysis of genes in enriched clusters identified by our method but missed by hierarchical clustering showed interestingly that their pharmacological activities reported by the current literature agree with the patterns predicted by our method. These methods should be useful for any studies that use expression data to analyze comparatively patterns of gene response to multiple treatments.

Additionally, we have introduced **mDAG** and **PA**, two web-based software packages to facilitate analysis, visualization, and interpretation of gene expression data with directed graphs. The gene patterns may be further analyzed using several external well-established tools such as DAVID [81] or GeneMANIA [82], which have been incorporated into mDAG and PA to help users explore hypotheses about gene functions and regulations as well as other downstream analyses. mDAG and PA are useful for any studies that comparatively analyze response patterns in gene expression data with multiple treatments such as chemicals, cell types.

Chapter 3

Short Read Alignment

Short read alignment is the process of aligning *short reads*, which are short fragments extracted from DNA (RNA) of an individual, to a reference genome (transcriptome) from the same species. This problem is motivated by the fact that genomes within the same species are very similar, therefore the reference genome (which is constructed before) can facilitate analyzing genomes or transcriptomes of new individuals using short reads sequenced across their genome or transcriptome. Results from the alignment process can be used for a wide range of problems, from gene expression analysis, genome analysis, to many more problems in genomic and bioinformatics research. For example, mapping RNA short reads to the reference genome or transcriptome is the first step of the standard approach to measuring gene expression levels using RNA-Seq technologies. Another example is, mapping DNA short reads to the reference genome is the first step of the standard approach to identifying genetic variants using NGS data. The alignment of short reads to reference genomes therefore plays an important role in bioinformatics and biomedical research and applications.

Currently genomes for a huge number of individuals from many species have been sequenced using next-generation sequencing technologies. With this increasing amount of sequencing data, a lot of methods together with software packages for read alignment have been developed. They employed diverse data structures and algorithms for efficiently and accurately aligning reads to the reference. Although these methods and tools have been proven to work well with the majority of current data sets, many challenges still exist. One needs to develop efficient but accurate algorithms or needs to deal with sequencing error of input data. The complexity of the reference genome itself (e.g., the existence of long repetitive regions) and its incompleteness also put further difficulty to accurately aligning reads to the reference. Therefore it is still desirable to develop new algorithms and tools that perform well across a wide range of data aspects.

In this chapter, we introduce our approach to aligning short reads to reference genomes and our analysis of relationship between sequence complexity and performance of short read aligners. Most of content of this chapter has been published in [63, 64, 65, 66]. In the next sections, we describe in detail this work. In Section 3.1 we describe an overview of the problem. In Sections 3.2 and 3.3, we introduce a randomized approach to improving accuracy of the alignment of short reads to the reference genome across a wide range of read lengths and base error rates as well as an analysis of relationship between sequence complexity and performance of short read aligners. Finally, Section 3.4 provides conclusion and future directions of our work.

3.1 Introduction

As mentioned earlier, the alignment of short reads to the reference genome is a critical task in many bioinformatics and genomic applications. However, there are many challenges in obtaining accurate alignment of reads to the reference. Current technologies produce reads with lengths much shorter than genomes (typically read lengths are hundreds or thousands while genome lengths are millions to billions) which make read aligners struggle to obtain accurate alignment. Even if advances in technologies produce longer and more accurate reads, many challenges still remain. For example, complex genomic variants between individuals in population such as insertions or deletions or other types of structural variations can complicate the alignment. Additionally, the repetitive regions and the incomplete regions of the reference genomes are still sources of erroneous alignment despite a lot of effort in developing advanced read aligners.

To address these problems, researchers have proposed many methods together with software packages, most of them are based on the *seed-and-extend* approach. Several data structures and techniques such as the hash tables, q-grams, suffix trees/suffix arrays, or FM index are used to speed up searching for seeds [31, 32, 34, 33, 39, 42, 37]. For example, Bowtie uses the FM index to build a permanent index of the reference genome. It then applies backtracking algorithm to find alignments. BWA also utilizes the BWT, but

unlike Bowtie, can handle gaps and mismatches in the reads. More advanced versions of these methods, which are designed to work with longer reads, have been developed including Bowtie2 [40] and BWA-SW [38]. Additional heuristics are also used to enhance efficiency. Bowtie2 [40] and CUSHAW2 [41], for example, use several filtering techniques with *seeds* to quickly identify true candidates for alignment. GASSST [39] uses a filtering technique to reduce noisy seeds. For the *extend* phase, most of methods employed alignment algorithms such as the Needleman-Wunsch or the Smith-Waterman to calculate similarities (dissimilarities) between reads and the reference and then obtain the final alignment. Implementations of some of these approaches, e.g. Bowtie2 or CUSHAW2, take advantages of parallelism or special-purpose architectures.

Although these approaches and tools have been proven to work well with vast of current data sets, many challenges still exist. The rapid advancement of sequencing technologies continually changes many sequencing aspects such as read lengths, sequencing errors, or type of errors. Advanced technologies generally produce longer reads, which facilitate alignment of reads to complex reference genomes, but may have worse accuracy in some cases. On the other hand, technologies that produce shorter reads can be less expensive, and might have better accuracy. Nevertheless, many existing algorithms struggle to perform consistently across such aspects. Thus, it is necessary to design algorithms and tools that perform well across a wide range of input aspects.

One more problem is, although it is frequently observed that accurately aligning short reads to genomes becomes harder if they contain complex repeat patterns, there has not been much effort to quantify the relationship between complexity of genomes and hardness of short read alignment. Existing measures of sequence complexity seem unsuitable for the understanding and quantification of this relationship. Therefore, it is necessary to come up with a better formal establishment of connection between genome complexity and the accuracy of short read aligners.

In the next sections, we will describe our contributions to these problems.

3.2 RandAL: A Randomized Approach to Aligning Reads to Reference Genomes

3.2.1 Introduction

Currently many existing algorithms for read alignment struggle to perform consistently across a wide range of read lengths and sequencing error rates. Methods such as Bowtie [35] and Burrows-Wheeler Alignment (BWA) [36] tend to perform better with shorter reads. More advanced versions of these methods include Bowtie2 [40] and BWA-SW [38] are designed to work with longer reads. Bowtie2 can align reads with gaps and works better than Bowtie at longer reads. BWA-SW exploits several heuristics to speed up the local alignment of reads and obtain accurate alignments. The use of heuristics can improve performance several folds, but might lead to over-tuning parameters to a particular set of inputs, e.g. read lengths, species, or base error rates.

Here we introduce a novel algorithm for short read alignment that performs consistently well over a wide range of read lengths, from 35 to several hundreds base pairs. We employ two FM indices for efficient bidirectional exact substring matching. To deal with inexact matching (i.e. allowing gaps), first, we find common substrings between reads and the reference genome. Then, these common substrings are extended to complete alignments based on a bounded threshold on the edit distance. We use a special pruning mechanism to shorten vastly the running time of computing edit distances in a vast majority of cases. The use of randomization in aligning reads to genomes increases the probability of finding seeds quickly and enables us determine methodologically important parameters to speed up the entire alignment process. Our experimental results show that our algorithm performed consistently well on a wide range of read lengths across several bacterial and eukaryotic genomes. The alignment quality of our method was better or generally as good as that of all compared methods. We have also developed RandAL, a software package for aligning short reads to the reference genome based on this approach. The tool and its source code are publicly available at http://github.com/namsyvo/RandAL.

3.2.2 Method

Given a reference genome $\mathcal{S}$ and a set of reads $\mathbf{R} = \{r_1, \cdots, r_n\}$, the main problem is to align each r_i to $\mathcal{S}$. The reference genome $\mathcal{S}$ and the reads are DNA sequences, or strings over the alphabet of characters $\Sigma = \{A, G, C, T, N\}$. The alignment of a read r to $\mathcal{S}$ is essentially finding a substring of $\mathcal{S}$ that matches r the most. The set of reads $\mathbf{R}$ are substrings of another genome $\mathcal{R}$ that is different from, but belongs to the same species as $\mathcal{S}$ (that is, $\mathcal{R}$ differs from $\mathcal{S}$ only several percent of length and the differences must have biological meaning). Our strategy for read alignment is based on the following ideas:

1. Detection of identical substring matches between r and $\mathcal{S}$ is based on common substrings of r and $\mathcal{S}$. As we know r and $\mathcal{S}$ differ only slightly, we expect long common substrings exist.

2. A special data structure called the FM-index is used to facilitate memory-efficient, time-optimal exact string matching. One can use FM-index to find common strings between r and $\mathcal{S}$ in linear time with regard to $|r|$, which is much smaller than $|\mathcal{S}|$, therefore facilitates efficient detection of the common substrings.

3. Randomization is employed to find long common substrings between r and $\mathcal{S}$ efficiently and empowers us to methodologically determine important parameters that are used in critical steps of the algorithm. This translates into consistent performance in terms of time and accuracy across different species.

4. To account for insertion/deletion polymorphisms as extending the common substrings, we utilize the edit distance algorithm. Additionally, we employ a pruning heuristic to shorten the computation of edit distance, without compromising quality of alignment.

These ideas will be discussed in greater detail in the following sections.

Indexing the Reference Genome

Naive string matching takes quadratic time and therefore is too costly. Researchers have used data structures such as suffix tree, suffix array, and FM index to speed up string matching significantly. The FM index [29] in particular is desirable because it allows exact string matching to be done optimally in $O(m)$ time, where m is the length of the query (i.e. the read), and is very space efficient. The FM index of the genome is a substring index that takes advantage of properties of the Burrows-Wheeler Transform (BWT) to search incrementally all suffices of a read in the reference genome *backwardly* using the *last to front* mapping:

$$LF(i) = C[query[i]] + Occ(query[i], i)$$

where $C[x]$ is the number of occurrences of characters that are lexically smaller than x in the genome, and $Occ(x, i)$ is the number of occurrences of x in the first i characters of the Burrow-Wheeler transform of the genome.

By design, the search direction is in reverse (backward) order with respect to the sequence. To facilitate bidirectional string matching (to be discussed next), we employ two *FM indices*. A conventional FM index that traces substring matches *backward* is denoted as $\mathfrak{B}$. To facilitate searching in the forward dimension, we created an FM index for the reverse of the reference genome, $\mathcal{S}$. Searching using this index, denoted as $\mathfrak{F}$, is equivalent to search in the forward direction in $\mathcal{S}$. The pair of indices $(\mathfrak{F}, \mathfrak{B})$ helps us identify long identical stretches of DNA in the reference genome $\mathcal{S}$ and each read r_i.

Finding Common Substrings between Reads and Genomes

Given a read r and a specific position p in r, Algorithm 6 outlines the steps in finding longest common substrings of r and the reference genome $\mathcal{S}$, *with respect to* p. Figure 20 illustrates this algorithm. Longest common substrings (with respect to p) are constructed by concatenating maximal matches between substrings of $\mathcal{S}$ and those of r,

which begin and end at p. Searching for matches between substrings of $\mathcal{S}$ and substrings of r is facilitated by the backward and forward FM indices $\mathfrak{B}$ and $\mathfrak{F}$. To save time and reduce false positives, we only consider common substrings with lengths at least W.

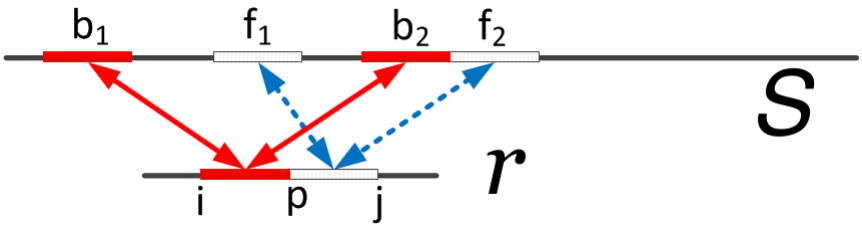

Figure 20: Finding longest common substrings (seeds): $r_{i\cdots p-1}$ and $r_{p\cdots j}$ may match several substrings of the genome $\mathcal{S}$, but fews of these (e.g. b_2 and f_2) form contiguous substrings, which is considered as seeds.

Algorithm 6: CommonSubstrings(read r, position p)

Let B be substrings of reference genome $\mathcal{S}$, which match exactly & maximally to $r_{i\cdots p-1}$. $\{B$ is found using $\mathfrak{B}\}$
Let F be substrings of reference genome $\mathcal{S}$, which match exactly & maximally to $r_{p\cdots j}$. $\{F$ is found using $\mathfrak{F}\}$
$M := \emptyset$
for each $b \in B$ **do**
  for each $f \in F$ **do**
    Let $s := b \oplus f$ be a concatenation of b and f.
    if s is a contiguous block in $\mathcal{S}$ and $|s| \geq W$ **then**
      $M := M \cup s$
return M

The choice of minimum length of common substring W is important. If W is too small, the set of common substrings M is large, and we will consider many common substrings between the read and the genome to construct alignments. The more common substrings we consider, the more likely we can find the correct position of the read in the genome to align; but we also more likely make mistakes of aligning the read to an incorrect position. In other words, with smaller W, we might get more true positives (correct alignments) and more false positives (incorrect alignments) at the same time. On the other hand, if W is too large, we might not be able to find any common substrings and

consequently unable to align the read to the genome. Therefore, inappropriate choices of W results in bad performance.

Our strategy for determining good values of W is based on randomization. As we shall see soon, the value p given to Algorithm 6 would be a random index of the read. To calculate W, first suppose that the correct substring of the reference genome $\mathcal{S}$ to align to the read r is r'. Let d be the edit distance between r and r'. These d mismatches divide r into $d+1$ blocks. Each block (except the last one) includes the closest mismatch to it. Let the sizes of the blocks be $m_1, m_2, \cdots, m_{d+1}$. We have $|r| = m = \sum_{i=1}^{d+1} m_i$.

The random choice of p implies that the common substring found by Algorithm 6 would be a random block, which is selected with probability $p_i = \frac{m_i}{m}$. This implies that the expected size of block i is $E[S_i] = m_i p_i = \frac{m_i^2}{m}$. Thus, the expected size of a random block, i.e. the expected length of the common substring, is $E[X] = \sum_{i=1}^{d+1} E[X_i] = \sum_{i=1}^{d+1} \frac{m_i^2}{m}$.

The Cauche-Schwarz inequality tells us that

$$\left(\sum_{i=1}^{d+1} \frac{1}{d+1}\frac{m_i}{m}\right)^2 \leq \sum_{i=1}^{d+1}(\frac{1}{d+1})^2 \sum_{i=1}^{d+1}(\frac{m_i}{m})^2$$

After simplifying, these imply that $E[S] \geq \frac{m}{d+1}$. In other words, we have established that:

Lemma: *The expected length of the common substring between a read and the reference genome found by Algorithm 6 is at least* $\frac{m}{d+1}$.

Although we do not know what d, the distance between r and its aligned substring r', is, it can be estimated by the rates of single nucleotide polymorphism (SNP) of the given genome and given rate of sequencing error. Let b be the rate of each nucleotide being mutated or sequenced erroneously, which we may assume to be distributed by a binomial distribution with mean $\mu = mb$ and variance $\sigma^2 = mb(1-b)$, where m is the read length. Therefore upper bound t of d might be estimated by $\mu + c\sigma$, for some

constant c. With 100,000 reads, we found that $c = 4$ produces good performance with high true positives and low false positives.

In summary, the two critical parameters of our method t and W are methodologically derived as follows:

- The upper bound of the distance between a read and its aligned string, $t = \lceil mb + 4\sqrt{mb(1-b)} \rceil$.
- The lower bound of the expected length of common substrings, $W \sim \frac{m}{t} \leq \frac{m}{d+1} \leq E[S]$.

W appears in Algorithm 6, and t appears in Algorithm 7, which is the next step after finding common substrings between reads and the reference genome.

Algorithm 7: AlignRead(read r)

```
p := 1
m := |r|
for i from 1 to A do
    C := ∅
    M := CommonSubstrings(r, p)
    for each s ∈ M, which is a substring of S do
        Let r_{i...j} be the substring of r that matches s exactly.
        Let s_L be the (i − 1)-substring of S, preceding s
        Let s_R be the (m − j)-substring of S, following s
        d := edit-dist(r_{1...i−1}, s_L) + edit-dist(r_{j+1...m}, s_R)
        if d ≤ t then
            C := C ∪ (s_L ⊕ s ⊕ s_R)
    if C has at least one sequences then
        Return "fail to align", if C has more than 2 sequences.
        Otherwise, align read r to each sequence of C. STOP.
    p := random(1,|r|)
return "fail to align"
```

Extending Common Substrings to Align Reads to the Reference Genomes

Using long exact common substrings as seeds to align reads to genomes is similar to [40, 41]. Our approach promises to be efficient because instead of exhaustively

traversing indices of a read to find optimal common substrings, we find common substrings with respect to random index p of the read.

In Algorithm 7, we iterate at most A times to find long common substrings between $\mathcal{S}$ and each read r. In each iteration, given a random position p, we invoke Algorithm 6 to find the longest common substrings (M) of $\mathcal{S}$ that match to a substring of r with respect to p. As illustrated in Figure 21 each string $s \in M$ corresponds exactly to a substring $r_{i\cdots j}$ of r. This exact match between $r_{i\cdots j}$ and s, naturally, pairs up $r_{1\cdots i-1}$ (a prefix of r) to s_L, the corresponding substring of $\mathcal{S}$ preceding s, and $r_{j+1\cdots m}$ (a suffix of r) to s_R, the corresponding substring of $\mathcal{S}$ following s. If the total edit distances of these two pairs are less than the previously calculated upper-bound t, we align r to the corresponding substring of $\mathcal{S}$.

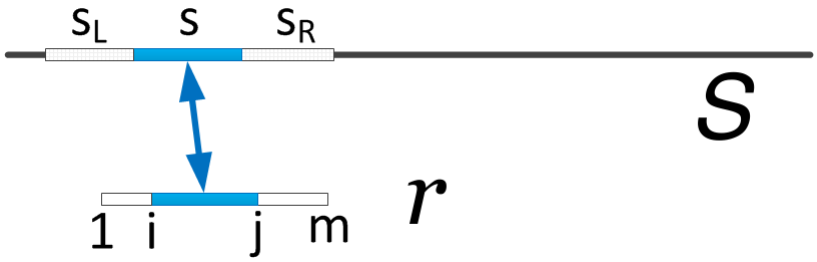

Figure 21: Extending common substring to alignment. Alignment of a read r to the reference genome $\mathcal{S}$ by extending a common substring of r and $\mathcal{S}$ (found in Algorithm 6). There are generally many substrings of $\mathcal{S}$ that match identically to the substring $r_{i\cdots j}$ of r.

Note that in the first iteration, the position p is 1 and not a random index of r. The reason for this is that we would like the method of finding long common substrings (Algorithm 6) to be *symmetrical* in the sense that b and f could "wrap around" r. In other words, when $p = 1$, b is a suffix of r and f is a prefix of r. In this case, the concatenation of $b \oplus f$ is not a contiguous substring, but rather two contiguous strings separated by a big gap. This conceptualization of "wrapping around" the read, or thinking of it as a circular instead of linear string, turns out to be quite effective in practice. In many cases, $p = 1$ leads to very long common substrings that lead to correct alignments of reads.

If we cannot align r to any substring of $\mathcal{S}$ after A attempts, then r is unaligned to $\mathcal{S}$, the reference genome. So, it is important to choose A appropriately. If A is too small, there will be many unaligned reads. If A is too large, the algorithm is slow. To select an appropriate value of A, let us again assume that the read and its correct alignment to the genome differ in d places (again $d \leq t$), consequently diving the reads into $d+1$ blocks. We want to select a value for A so that the longest block (longest common substring) can be sampled with high certainty. The probability that the longest block is selected (i.e. if a random index p lands inside it) is $\frac{m^*}{m}$, where m^* is the length of the longest block. On the other hand, the Pigeonhole Principle dictates that $m^* \geq \frac{m}{d+1}$ (Otherwise, the total lengths of $d+1$ blocks would be less than m.) This means, $d+1 \geq \frac{m}{m^*}$, which is the expected number of iterations to sample p to select the longest block.

Thus, setting $A = t+1 \geq d+1$, the longest common substring between a read and the genome is sampled expectedly after A iterations. Further, if $A = c \cdot (t+1)$, then the probability of landing in the longest block is exponentially increased as a function of c. Trading for speed, $c = 1$ seems to work fine in practice, because even if Algorithm 6 does not return the longest common substring, it is often possible to extend it to find the correct alignment for the read. But longest common substrings minimizes the chance of running into repeats in the genome; i.e. common substrings upon which extensions will lead to incorrect alignments.

Fast Heuristic for Computing Edit Distances

Computing edit distances consumes much time of the alignment algorithm (Algorithm 7). In steps 10-11 of Algorithm 7, we compute the edit distance between a read and a substring of the genome and discard it if the distance is greater than t. As each read often match with few substrings of the genome, we expect that such edit distances often exceed t. Examining lines 10-11 of Algorithm 7, we see that actually we do not need to compute the exact value of $d(x, y)$, the edit distance of x and y, as long as we can answer correctly the query $d(x, y) \leq t$.

We claim that the edit distance of x and y, $d(x, y) \leq t$ if and only if Bound$(x, y, t) \leq t$, where Bound is defined in Algorithm 8. To see this, observe that

- If $d(x, y) \leq t$, then Bound(x, y, t) returns $d(x, y)$.
- If $d(x, y) > t$, then Bound(x, y, t) returns either $d(x, y)$ or $t + 1$. The only difference between Bound and the conventional edit distance lies in line 6 of Algorithm 8. Analyzing line 5, we see that once $d_{i,j} > t$ for $1 \leq j \leq m$ (line 6), then $d_{m,m} > t$.

If $d(x, y) > t$, Bound(x, y, t) might not compute the edit distance correctly. Nevertheless, $d(x, y) \leq t$ if and only if Bound$(x, y, t) \leq t$. For aligning reads to bacterial genomes, Bound is much faster than the worst-case complexity $\Theta(m^2)$.

Algorithm 8: Bound(x, y, t)

```
1: d_{i,0} := 0 for 0 ≤ i ≤ |x|
2: d_{0,j} := 0 for 0 ≤ j ≤ |y|
3: for i := 0 to |x| do
4:    for j := 1 to |y| do
5:       d_{i,j} := min(d_{i-1,j-1}+(x_i == y_j), d_{i-1,j} + 1, d_{i,j-1}+1)
6:    return t + 1 if d_{i,j} > t for 1 ≤ j ≤ max{|x|, |y|}
7: return d_{|x|,|y|}
```

3.2.3 Results

One important performance comparison between our tool RandAL and other state-of-the-art tools is shown in Figure 22. This figure shows the average performance (*precision* in the y-axis versus *recall* in the x-axis) of 6 aligners across 6 different read lengths on 6 bacterial genomes (left) and 6 eukaryotic genomes (right), respectively. Both BWA and Bowtie, which designed for reads shorter than 100, showed a poor performance as read length increases. Bowtie did better at recall for longer reads than BWA, but it was still not competitive to the other 4 aligners, including RandAL. Among the top 3 aligners, Bowtie2, CUSHAW2, and RandAL, we can see that RandAL did noticeably better in terms of precision and was still competitive in terms of recall. More importantly, we can

see that across the wide range of read lengths from 35 to 400 for both bacterial and eukaryotic genomes, the performance of RandAL was consistently high in terms of both precision and recall. This consistency distinguished RandAL from the other top aligners.

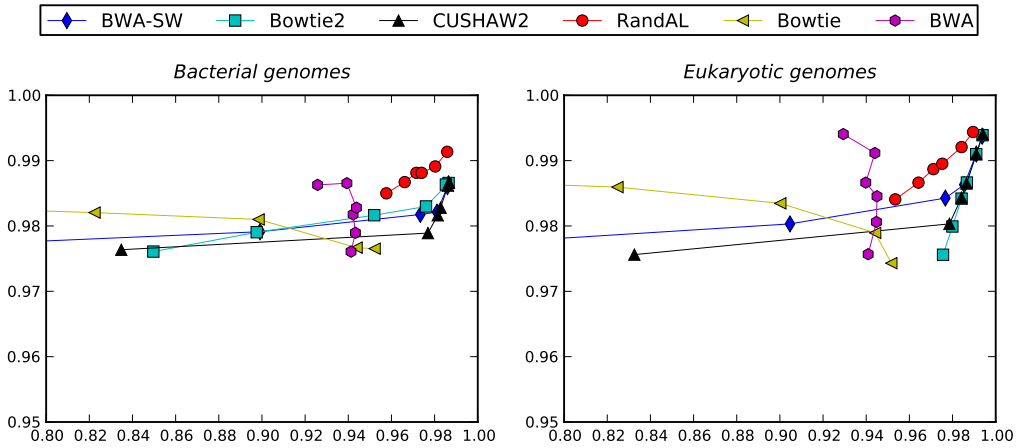

Figure 22: Alignment performance across 6 different read lengths. Recall (x-axis) versus Precision (y-axis) averaged across bacterial genomes and eukaryotic genomes, respectively, at read lengths of 35, 51, 76, 100, 200, and 400bp.

Additionally, our experimental results showed that, although eukaryotic genomes are expected to be harder to align than bacterial genomes, an examination of performance of the top 4 aligners including our method revealed that these aligners did not always perform better on bacterial genomes; eukaryotic genomes were not always harder to align. Using repeat density as a measure of sequence complexity, we showed that this measure correlated highly negatively with alignment quality (precision and recall). This implied that for larger and more complex genomes with many more repeats, these aligners will similarly suffer, as expected.

In practice, there are several problems need to be concerned. For example, the reads can be *single-end* or *paired-end*, or *mate-pair* reads. The advantages of pair-end or mate-pair reads can be exploited to further improve accuracy of alignment compared to single-end reads. Moreover, NGS platforms usually provide qualities of bases on reads. The quality of a base on reads is determined as a function of probability that the base is

wrong. Such qualities turn out to be very useful to improve accuracy of alignment. Characteristics of NGS data such as pair-end (or mate-pair) reads or base qualities will be further explored in our proposing work on variant calling.

RandAL is implemented in C++; FM-index codes are adapted from an external library (http://code.google.com/p/fmindex-plus-plus). We compared our method against several aligners including Bowtie [35], BWA [36], Bowtie2 [40], BWA-SW [38], and CUSHAW2 [41]. We chose these methods based on the fact that they are recently published, very popular and their software are available. Comparison tests were conducted on a workstation with two Intel Xeon E5-2680 2.70GHz CPU and 64 GB RAM.

Each aligner is tested with 100,000 simulated reads generated for each of 6 bacterial genomes and 6 chromosomes of eukaryotic genomes. Sizes of these genomes range from 1.3 and 28 millions bases; see Table 1. Genomes were obtained from EMBL-EBI (http://www.ebi.ac.uk/genomes). Since recent sequencing technologies produce read lengths ranging from 35 to 400bp at greater speed and lower cost than previous technologies (e.g. Sanger sequencing) [91], we choose this range of read lengths to evaluate the methods. More specifically, the reads were generated at lengths 35, 51, 76, 100, 200, and 400 as these lengths have been mentioned in the literatures. The *wgsim* C program, part of the SAMtool package [44], was used to generate reads.

Extensive comparisons were performed using SAMtool's default settings, with base error rate at 2%; 15% of polymorphisms are indels with lengths drawn from a geometric distribution with density $0.7 * 0.3^{l-1}$. Additionally, we present summary results for 1% and 4% base error rates with similar trends and conclusions.

Aligned reads from aligners are evaluated using the *wgsim_eval* Perl script, a part of the SAMtool package, using the default setting in which a read is mapped correctly if its mapping position is within a distance of 20 from the correct position. To compare alignment quality, we defined:

$$Precision = \frac{\text{\# correctly aligned reads}}{\text{\# aligned reads}}$$

$$Recall = \frac{\text{\# correctly aligned reads}}{\text{\# reads}}$$

Alignment Quality of 6 Aligners

At the outset, we found that Bowtie and BWA were decent performers when read lengths were short, i.e. between 31-56 bases. When read length increased, however, these two aligners were not competitive. Figure 23 shows the average performance (*precision* in the y-axis versus *recall* in the x-axis) of 6 aligners on 6 bacterial genomes and 6 eukaryotic genomes, respectively. Both BWA and Bowtie suffered from a decrease of precision as read length increases. For BWA, recall peaked at around 94% even at longer reads. Bowtie did better at recall for longer reads than BWA, but it was still not competitive to the other 4 aligners, including RandAL. Its bad performance at longer reads is unacceptable because technological trends tend to produce longer reads. For this reason, we will drop them out of head-to-head comparisons at this point, and will only compare the 4 best aligners: Bowtie2, BWA-SW, CUSHAW2 and RandAL.

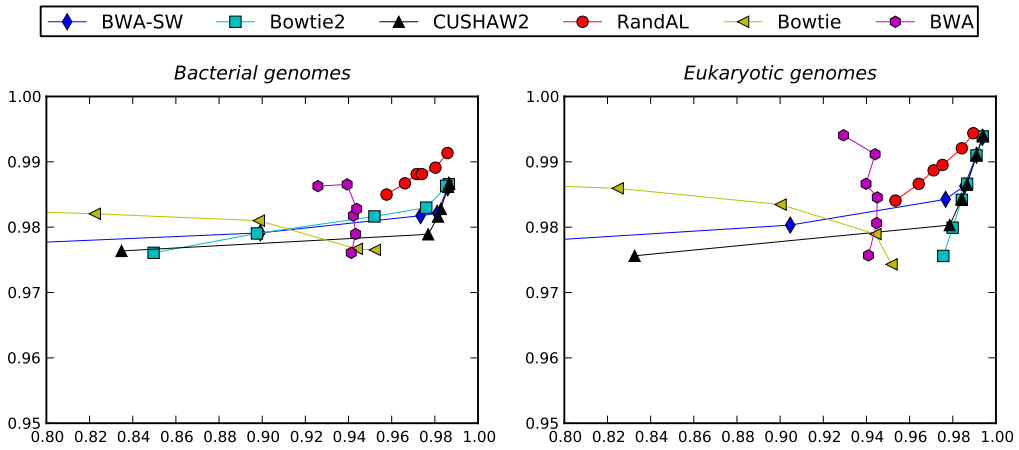

Figure 23: Alignment performance across 6 different read lengths. Recall (x-axis) versus Precision (y-axis) averaged across bacterial and eukaryotic genomes, respectively, at read lengths of 35, 51, 76, 100, 200, and 400bp.

A closer look at Figure 23 reveals that BWA-SW was relatively competitive but come roughly in the last place. There is no consistent winner (in terms of both precision and recall) among the top 3 performers, Bowtie2, CUSHAW2, and RandAL. Nevertheless, we can see that RandAL did noticeably better in terms of precision and was still competitive in terms of recall. Importantly, we see that across the wide range of read lengths from 35 to 400 for both bacterial and eukaryotic genomes, the performance of RandAL was consistently high in terms of both precision and recall; average precision was never below 0.98 and average recall was never below 0.95. This consistency distinguishes RandAL from the other top aligners.

An even closer look at individual bacterial and eukaryotic genomes (Figure 24) further reinforces the consistency in performance of RandAL. The lowest precision RandAL got in all 12 genomes was about 0.96, and the lowest recall was about 0.90. In comparison, for the other top performers, the lowest precision was about 0.93 and lowest recall was about 0.80.

All top 4 aligners perform really well in both precision and recall as read length increases. Their performance was quite similar at 400 read length. At shorter read lengths, however, RandAL outperformed the rest, often in both precision and recall.

Rates of Misalignment of Top 4 Aligners

Misalignment means aligning a read at an incorrect position. Misalignment increases the likelihood of running into problems later when we are interested in assembling reads into a complete genome and to identify where the constructed genome different from the reference genome (SNP calling).

The misalignment rate is calculated by dividing the number of incorrect aligned reads by the total number of reads. Figure 25 shows that averaging across all bacterial and eukaryotic genomes, RandAL got noticeably lower misalignments than the other aligners at all different read lengths. This result suggests that RandAL will likely work well with other tools and methods that require alignment of reads to reference genomes.

Alignment Quality at Different Base Error Rates

We have compared performance of 4 different methods using a base error rate of 2%; each nucleotide is mutated with the probability of 2%. Our analyses at different base error rates show similar behaviors as we have observed at 2% error rates. We present a summary analysis at two other base error rates of 1% and 4% at read lengths of 35bp, 100bp, and 400bp. Table 2 summarizes the average precision and recall of the top 4 aligners. These numbers suggest the followings:

1. All methods performed well at 1% base error rate.

2. With 4% base error rates, the other methods suffered, particularly with shorter reads. The best of them (Bowtie2) got ∼63% recall at 35bp. Low recall rate means few reads (out of the total number) were aligned correctly.

3. Our method consistently achieved the highest performance (or among the highest performance) across different read lengths and base error rates. In precision, our method always got the highest, consistently above 97.8%. In recall, even at worst case of 4% base error rate and 35bp read length, we got ∼94%.

Table 2: Average precision and recall at 1% and 4% base error rates

		35bp		100bp		400bp	
		Precision	Recall	Precision	Recall	Precision	Recall
1% base error	BWA-SW	97.60	82.86	98.30	98.29	98.98	98.98
	Bowtie2	97.60	93.40	98.31	98.25	99.00	**99.00**
	CUSHAW2	97.59	92.81	98.33	**98.33**	98.99	98.99
	RandAL	**98.88**	**95.49**	**99.09**	97.04	**99.18**	98.45
4% base error	BWA-SW	97.64	44.93	98.31	97.05	98.97	**98.96**
	Bowtie2	97.61	62.92	98.32	91.62	98.96	98.94
	CUSHAW2	97.67	60.67	98.34	**98.12**	98.95	98.95
	RandAL	**97.80**	**93.55**	**98.66**	97.48	**99.08**	98.48

Raw Running Times of Top 4 Aligners

Theoretically, asymptotic complexity of our method in aligning a read of length m is proportional to $m + m^2$. The worst case complexity of m^2 is due to edit distance

computation. The heuristic for computing edit distance, however, reduces this worst-case complexity significantly in practice. Our testing showed that the running times of other methods, like ours, did not depend much on genome sizes.

Table 3 shows the averaged running times (in seconds) of the 4 aligners in aligning 100,000 reads. Our method suffered with shorter reads, but were the second fastest with longer reads ($\geq$ 100bp). It seems that the benefit of randomization becomes more evident with longer reads.

Bowtie2 was the fastest across the board, but as shown in the previous section, its alignment quality is not as good as our method or CUSHAW2. Compared to ours, CUSHAW2 was significantly slower. Observing running times at different read lengths, we speculate that CUSHAW2 might be much be slower than ours with longer reads.

Table 3: Average running times of top 4 aligners at different read lengths

	35bp	51bp	76bp	100bp	200bp	400bp
BWA-SW	8.1	13.4	21.6	30.1	56.9	105.2
Bowtie2	2.8	4.1	5.8	8.1	18.3	41.6
CUSHAW2	4.2	7.8	12.7	19.3	67.8	228.5
RandAL	11.1	12.9	13.6	14.5	26.2	81.6

3.2.4 Summary

We introduced RandAL, a novel randomized approach to aligning reads to reference genomes. We showed that it performed among some of the top aligners that currently exist. Unlike the other aligners, however, RandAL distinctly performs consistently well across a wide range of parameters (read lengths and error rates) across all tested bacterial and eukaryotic genomes. As current sequencing technologies can produce reads in the tested range at low cost [91], our approach promises to work well with these technologies.

3.3 Correlation between Sequence Complexity and Alignment Performance

3.3.1 Introduction

Although eukaryotic genomes are expected to be harder to align than bacterial genomes, an examination of performance of the top 4 aligners in Figure 4 reveals that these aligners did not always perform better on bacterial genomes; eukaryotic genomes were not always harder to align. To quantify the degree of difficulty in aligning reads to genomes, we define *repeat density* as a measure of how many repeats a genome has. Since repeats directly affect alignment quality, the notion of repeat density is meant as a quantifiable approximation of genome complexity.

Here we formally have established a connection between sequence complexity and performance of read aligners. We found that *length-sensitive measures* of sequence complexity had the highest correlation to accuracy of alignment of short reads to reference genomes. This result allowed us to quantify the relationship between complexity of genomes and hardness of short read alignment. It also provided additional useful information for selecting suitable aligners for new genomes without running many of them to select the best one. This work has been published in ([65, 66]).

3.3.2 Method

Measures of Complexity

In our work, we have investigated several measures of complexity, including I-complexity, D-complexity, distinct substrings D_k, and repeat densities R_k.
I-complexity was introduced by Becher et al. as a measure of complexity of strings. It is proportional to the number of distinct substrings of the input string g:

$$I(g) = \sum_{i=1}^{|g|} \log_4(LCP[i] + 2) - \log_4(LCP[i] + 1)$$

where $LCP[i]$ is the length of the longest common prefix of the suffixes of g starting at positions $S[i]$ and at $S[i-1]$, and S is the suffix array of g.

We introduced a similar measure, D, counts directly the rate of distinct substrings:

$$D(g) = \frac{2 \cdot |\{x : f(x) > 0\}|}{|g| \cdot (|g| + 1)}$$

where $f(x)$ denotes the number of occurrences of x in g.

In addition to the I and D, we introduced two length-sensitive measures of sequence complexity, D_k and R_k. Given a number k, we defined D_k and R_k as follows, use the same notation of $f(x)$ in D:

$$D_k(g) = \frac{|\{x : f(x) > 0, |x| = k\}|}{|g| - k + 1} \quad and \quad R_k(g) = \frac{\sum_{f(x)>1,|x|=k} f(x)}{|g| - k + 1}$$

D_k and R_k measure the rates of distinct substrings and repeats, respectively, of length k. R_k is related to the function $C(k, r)$ proposed by Whiteford et al. $C(k, r)$ is the count of k-mers repeating exactly r times. Therefore, $R_k = \sum_{r>1} r \cdot C(k, r)$.

One important notice is, all measures I, D, D_k, and R_k could be computed in linear time and space using suffix and LCP arrays. More details of these measures can be found in our recent publications ([65, 66]).

From our experimental results, R_k and especially D_k, where k is similar to the read length, correlated very highly to alignment performance in terms of precision and recall. This result showed that the length-sensitive measures can provide additional information for choosing aligners that would align consistently accurately on new genomes. We also showed that these measures can be computed efficiently in linear time, making it useful in practice to estimate quickly the hardness of alignment for new genomes without having to align reads to them.

3.3.3 Results

Figure 26 shows the correlation between complexity measures, including D_k, R_k, D, I, and alignment precision at read length 100 for the 100 genomes. We can see that *length-sensitive measures* D_k and R_k have much higher correlations with alignment

precision than D and I for most aligners. Similar results were also observed with recall and with other read lengths. However, if recalls were comparative lower for several aligners with several lengths, their correlations were also comparative lower than the other aligners', especially if read length is very short. It is important to note that some aligners were designed to work optimally with longer reads and consequently do not work very well with shorter reads. One can conclude that if aligners perform predictably in their comfort zones, D_k (or R_k) is a good complexity measure that correlates highly to the accuracy of aligning reads to genomes.

To investigate how much repeat density correlates with the difficulty of aligning short reads to genomes, first, we computed R_k at each read length 35, 51, 76, 100, 200, and 400. To get a glimpse of its distribution, we show the values of R_k of the bacterial and eukaryotic genomes, for k's equal to these read lengths, in Table 4.

Table 4: Repeat density of genomes, R_k, at various length k

	Genome	Repeat Density at various k					
		35	51	76	100	200	400
Bacteria	*Wolbachia endosymbiont*	0.181	0.161	0.144	0.134	0.107	0.077
	Staphylococcus aureus...	0.064	0.058	0.053	0.050	0.043	0.036
	Escherichia coli 042	0.053	0.044	0.036	0.031	0.023	0.017
	Pseudomonas aeruginosa ...	0.041	0.037	0.033	0.031	0.026	0.021
	Streptomyces hygroscopicus ...	0.046	0.042	0.038	0.036	0.031	0.025
	Sorangium cellulosum ...	0.038	0.030	0.024	0.020	0.015	0.011
Eukaryota	*Debaryomyces hansenii* ...	0.036	0.032	0.028	0.025	0.019	0.013
	Ectocarpus siliculosus ...	0.092	0.073	0.056	0.046	0.030	0.020
	Schizosaccharomyces pombe ...	0.050	0.047	0.045	0.042	0.036	0.030
	Caenorhabditis elegans ...	0.138	0.105	0.080	0.066	0.039	0.024
	Taeniopygia guttata ...	0.129	0.100	0.070	0.050	0.017	0.002
	Drosophila melanogaster ...	0.068	0.065	0.062	0.060	0.052	0.042

Second, for each k, we computed the Pearson correlation between R_k of all bacterial and eukaryotic genomes and the performance (precision and recall) of each aligner on aligning reads of length k to the genomes. Table 5 shows that repeat density is correlated highly negatively to performance (precision and recall) of all aligners. This

means the larger $D(\mathcal{S}|k)$ is, the worse any aligner will perform. In other words, $D(\mathcal{S}|k)$ is good indicator of alignment difficulty. That said, we also observe that for BWA-SW at small lengths (k=35, 51), the negative correlation was weakest. This is probably due to BWA-SW trimming short reads.

Table 5: Pearson correlation coefficients of R_k and performance

		k=35	k=51	k=76	k=100	k=200	k=400
Correlation of repeat density and precision	BWA-SW	-0.94	-0.95	-0.96	-0.95	-0.97	-0.95
	Bowtie2	-0.94	-0.95	-0.96	-0.96	-0.97	-0.95
	CUSHAW2	-0.94	-0.94	-0.95	-0.95	-0.95	-0.93
	RandAL	-0.84	-0.83	-0.87	-0.88	-0.93	-0.94
Correlation of repeat density and recall	BWA-SW	-0.64	-0.37	-0.90	-0.95	-0.97	-0.95
	Bowtie2	-0.95	-0.94	-0.96	-0.97	-0.97	-0.96
	CUSHAW2	-0.95	-0.94	-0.95	-0.95	-0.95	-0.93
	RandAL	-0.95	-0.96	-0.97	-0.97	-0.97	-0.96

We also found that sequencing errors did not affect very much the correlation between repeat complexity and aligner's performance in terms of precision and accuracy. As summarized in Table 6, at each sequencing error rate, we found the correlation between the best R_k and alignment performance in terms of precision and accuracy were all very strong at between -0.95 and -0.98.

Table 6: Correlation between R_k and performance at different error rates.

	Corr. with Precision			Corr. with Accuracy		
base error rate	0.5	1.0	2.0	0.5	1.0	2.0
Bowtie2	-0.96	-0.98	-0.97	-0.96	-0.98	-0.97
BWA-SW	-0.95	-0.96	-0.97	-0.96	-0.98	-0.97
CUSHAW2	-0.95	-0.96	-0.96	-0.95	-0.96	-0.96

3.3.4 Summary

To our knowledge, this work is the first that establish formally and quantitatively the effect of repeats on the performance of short read alignment. Using repeat density as a measure of genome complexity, we showed that this measure correlated highly negatively with alignment quality (precision and recall). This implies that for larger and more

complex genomes with many more repeats, these aligners will similarly suffer, as expected. This approach helps researchers select appropriate aligners based on repeat complexity of genomes of interest, instead of relying on established evaluation of aligners' performance on known genomes that might have very different complexity compared to the genomes of interest.

3.4 Conclusion

In this chapter, we have introduced our approach to accurately aligning short reads to reference genomes and our analysis of relationship between sequence complexity and performance of short read aligners. We have also introduced RandAL, a software package for aligning short reads to the reference genome based on our approach. For the read alignment problem, our approach exploited a randomized strategy that helps efficiently finding seeds as well as enables effectively estimating key parameters for alignment process. It utilized two FM indices to facilitate efficient bidirectional searching from a random position on read. These improvements helped to align effectively and accurately short reads that come from a variety of technologies with different read lengths and rates of sequencing error. Our extensive comparisons showed that our aligner outperformed popular aligners in most instances and was unique in its consistent and accurate performance over a wide range of read lengths and error rates.

We also introduce an analysis of genome complexity which established formally and quantitatively the effect of repeats on the performance of short read alignment. We found a strong correlation between accuracy of short read aligners and repeat complexity. This work demonstrated that genome complexity and computational hardness of aligning short reads can be correlated and predicted well. High correlation of complexity and alignment accuracy provides practitioners an additional tool and criterion for evaluating aligners. The accurate prediction of performance can potentially help reduce computational cost by selecting appropriate aligners without running many of them.

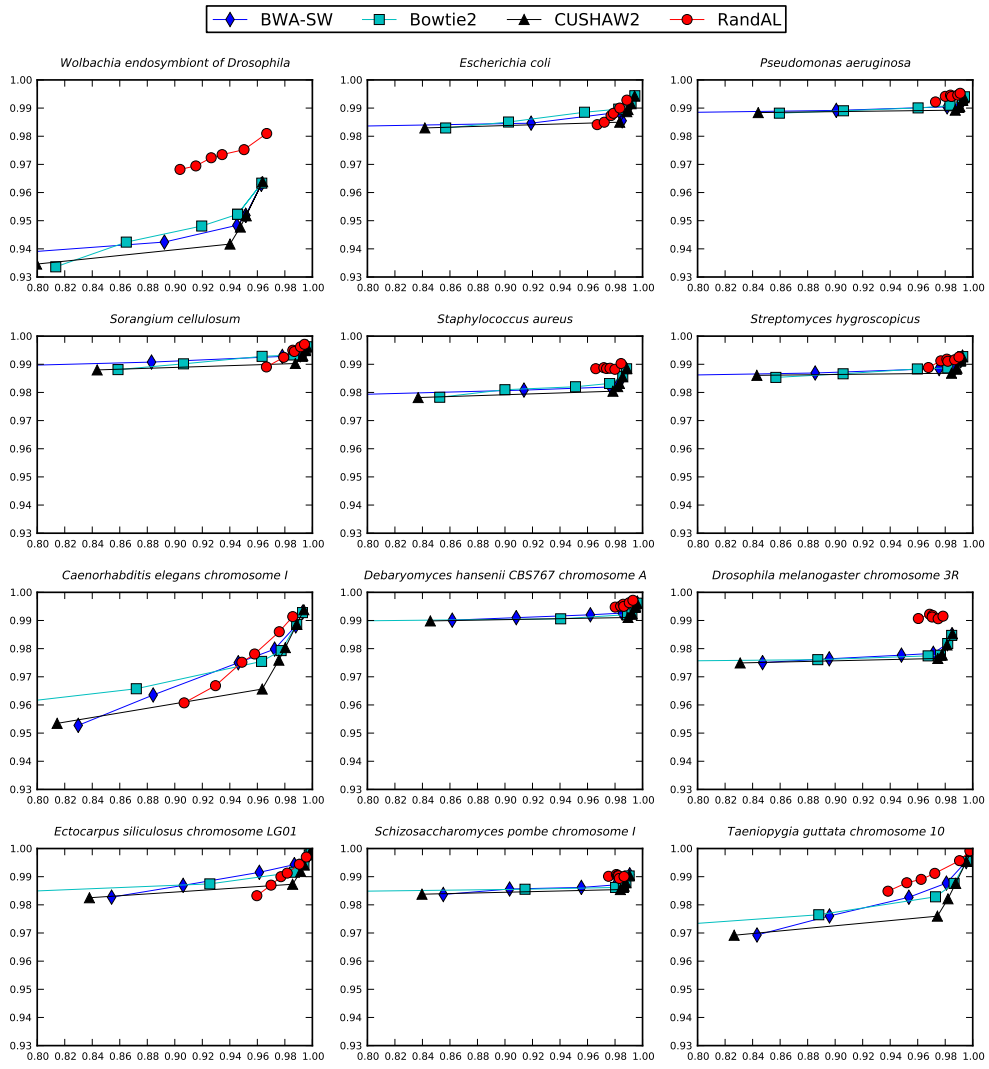

Figure 24: Recall versus Precision of Top 4 Aligners. Performance of top aligners on bacterial genomes (top 6 figures) and eukaryotic genomes (bottom 6 figures). X-axis is recall; Y-axis is precision.

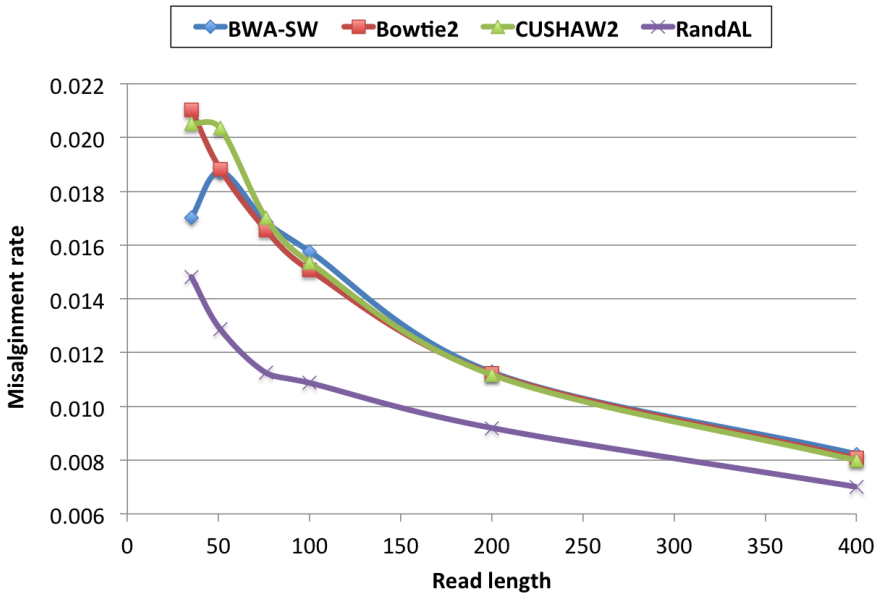

Figure 25: Rate of misalignment averaged across bacterial and eukaryotic genomes.

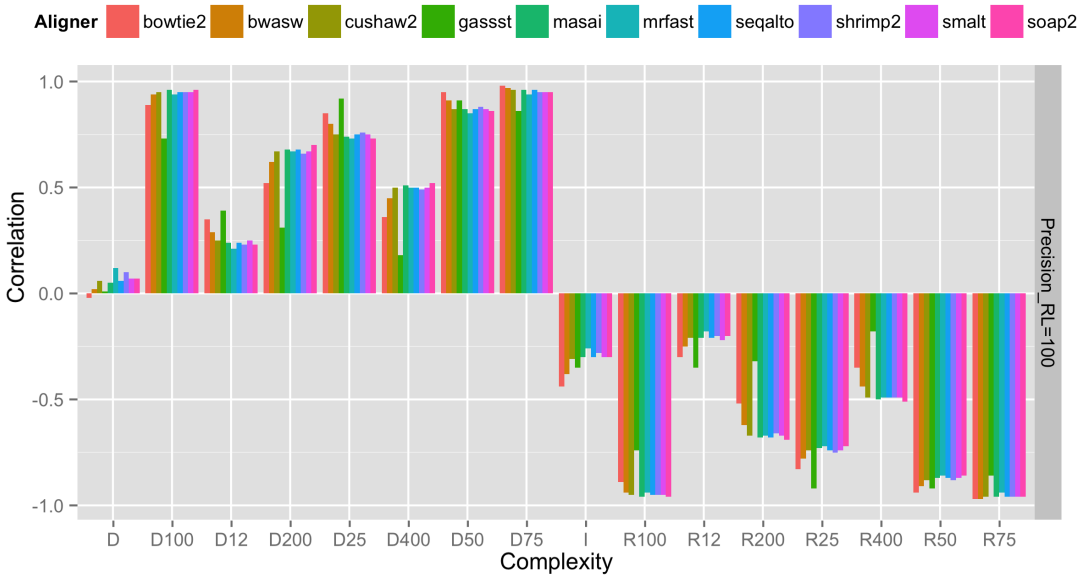

Figure 26: Correlation coefficients between different measures of complexity and aligners' performance (precision) at read length 100.

Chapter 4

Genomic Variant Detection

Genomic variant detection, so-called variant calling, is the process of detecting genomic variants between individuals in a population. Genomic variants are differences between DNA of individuals, which can be single nucleotide polymorphisms (SNPs) or structural variants such as insertions/deletions (INDELs), duplications, inversions or translocations of DNA sequences. Variant calling methods using the NGS data typically use read alignment as an initial step. The differences between the aligned reads and the reference genome are then analyzed to identify genomic variants. Genomic variant detection is an important study in many applications using next generation sequencing technologies [2]. This study has great significance in bioinformatics and results from this process can be used for a wider range of problems, from genetic research to medical field. For example, one can predict how genomic changes are related to phenotype in their organism of interest, or associate genetic changes to disease risk, medical treatment efficacy or other traits of interest [4, 92].

Currently a massive high-throughput sequencing data for humans and other species have been produced by next-generation sequencing technologies. With this huge amount of sequencing data, many algorithms and tools with several approaches have been developed for variant calling, most of them rely on statistical methods such as Bayesian methods or logistic regression models. Although many methods have been proven to work well with many datasets, they still have several disadvantages. For example, most of them call variants by analyzing aligned reads produced by an external aligner, which can limit the use of information from both read alignment and variant calling processes. One more problem is that current variant callers do not take full advantage of a huge amount of existing genomic variants, which are collected and validated in public databases such as dbSNPs, to support accurately detecting variants. Therefore it is still desirable to develop new algorithms and tools for accurately detecting genomic variants.

In this chapter, we describe our approach to improving accuracy of variant calling. Part of this work has been published in our recent publications [67]. In the next sections, we describe in detail our work. In Section 4.1 we describe an overview of the problem. In Section 4.2 we introduce an approach to improving accuracy of variant calling. In this approach, we integrate information of existing genomic variants into a unified process of read alignment and variant calling. Bayesian method is exploited to determine called variants and calculate their qualities. By proposing this approach, we aim to detect more accurate variants than state-of-the-art methods such as GATK [46] or SAMtools [44], especially with low coverage sequencing data. Section 4.3 describes our most up to date results of the method on both simulated and real data. Finally, Section 4.4 provides conclusion and future directions of our work.

4.1 Introduction

Recent advances in next-generation sequencing technologies (NGS) make it possible to provide cost-effective, high-throughput, and large-scale sequencing data for humans and other species. This facilitates a wide range of genomics and bioinformatics research including genomic variant detection. Many methods and tools with several approaches have been developed for calling variants using NGS data. Most of them are based on analyzing read alignment results using statistical methods such as Bayesian methods, hidden Markov models or logistic regression models [93, 48, 46, 47, 50, 54]. Although many methods and tools are well-established and have been proven to work well with many datasets, they still have several disadvantages.

A common weakness of current methods is that they detect variants based on analyzing aligned reads from external aligners such as our tool RandAL (Chapter 3). To speed up the alignment process, most aligners ignore the possible relation between reads (e.g., reads come from the same regions), which might produce inconsistent aligned reads and therefore complicate variant calling. There were several efforts to overcome this problem. Some methods perform realignment to recover misaligned reads at the region of

interests [94, 46]. Some other methods used de novo assembly to construct unitigs, and then variants can be called by mapping the unitigs against the reference genome [95, 96, 54]. Some researchers exploited a special data structure called alignment graphs to improve detection of long INDELs [97]. Researchers also exploited multi-samples to support variant detection [93, 50]. Nevertheless, they still have not solved the problem from the root since read alignment was already done *a priori*. This makes it difficult to take full advantage of input data to support whole variant calling process. Consequently, they usually require high sequencing coverage to be able to detect variants accurately. They also usually rely on complicated algorithms and require high-computational cost.

Another disadvantage of current methods is that they have not taken full advantage of known variant information to support variant calling. Currently, a lot of known variants in populations of humans and other species have been collected in public databases such as dbSNPs. For example, a large number of human genetic variants were detected and validated with a lot of efforts such as the 1000 Genomes Project [98]. Such variants were collected from a large numbers of samples thus a large amount of genetic variants are likely exist in these databases. Therefore it would be desirable to design methods which can efficiently exploit them to support calling variants in new genomes, especially INDELs and other structural variants such as inversions or transpositions. Nevertheless, the known variants have not been exploited efficiently to support variant calling although they have been used in some ways. For example, SOAPsnp [45] chose a prior-genotype probability by the use of dbSNP in the analysis of human data; while Atlas [99] used prior SNP probability and prior error probability in its training data sets. However, these tools have not used that known variant information to support analyzing aligned reads. Crossbow [100] proposed a method which can query the genomic variant databases to analyze the mismatches, but this approach had limits in improving variant detection. Recently, BWBBLE [43] were among a few tools which can use known variants to support read alignment, but this method has not been adapted to variant calling problem.

One more problem of current methods is in their conventional pipeline, in which variants are called through many separated steps including read alignment, alignment sorting, indel realignment, variant calling, and some other auxiliary steps. Although this pipeline provides flexibilities for variant calling task, it has several disadvantages. Firstly, it has some limits in incorporating information from one phase to others to help improve variant calling. Secondly, this requires a careful selection of a diverse set of tools, which turns out needs more human intervention that is error-prone. Although there were several efforts to deal with these problems, they did not solve them from the root. For example, Crossbow [100] is a cloud-computing software that combines the aligner Bowtie and the SNP caller SOAPsnp using Hadoop to call variants in parallel. Although this approach can help speed up the whole SNP calling process, the read alignment and SNP calling phases are still performed independently.

In addition to the above challenges, data related problems also put more challenges into variant calling problem. They include, but are not limited to, read length, sequencing errors, and incompleteness and misassembles of reference genomes. The repeats in genome sequences also put further difficulties to this task [3]. Therefore, it is desirable to design algorithms and tools to further improve accuracy of variant detection.

In this work, we introduce a novel approach that leverages known variant information to detect both *known* and *unknown* variants from NGS data. In this approach, we introduce a novel representation, which combines reference genomes and their associated known variant profiles in an appropriate manner, to help performing read alignment and variant calling efficiently and accurately. We also develop a novel algorithm, which can take into account known variant information, to align reads to the reference and update the variant profile *during* the alignment of reads. This strategy allows reads to be aligned more accurately, and variants can be called more accurately even with few numbers of reads (low-coverage, 5x or fewer) compared to popular methods. In particular, our method was better at calling INDELs that are otherwise hard to

call even at high coverage (often 20x or more). The method's high accuracy at low coverage can help reduce experimental cost of variant detection. We have also developed IVC, a software package for detecting genomic variants based on this approach. The tool and its source code are publicly available at http://github.com/namsyvo/IVC.

4.2 IVC: An Integrated Approach to Detection of Genomic Variants

The overall objective of our approach is to detect genetic variants from short reads that come from an individual's genome. The novelty of this approach is in the incorporation of known information of genetic variants, which are collected in public databases by efforts such as the 1000 Genomes Project, into the variant calling process. The main reason behind this idea is, we expect that information of known variants can help to identify the known variants of a new individual's genome, i.e., variants that are already exist in databases, more efficiently. More importantly, we also expect that by leveraging the known information in the variant calling process, we can detect *unknown* variants, i.e., variants that are not exist in databases, more accurately.

Given an individual's short-read dataset, a reference genome, and a known variant profile, our approach of detecting the individual's genetic variants can be described as followings. First, we create a *meta-genome* to represent the reference genome and known variant profile. This meta-genome is then used to build an index structure. An *iterated randomized algorithm* is then developed to search for *seeds* of alignment between reads and the reference with the assistance of that index structure. A specific-designed alignment algorithm, which can take into account information of known variant locations, is developed to fully align reads to the reference. Profiles of potential candidate variants (both known and unknown) are updated *dynamically* during the alignment process. As the alignment process has been finished, variants are called based on quality scores derived from the probability of each potential candidate variant being a true variant.

Our approach, called IVC (Integrated Variant Calling), includes two main technical contributions: (1) a *meta-genome* representation which combines the reference

genome and known variant profile in an appropriate manner, and (2) a specific-designed alignment algorithm which can take into account known variant locations in determining optimal pairwise alignment. Both of them are designed to efficiently and effectively exploit known variant information in read alignment and variant detection processes, which is difficult to do with standard reference genomes and conventional alignment algorithms. Two other contributions are also essential to help improving accuracy of variant calling: (3) an *iterated randomized algorithm* which can locates seeds accurately and efficiently, and (4) a strategy to update variant profiles *dynamically* during the alignment process, which helps improving accuracy of read alignment. These contributions will be described in the following sections.

4.2.1 Representing and Indexing Reference Genomes with Incorporated Known Genomic Variants

To incorporate known genomic variants into the reference genome, several approaches have been introduced. [101] proposed a graph representation produced by integrating multiple genomes. Several work addressed the problem of text indexing with wildcards in the context of short-read alignment such as [102, 103], however, they were able to deal with only single variations. Recently, [43] introduced a generalized FM index in which SNPs are represented by IUPAC symbols. While these approaches are theoretically interesting and they are elegant in the sense that each representation is one type of data structure, they either cannot fully describe SNPs and INDELs or are computationally inefficient or difficult to implement in practice.

Here we introduce a novel representation which incorporates variants into the reference genome in a simpler and more efficient way. First, we create a reference meta-genome $\mathcal{G}$ which is composed of characters A, C, G, T, N, and V. Positions with A, C, G, T, and N are corresponding to bases in the standard reference genome which are not marked as variant locations in the databases. Positions with V represent locations of known genetic variants in databases, which can be either SNPs or INDELs (other types of

variants will be added into the design in the future). Thus, a V character can represent multiple bases (e.g. A or C) in case of SNPs or multiple sequences of bases (e.g., AC or $ACGGT$) in case of INDELs. Together with this reference meta-genome, we create a hash table H with keys representing locations of known variants and values representing the variant profiles at corresponding locations. During the alignment of short reads to the reference, whenever a known variant location is reached, the hash table is used to retrieve variants at that location. The main advantage of using a hash table together with the meta-genome is that it allows us to retrieve information of known variants in a quick and simple way, although it might requires accessing both data structures during the alignment of reads to the reference.

In summary, our representation consists of two data structures:

1. A meta-genome $\mathcal{G}$, where $\mathcal{G}[i] = \mathcal{R}[i]$ if i is not a variant location and $\mathcal{G}[i] =$ "V" if i is a variant location, where $\mathcal{R}$ is the standard reference.

2. A hash table $\mathcal{H}$, where $\mathcal{H}[i]$ holds the content of variants at location i.

In this approach, mapping reads to the multi-genome might require assessing both data structures. However, it is efficient once we formulate an effective way to match approximately reads to the multi-genome.

Together with the meta-genome G and the hash table H, an *index* I is also created to speed up the alignment of short reads to the reference. The use of an index to facilitate short-read alignment is commonplace. The purpose of such indexes is to quickly identify long common substrings, known as *seeds*, between reads and the reference. Those seeds are then extended to find the complete alignment between reads and the reference. Researchers have used various types of data structures such as hash tables or FM-indexes to build such indexes [1]. A standard FM-index is a data structure built for a specific string to allow optimal linear storage and linear time substring query on that string [29]. In particular, given a string s, to find out if s is a substring of another string t, the FM-index

search algorithm starts at the end of s and proceeds in a *backward* fashion to identity all currently matched suffixes of s in t, using the FM-index data structure created from t. The search stops when either it cannot find any occurrences of the current suffix of s in t or it reaches beyond the beginning of s.

In our approach, we exploit the FM-index for indexing the meta-genome G, which consists of characters A, C, G, T, N, and V as described above, in which V is considered as a wildcard character. However, to facilitate alignment of reads to the reference meta-genome, we developed an *iterated randomized algorithm* to replace the traditional FM-index search algorithm. This algorithm turns out to be very efficient in our experimentation as described in next sections.

4.2.2 Overview of Our Variant Calling Algorithm

Given a reference meta-genome G, a hash table H storing information of known variant locations, and an index I, our integrated variant calling algorithm (IVC) takes as input a set of reads R and works as follows:

Algorithm 9: IVC(R)

1: Align each read in R to the reference meta-genome G using the index I and table H. The alignment of each read consists of two main steps: (1) searching for seeds using a randomized algorithm and (2) extending seeds into a full alignment using an asymmetric alignment algorithm.
2: In the extension phase, as locations of known variants are encountered (those with V characters), the variant profiles at those locations in H are updated using Bayesian method.
3: In the extension phase, as differences (mismatches or indels) between reads and the reference outside of known variant locations are detected, *new* variant profiles are created (for the first aligned reads) and updated (for subsequence aligned reads) using Bayesian method.
4: After all reads are aligned and variant profiles are updated completely, candidate variants with maximum probability are declared to be the called variants.

The details of this process are described in the following sections.

4.2.3 Aligning Reads to the Reference Meta-genome

The purpose of aligning reads to the reference meta-genome in our approach is to accurately keep track of genomic differences between reads and the reference. Our alignment strategy is similar to popular methods in its decomposition of the alignment process into two steps:

1. (*Seeding phase*) Searching for (appropriate) seeds of the alignment between reads and the reference meta-genome.

2. (*Extension phase*) Seeds are extended on both sides (left and right) on reads and on the reference into a complete alignment.

However, to improve accuracy of aligning reads to the reference meta-genome as well as detecting differences between them, in each phase we introduce several techniques which can efficiently deal with sequencing errors and genomic variants.

Algorithm 10 describes our alignment strategy for each input read r to the meta-genome $\mathcal{G}$. For all proper alignment found, differences (mismatches, insertions/deletions) between the read r and its mates on the meta-genome $\mathcal{G}$ will be obtained by tracing back the alignment. These differences will be used to call variants. Procedure $FindSeeds$ in line 3 is used to find seeds, procedures $LeftAlign$ and $RightAlign$ in line 8 are used to calculate edit distances between r and its mates on $\mathcal{G}$ on the left and the right of the seeds, and procedures $LATraceBack$ and $RATraceBack$ in line 10 are used to get alignment and therefore the differences between r and its mates. These procedures will be described in next sections.

In each iteration (line 1), we invoke Algorithm 11, which will be described in section 4.2.3, to find the seeds $r_{i\cdots j}$. Naturally, the seeds will pair up $r_{1\cdots i-1}$ (a prefix of r) to g_L, the corresponding substring of $\mathcal{G}$ preceding s, and $r_{j+1\cdots m}$ (a suffix of r) to g_R, the corresponding substring of $\mathcal{G}$ following s. Then, if the total edit distances of these two pairs, which are calculated by the Algorithm 12 described in section 4.2.3, are less than

Algorithm 10: AlignRead(read r)

```
1: for n from 1 to A do
2:    V := ∅
3:    S := FindSeeds(r)
4:    for each s ∈ S, which is a substring of meta-genome G do
5:       Let r_{i...j} be the substring of r that matches s exactly.
6:       Let g_L be the (i − 1 + e_L)-substring of G, preceding s
7:       Let g_R be the (m − j + e_R)-substring of G, following s
8:       d, T := LeftAlign(r_{1...i−1}, g_L) + RightAlign(r_{j+1...m}, g_R)
9:       if d ≤ D then
10:         V := V ∪ LATraceBack(r_{1...i−1}, g_L, T) ∪ RATraceBack(r_{j+1...m}, g_R, T)
11:   if V ≠ ∅ then
12:      return V
13: return ∅
```

the threshold D, we align r to $\mathcal{G}$ and then get the alignment by using Algorithm 13, which will be described in section 4.2.3. Notice that the choice of D, which will be described in the next section, is very important for the accuracy of alignment.

There are several notices with the Algorithm 10. We iterate at most A times to find approximate matches between $\mathcal{G}$ and each read r. If we cannot align r to any places in $\mathcal{G}$ after A attempts, then r is declared to be unaligned to $\mathcal{G}$. So, it is important to choose A appropriately. If A is too small, there will be many unaligned reads. If A is too large, the algorithm is slow. The value of A is chosen experimentally.

Seeding Phase

Many current alignment methods determine proper seeds based on long exact matches between reads and the reference genome. To speed up exact matching, several data structures such as suffix trees, suffix arrays, and FM indexes are used to build index of genomes [31, 32, 34, 33, 39, 42, 37]. Although suffix trees can be constructed in linear time and provide optimal query search (linear time in length of queries), they are complicated to build and are inefficient in terms of space. Suffix arrays can also be constructed optimally in linear time, but their constructions are much simpler than suffix trees. Suffix arrays can further be used to construct FM indexes in linear time, by first

constructing Burrow-Wheeler transform [29]. FM indexes are desirable because they are space efficient and allow exact string searching to be done optimally in linear time in length of queries. By design, exact searching using FM indexes is done *backwardly* starting from the last character of a query. The search will stop if it reaches to the first character of the query, or there is no more exact matches at a particular position on the query. A good strategy to determine long exact matches must not start near positions on reads that contain differences between reads and the reference. However, these differences, which come from either genomic variants or sequencing error, can occur at any positions. Thus, many seed finding methods are essentially brute-force. For example, CUSHAW2 [41] attempts to find long seeds by exhaustively starting at all read positions. Brute-force searches are computational expensive.

Our seed finding strategy is an *iterated randomized approach*, which is faster than the brute-force algorithm while having competitive accuracy of finding seeds. This algorithm is based on searching for exact substrings on an FM index with the following modifications:

1. **The searching for seeds starts** at a random position in r and proceeds in a *forward* fashion. Instead of considering all positions on the read in the brute-force manner to find the longest exact matches, this algorithm starts the search from a random position on the read. By repeating this several times, we can find the nearest longest exact matches after a number of random iterations which is much less than those number requiring by the brute-force manner. In our approach, we prefer the *forward* search from a position near the beginning of the read than the conventional backward search from the end of the read. The main advantage of doing this is making the search less likely encountering sequencing errors earlier. This is based on observation that probabilities of sequencing errors are likely increased as the position of bases increases from the beginning of the read [104]. The randomized strategy was also proven to be efficient in searching for seeds by [64].

2. **The searching for seeds ends** as either it reaches a V character (a known variant location), or it cannot find any occurrences of the current suffix of r in G, or it reaches beyond the end of r. To avoid as many as possible variant locations (one source of differences between reads and the reference) as searching for seeds, here we exploit one advantage of the meta-genome representation. By looking at V characters on the meta-genome G using the hash table H, we can stop the search for seeds whenever it reaches a known variant location. In our experimentation, this strategy significantly improves the chance of finding proper seeds, therefore improves the chance of mapping reads to accurate locations *and* aligning reads accurately at those locations.

Algorithm 11 describes this iterated randomized algorithm. In this algorithm we aim at finding proper seeds based on long exact matches of the read r and the meta-genome G *with respect to* p.

Algorithm 11: FindSeeds(read r)

```
1:  p <− a random position on the read r.
2:  for n from 1 to A do
3:      repeat
4:          Start forward search in r from p using FM-index ℑ of meta-genome 𝒢.
5:      until seed length ≥ W OR search reaches the end |r|
6:      if seed found then
7:          return seeds
8:      else
9:          p <− a random position on the read r.
10: Return ∅.
```

In line 2, we set the maximal numbers of iterations to A. If A is too small, there will be many unaligned reads. If A is too large, the algorithm is slow. So, it is important to choose A appropriately. In line 5, we set minimum seed length to W. If W is too small, we might run into repeats, which lead to false positives. Small W also leads to more seeds and we have to perform more approximate matching for the next step (extending seeds), which is much more time-consuming. Therefore, the choice of W is very important. Our

strategy for determining good values of A and W is based on randomization, which is similar to the strategy we employed in RandAL (section 3.2.2).

Extension Phase

Many current methods have exploited dynamic programming-based alignment algorithms to efficiently extend seeds to a complete alignment. In particular, given a seed s, which is a substring of the reference and which matches exactly to a substring t of the read, we need to *extend* this exact match into a complete alignment by aligning the extracted left and right sides of s and t. To do that, algorithms such as Needleman-Wunsch or Smith-Waterman algorithms with some modifications and heuristics have usually been exploited [1]. These algorithms, however, can not be applied directly to our design, which needs to consider known variants in alignment process. Moreover, the representation of meta-genome requires differentiating V characters from other standard characters (A, C, G, T, or N) in the pairwise alignment. Therefore, a new alignment algorithm is needed to deal with theses problems.

Here we introduce an alignment algorithm which can take into account known variants in the pairwise alignment. This algorithm was implemented in procedures $LeftAlign$ and $RightAlign$ in line 8 of Algorithm 10. Our goal is computing alignment between the read $r[1 \cdots m]$ and the substrings of meta-genome $g[1 \cdots n]$, where $r[i] \in \{A, C, T, G, N\}$ and $\mathcal{G}[j] \in \{A, T, C, G, N, V\}$. To be precise, suppose that the read has the following form: xsy, and that its mate on the reference has the following form: usv, in which s is the seed found from the *seeding phase* (section 4.2.3). A complete alignment is obtained by aligning (1) x and u, and (2) y and v, and then combining them with the seed s. The alignment of x and u (and symmetrically between y and v) is done in two combination steps as follows:

1. **Non-gapped alignment**: starting at the end of x and u and backwardly, as long as the current character of x either matches the current character of u or (in case the current location of u is a known variant) matches one of the variants at the known

location of u. This process stops when two current characters do not match or the current location of u is a known INDEL. This step turns out can help to reduce running time of whole alignment process significantly. The main reason is, mutation rates are usually small, which means that mismatch locations, including INDELs, exist on only a small fraction of alignment locations. Therefore in most of cases, this step will be performed instead of pairwise alignment (the next step) which is much more time-consuming.

2. **Affine-gapped alignment**: the rest of x and u are aligned using an asymmetric edit-distance algorithm with an affine gap penalty scheme. This algorithm is different from the traditional edit-distance algorithm in that (1) the cost of deleting a prefix of u (on the reference) to align to an empty prefix of x (on the read) is 0 (asymmetric), and (2) the way of substituting, inserting and deleting bases depends on their locations on the reference, which can be known or unknown variant locations, and the corresponding cost is determined based on the variant profiles. By allowing the asymmetric alignment, we can align a short fragment x to a much longer fragment u, which is necessary to efficiently capture possible long reference-deletions, with an appropriate alignment cost. This is because the alignment cost should not depend on the length of the prefix of u which is aligned to an empty prefix of x. By using a variant location-dependent alignment process and variant profile-dependent cost scheme in an appropriate manner, we can efficiently exploit information of known variants to support determining the appropriate alignment.

The need for an asymmetric alignment between x and u is due to the high possibility that $u = u'x'$, where x' and x are near match and u' is a very long prefix of u. We called this *left-oriented* alignment. We also define the *right-oriented* alignment similarly to align asymmetrically y and $v = y'v'$, where y' and y are near match and v' is a very long suffix of v.

The following scheme describes our dynamic programming strategy for *left-oriented* alignment. In this strategy, we aim at computing edit distance with affine gap penalty between $s = s_1 s_2 \cdots s_m$ and $t = t_1 t_2 \cdots t_n$, in which s represents a read and t represents the matching meta-genome:

- Consider the right-most column of alignment between $s = s_1 s_2 \cdots s_i$ $(1 \leq i \leq m)$ and $t = t_1 t_2 \cdots t_j$ $(1 \leq j \leq n)$, we compute three traditional matrixes, D for an alignment (s_i, t_j), IS for an alignment $(s_i, -)$, and IT for an alignment $(-, t_j)$:

  $D(i,j)$: ... s_i $IS(i,j)$: ... s_i $IT(i,j)$: ... $-$

  ... t_j ... $-$ ... t_j

- If t_j is a standard character (A, C, G, T, N) then we use the traditional dynamic programming strategy for affine gap alignment.

- If t_j is a variant character V (a known variant location) then t_j can have several possibilities, which can be INDELs. The algorithm now needs to consider all possibilities of matching s to these possible variants. In the case of INDELs (more than one character), we need to look back on the s from s_i to take a equal-length sequence, i.e., the sequence $s_{i-k+1} s_{i-k+2} ... s_i$ (k is equal to length of INDEL), to match with the INDEL.

The Algorithm 12 describes our *left-oriented* edit distance algorithm. The *right-oriented* edit distance algorithm is similar. The alignment is asymmetrical due to the following initial configurations: (1) $IT_{0,j} = 0$, for $j \leq n$: the cost of converting t to an empty string is 0; (2) $IS_{i,0} = O$, for $j \leq n$, O is gap open penalty; (3) otherwise, the values were set to ∞. It is not possible to match reads to nothing from the reference genome. In our experimentation, the cost of substituting a read base s_i and reference base t_j is $-\log f(t_j)$, where $f(t_j)$ is the probability of t_j to be a variant. If t_j is a known variant, $f(t_j)$ is obtained from its probability in the variant profile, which is updated during the alignment of reads. If t_j is not a known variant, $f(t_j)$ is set to a user-defined

value, with default value of mismatches, gap opens and gap extensions are 4, 6 and 1, respectively, the same with default values of BWA-MEM.

Algorithm 12: $LeftAlign(r[1 \cdots m], g[1 \cdots n])$

$d \leftarrow 0$
while $m > 0$ and $n > 0$ and $g[n]$ has same-length possibilities **do**
  if n is NOT a variant position **then**
    $d \leftarrow d + cost(r[m], g[n])\}$
    $n \leftarrow n - 1$
    $m \leftarrow m - 1$
  else
    $d \leftarrow d + \min_{\forall g[n]: m-|g[n]| \geq 0}\{cost(r[m - |g[n]| + 1 \cdots m], g[n])\}$
    $n \leftarrow n - 1$
    $m \leftarrow m - |g[n]|$
$D[0][j] \leftarrow 0, 0 \leq j \leq n$
$D[i][0] \leftarrow \infty, 1 \leq i \leq m$
for $i = 1$ to m **do**
  for $j = 1$ to n **do**
    if $j - 1$ is NOT a variant position **then**
      if $r[i - 1] \neq g[j - 1]$ **then**
        $D[i][j] = D[i - 1][j - 1] + 1$
      else
        $D[i][j] = D[i - 1][j - 1]$
    else
      $D[i][j] = \min_{\forall g[j]: i-|g[j]| \geq 0}\{D[i - |g[j]|][j - 1] + cost(r[i - |g[j]| + 1 \cdots i], g[j])\}$
      $T[i - 1][j - 1] = \min_{g[j]}$
return $d + D[m][n], T$

This algorithm is exploited together with the aforementioned iterated randomized algorithm. After several iterations, if it found only one match with distinct minimum score, it will stop and considers that match as the correct candidate. Our experiment showed that, in many cases, (e.g, on the non-repeat regions of genomes) the number iterations are quite small. In some cases (e.g, on the repeat regions of genomes, which can result in multiple matches with similar score), the number iterations can reach the maximum number of allowed iterations. In this cases, we employed a filter strategy to determine the correct matches, which selects the correct candidate among the matches with lowest scores and proper insert sizes (for paired-end reads). Our method takes into

account variant information at the alignment phase, and consequently, aligned reads are more reliable, especially those alignments that overlap with known variant locations.

When the reads are declared to be aligned to the meta-genome, by tracing back from the computation of the edit distance, we can identify mismatches and insertions/deletions from the alignment. The trace back algorithm for *left-oriented* alignment is described in Algorithm 13. The trace back algorithm for *right-oriented* alignment is similar. These differences will then be used for detecting variants.

Algorithm 13: $LATraceBack(r[1 \cdots m], g[1 \cdots n], T)$

```
1:  V = ∅
2:  while i > 0 or j > 0 do
3:    if i > 0 and j > 0 then
4:      if j − 1 is NOT a variant position then
5:        if r[i − 1] ≠ g[j − 1] then
6:          V = V ∪ g[j − 1]
7:        i ← i − |g[n]|
8:        j ← j − 1
9:      else
10:       V = V ∪ T[i − 1][j − 1]
11:       i ← i − |T[i − 1][j − 1]|
12:       j ← j − 1
13:   else if i = 0 then
14:     j ← j − 1
15:   else if j = 0 then
16:     i ← i − 1
17: return V
```

4.2.4 Updating Variant Profiles and Calling Variants

The variant profile at a location stores all possible variants (SNPs or INDELs) and their probabilities at the location. Variant profiles of both known and unknown variants are kept track and updated using Bayes' theorem *during* the alignment of reads. After all reads are aligned and variant profiles are updated completely, the variant with maximum probability at each location is called as the predicted variant at the location. The main advantage of updating the variant profiles during the alignment process is, updated variant

profiles provide latest information to the alignment algorithm (section 4.2.3) *during* the read alignment process so the alignments are more likely correct.

To be specific, suppose a read base a_i is aligned to a location that is a known or unknown variant. Then, for each base b of the variant profile at the location, probability of b to be a true variant given that a_i is aligned to that location is calculated by:

$$P(\mathcal{T} = b|\mathcal{A} = a_i) = \frac{P(\mathcal{T} = b) \cdot P(\mathcal{A} = a_i|\mathcal{T} = b)}{P(\mathcal{A} = a_i)} \tag{4.1}$$

where $\mathcal{T}$ is a random variable representing the true variant and $\mathcal{A}$ is a random variable representing the aligned base at the location.

The *prior* probability $P(\mathcal{T} = b)$ is initially assigned based on the initial variant profile at the location. Initial profiles of known variants are assigned based on input variant databases (e.g., 1000 Genomes Project data), and the prior probability of variants is estimated from population allele frequency in the database. Initial profiles of unknown variants are assigned heavy bias toward the reference (more detail on this is given in the Supplementary material). This probability is then updated (replaced) by the *posterior* probability $P(\mathcal{T} = b|\mathcal{A} = a_i)$ *during* the alignment process.

To calculate the quantity $P(\mathcal{A} = a_i|\mathcal{T} = b)$, we consider two possible cases: (1) if $b = a_i$, then $P(\mathcal{A} = a_i|\mathcal{T} = b) = 1 - e_i$, where e_i is the probability that a_i is a sequencing error, which can be derived from base qualities of the read; (2) if $b \neq a_i$, then $P(\mathcal{A} = a_i|\mathcal{T} = b) = \frac{e_i}{3}$, which is probability of one of three non-a_i bases to be b. The other quantity, $P(\mathcal{A} = a_i)$, can be calculated as $P(\mathcal{A} = a_i) = \sum_b P(\mathcal{A} = a_i|\mathcal{T} = b)$. In case of an INDEL, the probability is a product of probabilities of corresponding reference and read bases.

After all reads have been aligned, the variant profiles are used to call variants at both known and unknown locations. For each variant profile location, let c be a base (or an INDEL) with highest probability, $f(c)$, among all of the other bases (or INDELs), then the

Phred quality score of c is given as $Q_c = -10\log_{10}(1 - f(c))$. Q_c is declared as quality score of the called variant c at the location.

Notice that the posterior probability given (the last) aligned base a_n using this dynamic strategy is equal to the posterior probability calculated for whole sequence of aligned bases $a = (a_1, a_2, ..., a_n)$. This can be proved by mathematical induction. First, we can easily check that the formula is correct with a_1. Then assume that the formula is correct with a_i. We can prove that the formula is correct with a_{i+1}.

INDEL Calling

In this section, we will give a more detail of our strategy for INDEL calling. Let assume that a_i is an INDEL, which can be denoted by $a_i = a_{i_1}a_{i_2}\cdots a_{i_m}$, $a_{i_k} \in \{A, C, G, T\}$, $k = 1...m$. At this point, to be simplistic, we assume that there is only one same INDEL aligned to the position, that is, a_i is identical to a_j, $\forall i, j$ if they are INDELs. Let $\mathcal{T}$ be the random variable representing true variant (can be SNPs or INDELs) at the position, and $\mathcal{A}$ be the random variable representing an aligned variant (can be SNPs or Indels as well) at the position. We now can calculate the posterior probability $P(\mathcal{T} = b|\mathcal{A} = a_i)$ for each base $b \in \{A, C, G, T, I\}$ (I represents the indel) given an aligned variant a_i with corresponding error rate e_i using formula 4.1. In the case of INDEL calling, the probability $P(\mathcal{A} = a_i|\mathcal{T} = b)$ given the aligned variant $a_i = a_{i_1}a_{i_2}...a_{i_m}$ with corresponding error rate $e_i = e_{i_1}e_{i_2}...e_{i_m}$, can be calculated as:

$$P(\mathcal{A} = a_i|\mathcal{T} = b) = \begin{cases} \prod_{k=1}^{m}(1 - e_{i_k}) & \text{if } a_i = b \\ \prod_{k=1}^{m}(e_{i_k}/3) & \text{if } a_i \neq b. \end{cases}$$

where,

- If a_i is INDEL ($a_i \in \{I\}$) and b is SNP ($b \in \{A, C, G, T\}$) then
  $P(\mathcal{A} = a_i|\mathcal{T} = b) = P(\mathcal{A} = a_{i_1}a_{i_2}...a_{i_m}|\mathcal{T} = b) = \prod_{k=1}^{m}(e_{i_k}/3)$
- If a_i is SNP ($a_i \in \{A, C, G, T\}$) and b is INDEL ($b \in \{I\}$) then
  $P(\mathcal{A} = a_i|\mathcal{T} = b) = P(\mathcal{A} = a_i|\mathcal{T} = b_1b_2...b_m) = e_{i_k}/3$

4.3 Results

4.3.1 Experimental Setup

Data

We evaluated IVC and the others on human chromosomes, which were available at the NCBI Genome Reference Consortium[1]. The known variant databases associated with the human chromosomes were available at the 1000 Genomes Project Consortium[2]. We performed evaluations primarily on GRCh37 Chromosome 1, which had approximately 250 millions bps. The associated variant profile 1000 Genomes Phase 1, which was collected from 1092 individuals, had $\sim$3 millions variant locations, of which $\sim$110 thousands ($\sim 3\%$) were INDELs and structural variants.

Since our method IVC was designed to leverage known variants to improve accuracy of detecting both known and *unknown* variants, we designed a set of simulated variants in which part of them are known and the others are unknown to IVC (and other tools as well). In our primary evaluation, 70% of all variant locations was randomly assigned to be known and the other 30% was assigned to be unknown to IVC. Comparisons were also done with percentage of known variant locations varied from 50% to 90%. These settings seems to be realistic with most of human data, of which a majority of variants were discovered.

To make the simulated data more "realistic", the variants were randomly selected from the databases based on their reported frequency. In our simulation, there were totally 211,855 variants, of which 17,761 INDELs. Among these INDELs, there were 8,379 Insertions and 9,382 Deletions, of which 201 Insertions and 398 Deletions had length $\geq 10bp$. We ignored Insertions and Deletions that are longer than read length. Paired-end reads were then generated using DWGSIM, a popular whole genome simulator[3], with

[1] http://www.ncbi.nlm.nih.gov/assembly/GCF_000001405.25

[2] http://ftp.1000genomes.ebi.ac.uk/vol1/ftp/phase1/analysis_results/integrated_call_sets

[3] https://github.com/nh13/DWGSIM

length 2x100bp and average insert size 500bp at coverage from 1x to 50x. The rate of sequencing errors was chosen to be 0.015% at the start and 0.15% at the end of reads to capture realistically base quality of high-quality real data. This error rate was increased monotonically from the beginning to the end of reads by DWGSIM. Additionally, ∼5% of random reads were inserted into the datasets by DWGSIM to reflect noise.

We also performed evaluation on a high quality Illumina HiSeq 2000 paired-end dataset, which was sequenced from sample NA12878 and provided by the Platinum Genomes project[4]. This dataset (ERR194147[5]) has read length 100 and coverage ∼50x. We used variant call set from 1000 Genomes data Phase 1, which does not include NA12878 individual, as known variant database for IVC. We assumed that part of variants of the individual NA12878 were already exist in the known database while the others were unknown. We expected that IVC could exploit the known variants from 1000 Genomes data to efficiently detect unknown variants for NA12878 using the ERR194147 dataset. We also used a more recent database for exome variants, called ExAC[6], as known variant database for IVC. This is a more robust variant database compared to 1000 Genomes data Phase 1, especially for INDELs, with much more number of individuals included in their data. We expected that IVC could take advantage of this database to further improve variant calling. To evaluate accuracy of all methods, we used Genome-In-A-Bottle (GIAB, [105]), a high quality variant calls set generated for the sample NA12878 using multiple datasets, technologies, and methods, as gold standard to evaluate variant callers. The evaluation for real data was also performed primarily on GRCh37 Chromosome 1.

A more detail description of the simulated and real data as well as their source links and citations was given in Supplementary materials.

---

4 http://www.illumina.com/platinumgenomes

5 http://www.ebi.ac.uk/ena/data/view/ERR194147

6 http://exac.broadinstitute.org

Evaluation of Accuracy

The accuracy of each calling method is defined in terms of *precision* and *recall* as follows:

$$Precision = \frac{TP}{TP + FP} \quad and \quad Recall = \frac{TP}{TP + FN}$$

where TP is the number of correctly called variants, FP is number of incorrectly called variants, and FN is number of true variants that are not called, by each method.

We compared IVC and other methods basically by considering the increase of accuracy (in terms of precision and recall) of IVC relative to other methods. The **percent of increase in precision (PIP) and recall (PIR)** of IVC relative to another method X is defined as $(\frac{p(IVC)}{p(X)} - 1) \cdot 100$, where p indicates either precision or recall. A positive (negative) PIP or PIR of a method indicate a better (worse) accuracy of that method, and the higher PIP or PIR, the better accuracy of the method (compared to another).

Variant Callers

We compared our method (IVC) to GATK UnifiedGenotyper (UG), HaplotypeCaller (HC) [106, 46], and SAMtools (ST) [44] for both SNP and INDEL calling. These tools have been reported to outperform others for variant calling [107, 108]. We also compared our method to Scalpel, a recent INDEL caller [109], which has previously shown to outperform many other state-of-the-art callers, to evaluate ability of IVC in calling INDELs. All four tools required an external short-read aligner, for which we chose BWA-MEM [110]. This tool has been applied widely among many popular aligners for read alignment and variant calling as well as suggested by all above methods [111, 4].

Additionally, we also used GATK IndelRealigner (IR), a realignment tool suggested by all tools UG, HC, and ST to improve their accuracy of Indel calling. Indel realignment is a post-alignment step, which realigns already-aligned reads around Indel locations. Table 7 showed the percent of increase in precision (PIP) and recall (PIR) of all these methods after using IndelRealigner. We found that this process could improve

Table 7: Percent of increase in precision (PIP) and recall (PIR) of all methods (excluding IVC) after using IndelRealigner

		SNP					INDEL				
		1x	5x	10x	20x	50x	1x	5x	10x	20x	50x
UG	PIP	0.00	0.14	0.23	0.34	0.45	10.95	1.00	0.29	0.48	0.46
	PIR	0.00	0.00	0.00	0.00	0.00	15.00	11.12	2.63	0.30	0.22
HC	PIP	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	-0.01	0.00
	PIR	0.00	0.00	0.00	0.00	0.00	0.18	0.03	0.01	0.00	0.01
ST	PIP	0.00	0.04	0.07	0.11	0.18	0.32	0.52	1.12	1.96	3.91
	PIR	0.00	0.00	0.00	-0.01	-0.01	3.53	1.38	0.30	0.19	0.06

considerably the accuracy of variants called by all tools UG, HC, and ST. In particular, for UG, precision and recall were increased by up to 11% and 22%, respectively. For ST, the increase was around 4% for precision and 6% for recall. Interestingly, while the improvement of precision was most dramatic with high-coverage data, the improvement of recall was most dramatic with low-coverage data. For HC, there was no improvement for SNP calling but some slight improvement of recall at low coverage for Indel calling. Based on the overall improvement in accuracy, we used IR to perform Indel realignment for each of the three tools in all comparisons with our own method.

Detail of used commands and parameter settings for these tools was given in Supplementary materials.

4.3.2 Overview of the Results

Our comprehensive comparisons on simulated data where IVC knew of 70% of variant information showed that IVC performed significantly better than the others (especially at low coverage) at detecting variants at Known locations, where it knew of variant information. This result will be described in section 4.3.3. At Unknown locations, where neither IVC nor the other methods knew of variant information, IVC also performed better than the others with overall higher (or at least competitive) recall and competitive precision in virtually all instances. This result will be described in section 4.3.3. In cases where Unknown locations are close to Known locations, IVC had significantly higher precision and recall at detecting unknown INDELs. This result will be

described in section 4.3.3. Varying IVC's knowledge of variant information from 50% to 90% showed that as IVC gained more knowledge it also performed better at calling variants, especially at Unknown locations. These results will be described in section 4.3.3.

For real data evaluation, IVC performed better than GATK-HC and SAMtools in many cases, especially calling known INDELs. This result will be described in section 4.3.4 IVC also showed a better accuracy on calling known INDELs than Scalpel, an INDEL caller that was mainly designed to call INDELs in exonic regions. This result will be described in section 4.3.4.

In terms of runtime and memory usage, our current implementation was better at low to medium coverage and competitive at high coverage. This result will be described in section 4.3.5.

4.3.3 Accuracy Comparison on Simulated Data

IVC has Superior Accuracy in Known Locations

As expected, IVC had superior accuracy of detecting variants at the Known locations. The left part of Table 8 showed the percent increase in precision (PIP) and recall (PIR) of IVC relative to the three variant calling methods at Known variant locations (values that are greater or equal to 10 are rounded to integers). IVC had superior accuracy compared to other tools at INDEL calling in the Known region. IVC were superior at both precision and recall. In terms of precision, there was between 1% and 2% improvement. In terms of recall, the improvement was drastic. Among the three other tools, HC was the best in terms of recall. Even so, IVC was drastically superior to HC in recall. At 50x, the improvement in recall was about 10%. At lower coverage, the improvement in recall was more than 10% at 10x, and 189% at 1x. The improvement of accuracy of calling SNPs in the Known region was not as drastic. Compared to UG, IVC's precision was very slightly worse, but its recall was much better, particular at lower coverage. Compared to HC and ST, IVC was better at both precision and recall, especially at low coverage. The advantage of exploiting known information of variants was shown

by the impressive accuracy of IVC compared to the others' at low coverage data (tens percent of increase at 5x and even hundreds percent of increase at 1x) and less impressive at higher coverage (20x-50x). Other methods did not work well at low coverage data but they did better at high coverage data.

Table 8: Percent of increase in precision (PIP) and recall (PIR) of IVC relative to the other methods (UG: GATK UnifiedGenotyper, HC: GATK HaplotypeCaller, ST: SAMtools) for coverage from 1x to 50x, in terms of SNP and INDEL calling.

	Known variant locations					Unknown variant locations				
	1x	5x	10x	20x	50x	1x	5x	10x	20x	50x
	SNP									
PIP UG	-0.08	-0.02	0.00	-0.01	0.01	0.37	0.58	0.36	0.36	0.43
PIR UG	126	3.10	0.17	0.10	0.08	0.04	0.01	-0.24	-0.04	-0.04
PIP HC	-0.03	0.03	0.02	0.02	0.03	0.49	0.42	0.20	0.08	0.09
PIR HC	140	4.09	0.34	0.21	0.19	6.29	0.97	-0.08	0.07	0.08
PIP ST	-0.03	0.26	0.53	0.25	0.01	0.07	0.07	0.41	0.38	0.16
PIR ST	147	4.69	0.92	0.52	0.23	9.41	1.61	0.46	0.35	0.11
	INDEL									
PIP UG	58	2.27	1.97	2.00	1.97	81	-1.14	0.62	0.86	1.19
PIR UG	23977	125	18	11	10	11039	90	8.28	4.75	5.34
PIP HC	1.03	1.72	1.78	1.74	1.67	2.51	0.31	0.96	0.74	0.74
PIR HC	189	15	9.83	9.52	9.53	19	2.00	1.54	3.66	4.22
PIP ST	2.70	2.18	2.05	2.01	2.24	5.01	2.08	3.49	5.72	13
PIR ST	157	34	17	12	12	3.06	16	7.38	5.53	5.81

IVC has Higher Accuracy in Unknown Locations

We found that at Unknown locations, where IVC had no prior knowledge of those variants, IVC was also more accurate than the other methods at detecting those variants. This means IVC was more accurate and often superior to all three methods at detecting new variants. The superiority was most dramatic at lower coverage and at detecting new INDELs. The right part of Table 8 showed the percent increase in precision and recall of IVC relative to the three other methods at Unknown locations.

In term of calling INDELs, IVC was particularly more accurate than the other methods at both precision and recall. At higher coverage, the increase in recall was between 3 and 5%. At lower coverage, the percent increase in recall was between 2 and

20% compared to HC and ST, and up to several hundreds percent compared to UG. In terms of precision, IVC, by and large, was also better that the other three methods. In some instances, IVC was slightly worse at precision, but was much better at recall (e.g. compared to UG at 5x, there was a decrease of $\sim$1%, but increase of $\sim$90% at recall).

In term of calling SNPs, IVC was also better than the other three methods. Compared to ST, IVC was better at both precision and recall. Compared to UG and HC, IVC was not universally better at both precision and recall at all coverages, however, at places where IVC lacked a little (less than 1%) in precision, it gained more in recall, and vice versa. The advantage of exploiting known information of variants to detect unknown variants was shown by the impressive accuracy of IVC compared to the others' at low coverage data and less impressive at higher coverage, similar to what were observed at Known locations. Again, other methods did not work well at low coverage data but they did better at high coverage data.

Raw precision and recall at different coverages is shown in Figure 27. IVC had superior recall and better/competitive precision across all coverages from 5x to 50x, especially at detecting INDELs. IVC was the only tool with performance at both precision and recall are always greater than 0.9 at 10x or higher coverage.

IVC's Accuracy at Varied Percent of Known Locations

We have compared performance of different methods with the assumption that IVC knew of 70% of variant information. In fact, similarly high accuracy of IVC was also observed when its knowledge of known variants varied from 50% to 90%. This finding was shown in Table 9. The table showed the percent increase in precision and recall of IVC relative to HC, the best tool among the others, when the coverage is fixed at 50x. In terms of calling SNPs, IVC's performance was a little better or similar to HC. In terms of calling INDELs, however, IVC's performance, especially at recall, was noticeably better than HC in all levels from 50% to 90%. Furthermore, as IVC's knowledge increased, its performance appeared to improve, especially at recall.

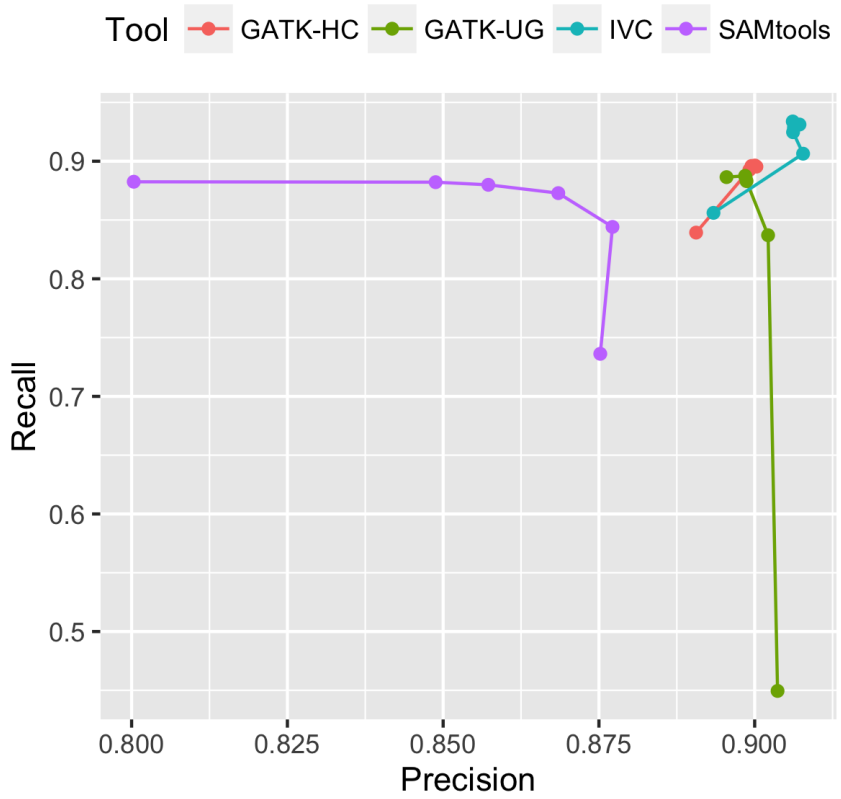

Figure 27: Precision versus Recall for INDEL calling at Unknown variant locations as coverage varies from 5x to 50x.

IVC has Higher Accuracy in Calling close-by INDELs

Two INDEL are considered to be *close-by* if their left-most ends are located closely to each other. Our analysis of INDELs in human chromosome 1 showed that such close-by INDELs are not rare. In fact, there are 14% of INDELs (15,356 in total of 110,233 INDELs) have at least one INDEL located within $30bp$ on the left or on the right. The number of such INDELs that have 2, 3, 4, 5, 6 close-by INDELs are 2268, 500, 110, 68, 14, respectively (there are no INDELs that have more than 6 INDELs located within $30bp$ on the left or on the right).

Close-by INDELs make it challenging for aligners to get the correct alignments and consequently for variant callers to make the correct calls [112]. An aligner's mistake in getting the correct alignment for one read will likely repeats for another read.

Table 9: Percentage of increase in precision (PIP) and recall (PIR) of IVC relative to HC at coverage 50x as IVC's knowledge of known variants increases from 50% to 90%.

	Known variant locations					Unknown variant locations				
	50%	60%	70%	80%	90%	50%	60%	70%	80%	90%
	SNP									
PIP HC	0.01	0.02	0.03	0.03	0.03	0.00	0.04	0.09	0.24	0.649
PIR HC	0.20	0.19	0.19	0.20	0.18	0.06	0.07	0.08	0.04	0.08
	INDEL									
PIP HC	0.87	1.35	1.67	1.67	2.28	-0.24	0.35	0.74	1.97	2.13
PIR HC	9.34	9.64	9.53	9.54	9.70	2.65	3.58	4.22	4.55	4.81

Therefore, high coverage probably does not help in improving the accuracy of calling close-by INDELs. To illustrate this, let's consider a toy example in which the read GTAATATTGT is aligned to the reference genome GTATTAGTGT:

```
location  12   34567890   1234567890
genome    GT---ATTAGTGT   GTATTAGTGT
read      GTAATATT-GT     GTAATATTGT
          correct algn    incorrect algn
```

The correct alignment yields two INDELs at location 2 and 5, while the incorrect alignment yields two SNPs at location 4 and 7. An aligner likely chooses the incorrect alignment as it has better scores (gaps in INDELs are generally penalized seriously by aligners). This mistake can be repeated for all reads aligned to this region therefore it makes the alignment based variant callers difficult to detect the correct variants even at high coverage. However, if we know *a priori* one of the two INDELs, we might be able to determine the correct alignment.

Our analysis of true positives and false negatives of all four methods confirmed this. All four methods were more likely to detect variants correctly when there were fewer close-by INDELs. Counting the number of INDELs located within $30bp$ on the left or on the right of a variant at coverage 50x, we found that on average, the difference in numbers of close-by INDELs between true positives and false negatives for all four methods is

approximately 1. Thus, on average, one close-by INDEL within a window size of 30 could make a difference between a correct call (true positive) and a wrong call (false negative).

IVC's ability to detect close-by INDELs compared to the other methods is apparent in **Table 8**. Even at high coverage 50x, IVC's recall rate of INDELs was 4-5% higher than the that of the other methods.

To analyze IVC's ability to detect these INDELs more carefully, we separate INDELs into three groups based on the number of known close-by INDELs within $30bp$: 0.50, 0.67 and 1.00. For example, Group 0.50 consists of all INDELs for which IVC is informed of 50% of close-by INDELs. Similarly, Group 1.00 consists of all INDELs for which IVC is informed of 100% of close-by INDELs. Figure 28 shows the numbers of true positives and false negatives of INDELs in each of the 3 groups, for all 4 methods. First, we see that the ratio between true positives and false negatives of GATK-UG, GATK-HC, and SAMtools are, respectively, similar in all 3 groups. This makes sense because these tools are not informed of close-by INDELs. Second, the more IVC is informed of close-by INDELs, the more likely it is able to detect INDELs correctly. In particular, when IVC is totally informed of close-by INDELs (Group 1.00), the number of true positives and false negatives increased and decreased significantly, respectively, compared to the others.

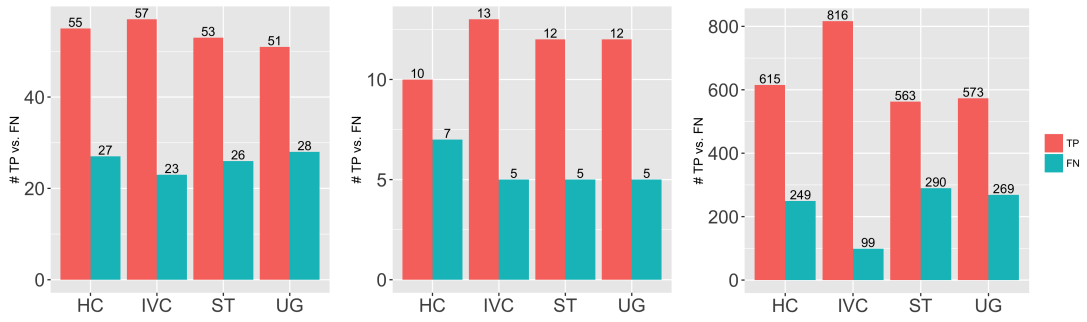

Figure 28: TP and FN at different fractions (left: 0.50, center: 0.67, right: 1.00) of known INDELs that are close to an INDEL.

4.3.4 Accuracy Comparison on Real Data

IVC' Performance on Platinum Genomes Data

To compare the ability to detect variants on real data of IVC, GATK HaplotypeCaller (HC) and SAMtools (ST), we chose a high-quality short-read dataset ERR194147 (of human individual NA12878), provided by Illumina Platinum Genomes. We used variants from the 1000 Genomes Phase 1 data as the known variant database for IVC. Since the reference variant data do not include the NA12878 individual, the experiment is meant to show how IVC exploits known variant information from population to detect variants for a new individual. In this experiment, GATK UnifiedGenotyper was eliminated because of its inferior performance on simulated data.

To evaluate accuracy of all methods, we used Genome-In-A-Bottle variant call set (GIAB, [105]), which was generated for the individual NA12878. This high-quality variant call set has been used as gold standard for benchmarking SNP and INDEL genotype calls in many studies [113, 55]. GIAB were obtained by integrating 14 datasets from five sequencing technologies, seven read mappers and three variant callers to help minimize bias toward any method.

Variant calls of all methods (including IVC) are divided into two main groups: (1) GIAB-Call: called variants that are exist in the GIAB variant call sets. They are likely true positives, and (2) GP-Call: called variants that are not exist in the GIAB dataset, but, however, are in the 1000 Genomes Phase 1 data. They are potentially true positives since they do exist in the population of thousands of individuals. We also consider called variants that are neither in the GIAB dataset nor in the 1000 Genomes database. Although they are potentially false positives, some of them might be new discoveries.

The two top diagrams of Figure 29 showed the number of called variants (SNPs and INDELs, separately) in GIAB-Call identified by each method. We can see that IVC shared with both HC and ST most of variant calls in GIAB-Call group. Moreover, IVC called more unique variants than all other methods. This result showed the practical

effectiveness of IVC compared to other state of the art methods. Further, the number of variants shared between IVC and HC is the highest compared to those between IVC and ST as well as those between HC and ST. This was consistent with the high performance of IVC and HC compared to ST with simulated data.

The most important result, which confirmed the advantage of exploiting known variants (here is variants from 1000 Genomes data) by IVC to detect variants in GP-Call group, was shown in the two bottom diagrams of Figure 29. In this group we saw that IVC called much more unique variants, especially INDELs, than the other methods. We got $\sim$9 times more unique SNPs and $\sim$20 times more unique INDELs than the others'. Because the 1000 Genomes data is collected from many people, we think that those variant calls are worth to consider for downstream analysis. Interestingly, while IVC shared the most number of SNPs with HC, it shared the most number of INDELs with ST, compared to the number of sharing variants between all two methods.

For the groups of called variants that are neither in the GIAB dataset nor in the 1000 Genomes database, we found that the agreement (disagreement) between IVC, HC, and ST were the lowest (highest) compared to two above groups ($\sim$3100 SNPs and $\sim$3300 INDELs). In particular, for INDEL calling, each method also called $\sim$4000-4500 unique variants, which were even more than the common calls. It is quite reasonable since those variants are potentially false positives, though some of them could be new discoveries.

IVC' Performance with ExAC Database as Known Variants

In this experiment we used ExAC database as the known variant source for IVC. Because ExAC is an exome database, this experiment was done on the exonic regions only, but we think it could still show how well IVC can perform with a more robust known variants. We also randomly selected reads from the original dataset ERR194147 (50x) to make sub datasets with 5x, 10x, and 20x, respectively, to investigate impact of coverage on performance of each method.

Table 10 showed that IVC called more INDELs with ExAC compared to 1000

Genomes data in both categories GIAB-Call (GIAB-INDEL) and ExAC-Call (ExAC-INDEL), especially at low coverage. It also called more SNPs in ExAC-Call category (ExAC-SNP). This result showed that IVC can take advantage of the ExAC in calling variants, especially INDELs. Interestingly, while ExAC helped call more SNPs than GP at low coverage (5x), the GP helped call more SNPs than ExAC at higher coverage (GIAB-SNP). We did an analysis of the SNPs shared by GIAB and ExAC or GP, respectively, and found that the number of SNPs in the former was larger than that of the latter. It is probably the reason why IVC could call more SNPs with support from GP than ExAC. Nevertheless, ExAC can help call more SNPs in low coverage, which confirmed the advantage of using a robust known variant source for calling variants at low coverage.

Table 10: Number of exonic variants called by IVC from dataset ERR194147 for sample NA12878 with ExAC and GP as input.

		5x	10x	20x	50x
GIAB-SNP	ExAC	1012	1101	1146	1169
	GP	660	1286	1359	1358
ExAC-SNP	ExAC	1979	2246	2285	2436
	GP	678	1708	2188	2051
GIAB-INDEL	ExAC	69	73	74	75
	GP	29	66	72	73
ExAC-INDEL	ExAC	1089	1260	1322	1381
	GP	33	88	113	101

To further investigate ability of IVC in calling INDELs, we benchmarked IVC against Scalpel, a more recent INDEL caller [109]. Scalpel performs a localized micro-assembly of specific regions of interest to improve detecting of mutations. Although Scalpel can run with WGS data, we evaluated Scalpel, GATK-HC, SAMtools, and IVC in exonic regions to make a fair comparison with Scalpel, which has been primary tested with exome-capture data.

Table 11 showed that Scalpel did a good job for calling INDELs in exonic regions. It called more number of total GIAB INDELs than other tools including IVC, especially at high coverage ($>$10x). However, for calling GIAB$\cap$ExAC INDELs (GIAB INDELs that

are in ExAC), IVC was better than Scalpel (and other tools), especially at low coverage (5x). This result showed that IVC can take advantage of ExAC, especially at low coverage. Moreover, IVC was able to call much more number of ExAC INDELs than other tools, including Scalpel. Because the ExAC database is quite a reliable source of variants, we think that those INDELs are worth to consider for downstream analysis.

Table 11: Number of exonic INDELs called by each method from dataset ERR194147 for sample NA12878.

		5x	10x	20x	50x
GIAB∩ExAC	SC	15	51	59	59
	HC	57	64	64	64
	ST	52	64	63	63
	IVC	65	66	69	70
GIAB∩NonExAC	SC	10	44	62	71
	HC	12	13	14	14
	ST	7	8	12	12
	IVC	4	7	5	5
ExAC∩NonGAIB	SC	30	70	59	52
	HC	64	61	55	58
	ST	44	55	57	61
	IVC	1089	1260	1322	1381

SC: Scalpel, HC: GATK HaplotypeCaller, ST: SAMtools, IVC: our method. GIAB∩ExAC: variant calls which exist in both GIAB and ExAC. GIAB∩Non-ExAC: variant calls which exist in GIAB but not in ExAC. ExAC∩Non-GIAB: variant calls which exist in ExAC but not in GIAB.

4.3.5 Comparison of Running Time and Memory Usage

IVC integrates short-read alignment and variant calling. IVC's workflow consists of 2 steps: building an index and calling variants. The other methods requires at least 6 separate steps with the assistance of external tools for indexing, aligning reads, sorting the SAM/BAM files, doing some post-alignment tasks, performing realignment, and calling variants. This is illustrated in (Figure 30). With the same input (reference genome in FASTA format, reads in FASTQ format, known variant databases in VCF format) and

output (variant calls in VCF format), IVC's workflow is simpler, requires less human intervention, and therefore decreases human errors and manipulation time.

We compared running time and memory usage of IVC and other tools with coverage from 5x to 50x. For the other tools we summed up runtime and memory of all steps in whole variant calling process in a proper way. All experiments were run on a workstation with 2 Xeon E5-2680 2.70GHz CPUs using multi-threaded mode of all tools whenever possible with maximum number of cores (32) (actually most of them could be run in multi-threaded mode). In this experiment, we ignored human-time for manipulating on input/ouput for intermediate steps of other tools. Figure 31 showed that IVC was faster than the others at low to medium coverage and was quite competitive at 50x.

4.4 Conclusion

By leveraging known genomic variants, IVC could significantly improve accuracy of detecting not only known but also *unknown* variants, including close-by INDELs, one source of hard-to-detect INDELs. Compared to well-known methods such as GATK UnifiedGenotyper, GATK HaplotypeCaller, and SAMtools, IVC had superior accuracy for calling *unknown* variants, especially INDELs that are hard to detect even at high coverage. Its high accuracy at low coverage can help researchers design less expensive experiments.

Another advantage of IVC is in its integrated and simplified workflow, which eliminates some intermediate steps and consequently reduces human intervention and errors. Using pipeline of several tools as conventional methods can increase flexibility, therefore one of our future works is providing users not only variant calls but alignment results. The alignment results then can be analyzed by other variant calling methods.

Although the data used in our simulation study do not contain long INDELs and other structural variants, our method were designed not to strictly limit itself from those types of variants. In particular, as long as information of long INDELs exists in databases, IVC can efficiently identify them from the data. Detection of long INDELs at unknown variant locations are also considered in the next version of IVC.

One important motivation of our work is the number of known genetic variants is rapidly growing. However, variant databases such as the 1000 Genomes Project data are not error-free despite best efforts. Since our method prefers the know variants if they exist by giving them high prior probabilities in variant profiles, the incorrect known variants can result in false positives. However, our method do not rely on only known variants, it considers the alignment between reads and the reference as well. Those incorrect variants will likely result in the alignments with ambiguities or high cost, which are likely to be eliminated during the alignment process. Consequently, those variants likely do not have high probabilities in the final variant profiles and they are likely not called. This makes our method less suffers from incorrect known variants.

Currently, our method considers data as haploid, though the theoretical framework is similar with other ploidies. The difference is in the content of variant profiles and the way of updating them. In particular, the variant profile which stores possibilities of haploid variants needs to change to store possibilities of variants with other ploidies. The variant profile updating strategy also needs to change to adapt to those kind of variants.

One further experiment we could do for other methods is matching their variant calls to the variant databases. This might reduced their false positives of variant calls, however, false negatives were still their problems. As shown in our comparison study, other methods did a good job on precisely detecting variants. Actually, their precisions were competitive with us (and with each other) in many instances. However, their recall rate was from lower to much lower than that of our method, i.e., they got more false negatives, which could not be recovered by matching the calls to the databases.

In summary, our method is promising in accurately detecting genetic variants from NGS data, including close-by INDELs, which are difficult to be detected by other methods due to low coverage or the hardness of detecting those variants. The current implementation is not optimized but still competitive with the others in both running time and memory usage.

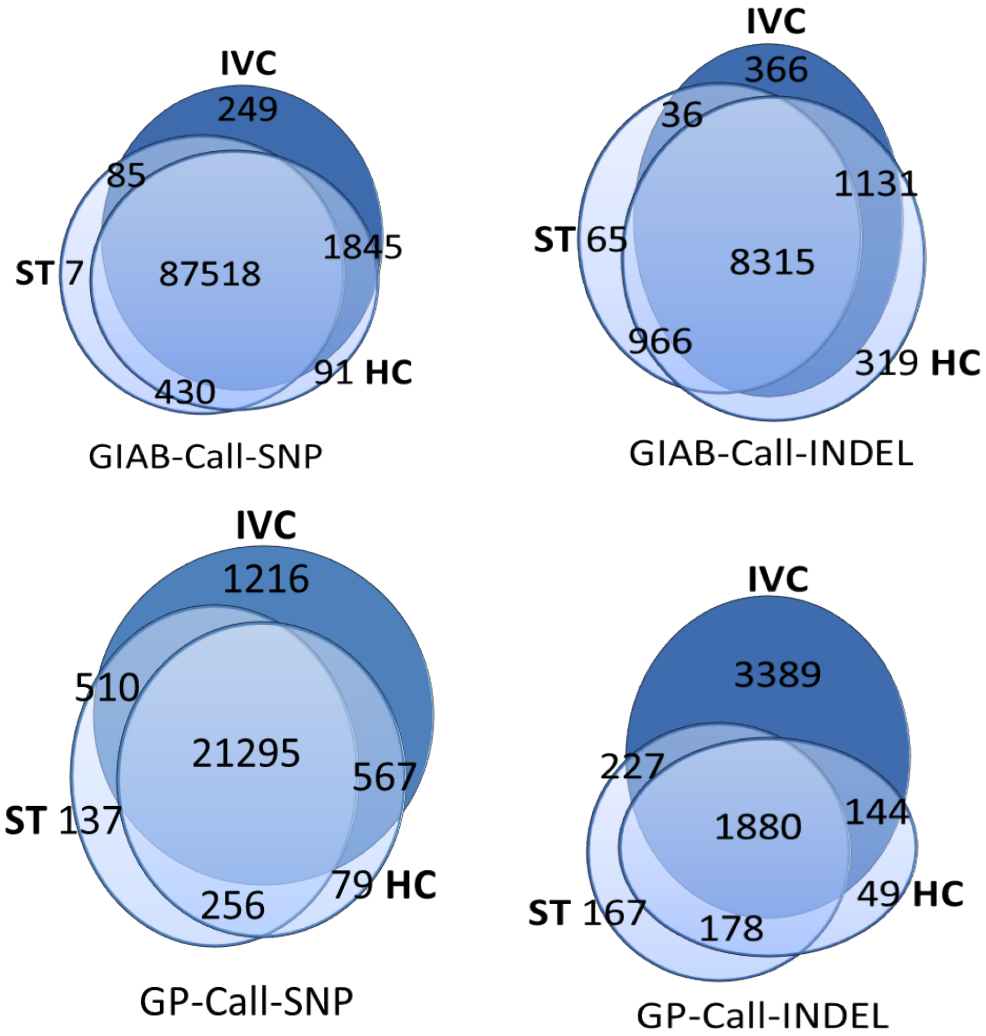

Figure 29: Venn diagram of SNPs and INDELs called by each method from dataset ERR194147 on sample NA12878. GIAB-Call: called variants that are in the GIAB dataset. GP-Call: called variants that are not in the GIAB dataset but are in the 1000 Genomes Phase 1 data.

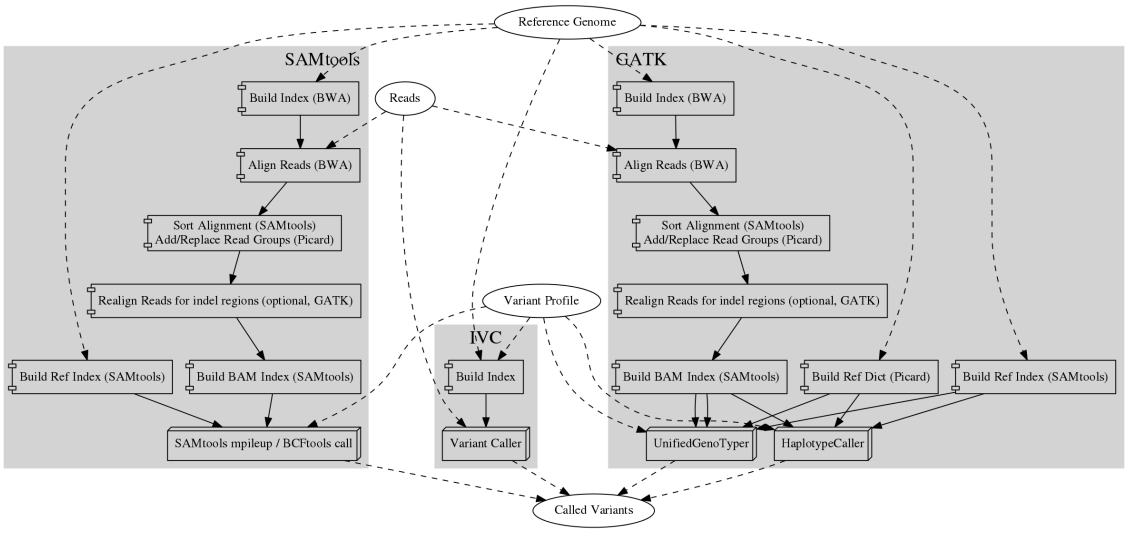

Figure 30: Workflow of IVC and GATK.

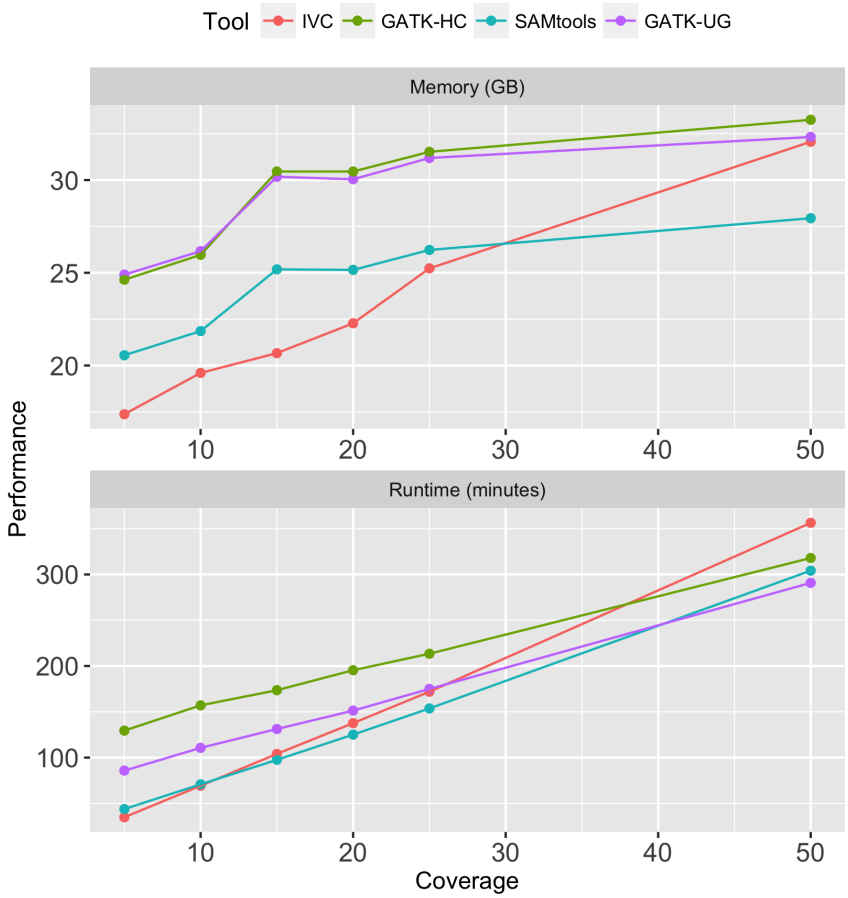

Figure 31: Runtime and memory usage of all tools as coverage varies from 5x to 50x.

Chapter 5

Conclusions

Presently, with a massive amount of biological data generated from advanced technologies, the need for developing efficient computational methods to analyze such data become very urgent. In this dissertation, we focused on developing computational methods to address several important issues in gene expression and genomic sequence analysis, two of the most important areas of bioinformatics. Among many problems in these areas, we selected three important problems to work on: (1) representing and analyzing patterns of gene response to multiple treatments (*gene pattern analysis*), (2) aligning short DNA sequences to reference genomes (*read alignment*), and (3) detecting genomic variants (*variant calling*). We made several important contributions to these problems, both in proposing novel approaches and methods as well as in developing tools for practical usage. Furthermore, our approaches opened several promising future directions for developing methods and tools to solve remaining problems in these areas. We will describe these contributions and directions in the following.

5.1 Key Contributions

For the *gene pattern analysis* problem, we introduced two novel methods to predict true patterns of gene response to multiple treatments in case of small sample size. These methods employed directed graphs and orderable sets, respectively, to represent gene patterns and exploit their certain properties using proper ways to deduce more and better functionally enriched gene clusters. *In the first method*, we employed directed graphs as representation of gene expression patterns. We then showed that true patterns of gene expression can be predicted by a pre-cautious use of synthetic data. We exploited the relationship between sample size and a graph property known as *contractibility* together with synthetic replicates to predict true patterns of gene response to treatments. Further, we showed how to use entropy to measure the certainty of such predictions. Directed graphs can benefit subsequent functional analysis as well. *In the second method*, we

showed that true patterns of response must be *orderable sets*. This property enables the exploitation of dependencies among outcomes of statistical tests that constitute patterns of gene response. Specifically, instead of obeying the convention of fixing the level of significance α at 0.05, we showed that α might be varied (in a different way for each gene) to predict true patterns more accurately. Moreover, motivated by the fact that a gene might participate in multiple biological functions or be involved in multiple molecular processes, we showed how to manipulate experimental replicates to predict and assign multiple patterns to each gene. These methods are especially useful for gene-expression studies designed to explore functional similarities and differences of similar chemicals.

For the *read alignment* problem, we introduced a novel method to improve accuracy of short-read alignment across a wide range of read lengths and base error rates. This method exploited an iterated randomized algorithm together with two FM indexes to improve alignment performance. In this method, we employed two FM indices for efficient bidirectional (exact) substring matching between reads and reference genomes. To deal with inexact matching (i.e. allowing gaps), first, we searched for common substrings, so-called *seeds* between reads and the reference. Then, these seeds were extended to complete alignments based on a bounded edit distance algorithm. We used an iterated randomized algorithm for aligning reads to the reference. This increases the probability of finding seeds quickly and enables us determine methodologically important parameters to speed up the entire alignment process. We also used a special pruning mechanism to shorten vastly the running time of computing edit distances in a vast majority of cases. Our method is useful for experiments that require consistent performance across a wide range of read lengths and sequencing error rates.

For the *variant calling* problem, we introduced a novel method to improve accuracy of calling genomic variants, especially insertions/deletions. This method leveraged existing genomic variants, such as those provided by the 1000 Genomes Project, during read alignment and variant calling process to improve variant calling performance.

In this method, we designed a special reference genome called multi-genome, which combines a given variant profile and a reference genome into a unified representation, to facilitate using existing variants. We then design a mixture technique, which combines hashing, modified FM index, and an asymmetric local alignment, to accurately align reads to the reference multi-genome and to call variants simultaneously. Our approach is useful for experiments with low coverage data. It also help reduce computational cost of calling variants by reducing running time and human intervention.

5.2 Future Directions

For the gene pattern analysis problem, we plan to make our current method working with gene expression data that are taken with RNA-Seq technologies due to the emerge of this type of data. The theoretical framework for RNA-Seq data is expected to be similar to current one. We also plan to apply and validate this method with several types of datasets, especially datasets with many treatments, even hundreds of treatments.

For the read alignment problem, we plan to further improve accuracy of our current method and apply it to diverse datasets from diverse species. The most important future work is adapting our method to the alignment of reads from RNA-Seq experiments, especially for discovering splice junctions.

For the variant calling problem, we plan to further improve our current method, in both accuracy and running time. We also plan to adapt our method to other ploidies. The theoretical framework for other ploidies is expected to be similar to current one. The most important future work is making our method working with other types of structural variants: long insertions/deletions, inversions, translocations, and copy number variants. In term of application, we plan to apply our method to several types of data sources, especially cancer data, which include a rich source of variants.

Bibliography

[1] H. Li and N. Homer, "A survey of sequence alignment algorithms for next-generation sequencing," *Briefings in Bioinformatics*, vol. 11, no. 5, pp. 473–483, 2010.

[2] R. Nielsen, J. S. Paul, A. Albrechtsen, and Y. S. Song, "Genotype and snp calling from next-generation sequencing data," *Nature Reviews Genetics*, vol. 12, no. 6, pp. 443–451, 2011.

[3] T. J. Treangen and S. L. Salzberg, "Repetitive dna and next-generation sequencing: computational challenges and solutions," *Nature Reviews Genetics*, vol. 13, no. 1, pp. 36–46, 2012.

[4] S. Pabinger, A. Dander, M. Fischer, R. Snajder, M. Sperk, M. Efremova, B. Krabichler, M. R. Speicher, J. Zschocke, and Z. Trajanoski, "A survey of tools for variant analysis of next-generation genome sequencing data," *Briefings in Bioinformatics*, vol. 15, no. 2, pp. 256–278, 2014.

[5] M. B. Eisen, P. T. Spellman, P. O. Brown, and D. Botstein, "Cluster analysis and display of genome-wide expression patterns," *Prot. Natl. Acad. Sci.*, vol. 95, pp. 14 863–14 868, 1998.

[6] M. L. Lee, F. C. Kuo, G. A. Whitmore, and J. Sklar, "Importance of replication in microarray gene expression studies: statistical methods and evidence from repetitive cdna hybridization," *Prot. Natl. Acad. Sci.*, vol. 97, 2000.

[7] M. K. Kerr, M. Martin, and G. A. Churchill, "Analysis of variance for gene expression microarray data," *Journal of Computational Biology*, vol. 7, pp. 819–837, 2000.

[8] Z. Wang, M. Gerstein, and M. Snyder, "Rna-seq: a revolutionary tool for transcriptomics," *Nature Reviews Genetics*, vol. 10, no. 1, pp. 57–63, 2009.

[9] A. Oshlack, M. Robinson, and M. Young, "From rna-seq reads to differential expression results," *Genome Biology*, vol. 11, no. 12, p. 220, 2010.

[10] F. Ozsolak and P. M. Milos, "Rna sequencing: advances, challenges and opportunities," *Nature reviews genetics*, vol. 12, no. 2, pp. 87–98, 2011.

[11] D. K. Slonim, "From patterns to pathways: gene expression data analysis comes of age," *Nature Genetics*, vol. 32, pp. 502–508, 2002.

[12] Z. Wang, M. Gerstein, and M. Snyder, "Rna-seq: a revolutionary tool for transcriptomics," *Nature Reviews Genetics*, vol. 10, no. 1, pp. 57–63, 2009.

[13] J. C. Venter, M. D. Adams, G. G. Sutton, A. R. Kerlavage, *et al.*, "Shotgun sequencing of the human genome," *Science*, vol. 280, no. 5369, p. 1540, 1998.

[14] M. L. Metzker, "Sequencing technologies—the next generation," *Nature Reviews Genetics*, vol. 11, no. 1, pp. 31–46, 2010.

[15] E. R. Mardis, "Next-generation dna sequencing methods," *Annual Review of Genomics and Human Genetics*, vol. 9, pp. 387–402, 2008.

[16] M. R. Fielden, R. Brennan, and J. Gollub, "A gene expression biomarker provides early prediction and mechanistic assessment of hepatic tumor induction by nongenotoxic chemicals," *Toxicol. Sci*, vol. 99, no. 1, pp. 90–100, September 2007.

[17] G. Natsoulis, C. Pearson, J. Gollub, P. Eynon, J. Ferng, R. Nair, R. Idury, M. Lee, M. Fielden, R. Brennan, A. Roter, and K. Jarnagin, "The liver pharmacological and xenobiotic gene response repertoire," *BMC Syst Biol*, vol. 4, p. 175, 2008.

[18] T. Sutter, X. He, P. Dimitrov, L. Xu, G. Narasimhan, E. O. George, C. H. Sutter, C. Grubbs, R. Savory, M. Stephan-Gueldner, D. Kreder, M. J. Taylor, R. Lubet, T. A. Patterson, and T. W. Kensler, "Multiple comparisons model-based clustering and ternary pattern tree numerical display of gene response to treatment: procedure and application to the preclinical evaluation of chemopreventive agents," *Mol Cancer Ther.*, vol. 1, pp. 1283–92, 2002.

[19] R. Hulshizer and E. M. Blalock, "Post hoc pattern matching: assigning significance to statistically defined expression patterns in single channel microarray data," *BMC Bioinformatics*, vol. 8, p. 240, 2007.

[20] V. Phan, E. George, Q. Tran, S. Goodwin, S. Bodreddigari, and T. Sutter, "Analyzing microarray data with transitive directed acyclic graphs," *Journal of Bioinformatics and Computational Biology*, vol. 7, no. 1, pp. 135–156, 2009.

[21] Q. Tran, L. Xu, V. Phan, S. Goodwin, M. Rahman, V. Jin, C. Sutter, B. Roebuck, T. Kensler, E. George, and T. Sutter, "Chemical genomics of cancer chemopreventive dithiolethiones," *Carcinogenesis*, vol. 30, no. 3, pp. 480–486, 2009.

[22] W. Pan, J. Lin, and C. Le, "How many replicates of arrays are required to detect gene expression changes in microarray experiments? a mixture model approach," *Genome Biol.*, vol. 3, p. research0022, 2002.

[23] I. Jeffery, D. Higgins, and A. Culhane, "Comparison and evaluation of methods for generating differentially expressed gene lists from microarray data," *BMC Bioinformatics*, vol. 7, no. 1, p. 359, 2006.

[24] C. Sima and E. Dougherty, "What should be expected from feature selection in small-sample settings," *Bioinformatics*, vol. 22, no. 19, pp. 2430–2436, 2006.

[25] L. Klebanov and A. Yakovlev, "Is there an alternative to increasing the sample size in microarray studies?" *Bioinformation*, vol. 1, no. 10, pp. 429–431, 2007.

[26] H. Yang and G. Churchill, "Estimating p-values in small microarray experiments." *Bioinformatics*, vol. 23, no. 1, pp. 38–43, 2007.

[27] P. Glaus, A. Honkela, and M. Rattray, "Identifying differentially expressed transcripts from rna-seq data with biological variation," *Bioinformatics*, vol. 28, no. 13, pp. 1721–1728, 2012.

[28] S.-Y. Kim, "Effects of sample size on robustness and prediction accuracy of a prognostic gene signature." *BMC Bioinformatics*, vol. 10, no. 1, p. 147, 2009.

[29] P. Ferragina and G. Manzini, "Indexing compressed text," *Journal of the ACM*, vol. 52, no. 4, pp. 552–581, 2005.

[30] M. Burrows and D. J. Wheeler, "A block-sorting lossless data compression algorithm," Tech. Rep., 1994.

[31] H. Li, J. Ruan, and R. Durbin, "Mapping short dna sequencing reads and calling variants using mapping quality scores," *Genome Research*, vol. 18, no. 11, pp. 1851–1858, 2008.

[32] R. Li, Y. Li, K. Kristiansen, and J. Wang, "Soap: short oligonucleotide alignment program," *Bioinformatics*, vol. 24, no. 5, pp. 713–714, 2008.

[33] N. Homer, B. Merriman, and S. F. Nelson, "Bfast: an alignment tool for large scale genome resequencing," *PLoS One*, vol. 4, no. 11, p. e7767, 2009.

[34] S. M. Rumble, P. Lacroute, A. V. Dalca, M. Fiume, A. Sidow, and M. Brudno, "Shrimp: accurate mapping of short color-space reads," *PLoS Computational Biology*, vol. 5, no. 5, p. e1000386, 2009.

[35] B. Langmead, C. Trapnell, M. Pop, S. L. Salzberg, *et al.*, "Ultrafast and memory-efficient alignment of short dna sequences to the human genome," *Genome Biology*, vol. 10, no. 3, p. R25, 2009.

[36] H. Li and R. Durbin, "Fast and accurate short read alignment with burrows–wheeler transform," *Bioinformatics*, vol. 25, no. 14, pp. 1754–1760, 2009.

[37] D. Weese, A.-K. Emde, T. Rausch, A. Döring, and K. Reinert, "Razers—fast read mapping with sensitivity control," *Genome Research*, vol. 19, no. 9, pp. 1646–1654, 2009.

[38] H. Li and R. Durbin, "Fast and accurate long-read alignment with burrows–wheeler transform," *Bioinformatics*, vol. 26, no. 5, pp. 589–595, 2010.

[39] G. Rizk and D. Lavenier, "Gassst: global alignment short sequence search tool," *Bioinformatics*, vol. 26, no. 20, pp. 2534–2540, 2010.

[40] B. Langmead and S. L. Salzberg, "Fast gapped-read alignment with bowtie 2," *Nature Methods*, vol. 9, no. 4, pp. 357–359, 2012.

[41] Y. Liu and B. Schmidt, "Long read alignment based on maximal exact match seeds," *Bioinformatics*, vol. 28, no. 18, pp. i318–i324, 2012.

[42] A. Ahmadi, A. Behm, N. Honnalli, C. Li, L. Weng, and X. Xie, "Hobbes: optimized gram-based methods for efficient read alignment," *Nucleic Acids Research*, vol. 40, no. 6, pp. e41–e41, Mar. 2012.

[43] L. Huang, V. Popic, and S. Batzoglou, "Short read alignment with populations of genomes," *Bioinformatics*, vol. 29, no. 13, pp. i361–i370, 2013.

[44] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, R. Durbin, and . G. P. D. P. Subgroup, "The sequence alignment/map format and samtools," *Bioinformatics*, vol. 25, no. 16, pp. 2078–2079, 2009.

[45] R. Li, Y. Li, X. Fang, H. Yang, J. Wang, K. Kristiansen, and J. Wang, "Snp detection for massively parallel whole-genome resequencing," *Genome Research*, vol. 19, no. 6, pp. 1124–1132, 2009.

[46] M. A. DePristo, E. Banks, R. Poplin, K. V. Garimella, J. R. Maguire, C. Hartl, A. A. Philippakis, G. Del Angel, M. A. Rivas, M. Hanna, *et al.*, "A framework for variation discovery and genotyping using next-generation dna sequencing data," *Nature Genetics*, vol. 43, no. 5, pp. 491–498, 2011.

[47] D. Challis, J. Yu, U. S. Evani, A. R. Jackson, S. Paithankar, C. Coarfa, A. Milosavljevic, R. A. Gibbs, and F. Yu, "An integrative variant analysis suite for whole exome next-generation sequencing data," *BMC Bioinformatics*, vol. 13, no. 1, p. 8, 2012.

[48] H. Li, "A statistical framework for snp calling, mutation discovery, association mapping and population genetical parameter estimation from sequencing data," *Bioinformatics*, vol. 27, no. 21, pp. 2987–2993, 2011.

[49] R. Nielsen, T. Korneliussen, A. Albrechtsen, Y. Li, and J. Wang, "Snp calling, genotype calling, and sample allele frequency estimation from new-generation sequencing data," *PLoS One*, vol. 7, no. 7, p. e37558, 2012.

[50] Y. Wang, J. Lu, J. Yu, R. A. Gibbs, and F. Yu, "An integrative variant analysis pipeline for accurate genotype/haplotype inference in population ngs data," *Genome Research*, vol. 23, no. 5, pp. 833–842, 2013.

[51] Q. Chen and F. Sun, "A unified approach for allele frequency estimation, snp detection and association studies based on pooled sequencing data using em algorithms," *BMC Genomics*, vol. 14, no. Suppl 1, p. S1, 2013.

[52] M. Forster, P. Forster, A. Elsharawy, G. Hemmrich, B. Kreck, M. Wittig, I. Thomsen, B. Stade, M. Barann, D. Ellinghaus, B.-S. Petersen, S. May, E. Melum, M. B. Schilhabel, A. Keller, S. Schreiber, P. Rosenstiel, and A. Franke, "From next-generation sequencing alignments to accurate comparison and validation of single-nucleotide variants: the pibase software," *Nucleic Acids Research*, vol. 41, no. 1, p. e16, 2013.

[53] H. Li, "Exploring single-sample snp and indel calling with whole-genome de novo assembly," *Bioinformatics*, vol. 28, no. 14, pp. 1838–1844, 2012.

[54] L. E. Mose, M. D. Wilkerson, D. N. Hayes, C. M. Perou, and J. S. Parker, "Abra: improved coding indel detection via assembly based re-alignment," *Bioinformatics*, p. btu376, 2014.

[55] H. Li, "Fermikit: assembly-based variant calling for illumina resequencing data," *Bioinformatics*, p. btv440, 2015.

[56] V. T. Phan, N. S. Vo, and T. R. Sutter, "mdag: a web-based tool for analyzing microarray data with multiple treatments," *BMC Bioinformatics*, vol. 12, no. Suppl 7, p. A7, 2011.

[57] N. S. Vo, V. Phan, and T. R. Sutter, "Predicting possible directed-graph patterns of gene expressions in studies involving multiple treatments," in *Proceedings of the ACM Conference on Bioinformatics, Computational Biology and Biomedicine*. ACM, 2012, pp. 579–581.

[58] N. S. Vo and V. Phan, "Pattern analysis: a web-based tool for analyzing response patterns in low-replication, many-treatment gene expression data," in *Proceedings of the ACM Conference on Bioinformatics, Computational Biology and Biomedicine*. ACM, 2012, pp. 539–541.

[59] V. Phan, N. S. Vo, and T. R. Sutter, "Inferring directed-graph patterns of gene responses in gene-expression studies with multiple treatments," in *5th international conference on Bioinformatics and Computational Biology (BICoB)*, March 2013, pp. 7–12.

[60] N. S. Vo, T. R. Sutter, and V. Phan, "mdag: A web tool for analyzing, visualizing, and interpreting response patterns in gene expression data with multiple treatments," *Advances in Bioscience and Biotechnology*, vol. 4, p. 706, 2013.

[61] N. S. Vo and V. Phan, "Exploiting dependencies of patterns in gene expression analysis using pairwise c," in *Bioinformatics Research and Applications*. Springer, 2013, pp. 173–184.

[62] N. S. Vo and V. Phan, "Exploiting dependencies of pairwise comparison outcomes to predict patterns of gene response," *BMC Bioinformatics*, vol. 15, no. Suppl 11, p. S2, 2014.

[63] N. S. Vo, Q. Tran, N. Niraula, and V. Phan, "A randomized algorithm for aligning dna sequences to reference genomes," in *3rd IEEE International Conference on Computational Advances in Bio and Medical Sciences (ICCABS)*. IEEE, 2013, pp. 1–6.

[64] N. S. Vo, Q. Tran, N. Niraula, and V. Phan, "Randal: a randomized approach to aligning dna sequences to reference genomes," *BMC Genomics*, vol. 15, no. Suppl 5, p. S2, 2014.

[65] V. Phan, S. Gao, Q. Tran, and N. S. Vo, "How genome complexity can explain the hardness of aligning reads to genomes," in *4th IEEE International Conference on Computational Advances in Bio and Medical Sciences (ICCABS)*. IEEE, 2014, pp. 1–2.

[66] V. Phan, S. Gao, Q. Tran, and N. S. Vo, "How genome complexity can explain the difficulty of aligning reads to genomes," *BMC Bioinformatics*, vol. 16, no. 17, p. 1, 2015.

[67] N. S. Vo, Q. Tran, and V. Phan, "An integrated approach for snp calling based on population of genomes," *BMC Bioinformatics*, vol. 15, no. Suppl 10, p. P30, 2014.

[68] J. Quackenbush, "Computational analysis of microarray data," *Nature Reviews Genetics*, vol. 2, no. 6, pp. 418–427, 2001.

[69] M. J. Heller, "Dna microarray technology: devices, systems, and applications," *Annual Review of Biomedical Engineering*, vol. 4, no. 1, pp. 129–153, 2002.

[70] J. D. Hoheisel, "Microarray technology: beyond transcript profiling and genotype analysis," *Nature Reviews Genetics*, vol. 7, no. 3, pp. 200–210, 2006.

[71] M. Garber, M. G. Grabherr, M. Guttman, and C. Trapnell, "Computational methods for transcriptome annotation and quantification using rna-seq," *Nature Methods*, vol. 8, no. 6, pp. 469–477, 2011.

[72] A. Conesa, P. Madrigal, S. Tarazona, D. Gomez-Cabrero, A. Cervera, A. McPherson, M. W. Szcześniak, D. J. Gaffney, L. L. Elo, X. Zhang, *et al.*, "A survey of best practices for rna-seq data analysis," *Genome Biology*, vol. 17, no. 1, p. 1, 2016.

[73] D. Geman, C. d'Avignon, D. Naiman, and R. Winslow, "Classifying gene expression profiles from pairwise mrna comparisons." *Statistical Applications in Genetics and Molecular Biology*, vol. 3, 2004.

[74] D. Geman, B. Afsari, A. C. Tan, and D. Q. Naiman, "Microarray classification from several two-gene expression comparisons," in *Machine Learning and Applications (ICMLA)*, 2008, pp. 583–585.

[75] A. Longacre, L. Scott, and J. Levine, "Linear independence of pairwise comparisons of dna microarray data," *J Bioinform Comput Biol*, vol. 3, no. 6, pp. 1243–62, 2005.

[76] Y. Benjamini and Y. Hochberg, "Controlling the false discovery rate: a practical and powerful approach to multiple testing," *J. R. Statist.*, vol. 57, pp. 289–300, 1995.

[77] M. Bengtsson, A. Ståhlberg, P. Rorsman, and M. Kubista, "Gene expression profiling in single cells from the pancreatic islets of langerhans reveals lognormal distribution of mrna levels," *Genome Research*, vol. 15, no. 10, pp. 1388–1392, Oct 2005. [Online]. Available: http://www.hubmed.org/fulltext.cgi?uids=16204192

[78] D. Hebenstreit, M. Fang, M. Gu, V. Charoensawan, A. van Oudenaarden, and S. Teichmann, "Rna sequencing reveals two major classes of gene expression levels in metazoan cells." *Molecular Systems Biology*, vol. 7, 2011.

[79] H. Wang, Z. Wang, X. Li, B. Gong, L. Feng, and Y. Zhou, "A robust approach based on weibull distribution for clustering gene expression data," *Algorithms Mol Biol*, vol. 6, no. 1, p. 14, 2011.

[80] M. Ohtaki, K. Otani, K. Hiyama, N. Kamei, K. Satoh, and E. Hiyama, "A robust method for estimating gene expression states using affymetrix microarray probe level data," *BMC Bioinformatics*, vol. 11, pp. 183–183, 2010. [Online]. Available: http://www.hubmed.org/fulltext.cgi?uids=20380745

[81] D. W. Huang, B. T. Sherman, and R. A. Lempicki, "Systematic and integrative analysis of large gene lists using DAVID bioinformatics resources," *Nature Protocols*, vol. 4, no. 1, pp. 44–57, 2009, PMID: 19131956.

[82] D. Warde-Farley, S. Donaldson, O. Comes, K. Zuberi, R. Badrawi, P. Chao, M. Franz, C. Grouios, F. Kazi, C. Lopes, A. Maitland, S. Mostafavi, J. Montojo, Q. Shao, G. Wright, G. Bader, and Q. Morris, "The genemania prediction server: biological network integration for gene prioritization and predicting gene function," *Nucleic Acids Research*, vol. 38, no. suppl 2, pp. W214–W220, 2010.

[83] L. Xu, N. Furlotte, Y. Lin, K. Heinrich, M. W. Berry, E. O. George, and R. Homayouni, "Functional cohesion of gene sets determined by latent semantic indexing of PubMed abstracts," *PLoS One*, vol. 6, no. 4, p. e18851, Apr. 2011.

[84] B. Efron, "Bootstrap methods: another look at the jackknife," *The Annals of Statistics*, pp. 1–26, 1979.

[85] A. C. Davison, *Bootstrap methods and their application*, ser. Cambridge Series in Statistical and Probabilistic Mathematics. New York, NY, USA: Cambridge University Press, October 1997, vol. 1.

[86] M.-K. Kwak and T. W. Kensler, "Targeting nrf2 signaling for cancer chemoprevention," *Toxicology and Applied Pharmacology*, vol. 244, no. 1, pp. 66–76, 2010.

[87] Q. Ma, "Role of nrf2 in oxidative stress and toxicity," *Annual Review of Pharmacology and Toxicology*, vol. 53, pp. 401–426, 2013.

[88] Y. Dewa, J. Nishimura, M. Muguruma, M. Jin, Y. Saegusa, T. Okamura, M. Tasaki, T. Umemura, and K. Mitsumori, "β-naphthoflavone enhances oxidative stress responses and the induction of preneoplastic lesions in a diethylnitrosamine-initiated hepatocarcinogenesis model in partially hepatectomized rats," *Toxicology*, vol. 244, no. 2, pp. 179–189, 2008.

[89] M. Kobayashi, L. Li, N. Iwamoto, Y. Nakajima-Takagi, H. Kaneko, Y. Nakayama, M. Eguchi, Y. Wada, Y. Kumagai, and M. Yamamoto, "The antioxidant defense system keap1-nrf2 comprises a multiple sensing mechanism for responding to a wide range of chemical compounds," *Molecular and cellular biology*, vol. 29, no. 2, pp. 493–502, 2009.

[90] J. Dong, D. Yan, and S.-y. Chen, "Stabilization of nrf2 protein by d3t provides protection against ethanol-induced apoptosis in pc12 cells," *PloS one*, vol. 6, no. 2, p. e16845, 2011.

[91] M. Schatz, A. Delcher, and S. Salzberg, "Assembly of large genomes using second-generation sequencing," *Genome Research*, vol. 20, no. 9, pp. 1165–1173, 2010.

[92] B. L. Cantarel, D. Weaver, N. McNeill, J. Zhang, A. J. Mackey, and J. Reese, "Baysic: a bayesian method for combining sets of genome variants with improved specificity and sensitivity," *BMC Bioinformatics*, vol. 15, no. 1, p. 104, 2014.

[93] V. Bansal, O. Harismendy, R. Tewhey, S. S. Murray, N. J. Schork, E. J. Topol, and K. A. Frazer, "Accurate detection and genotyping of snps utilizing population sequencing data," *Genome Research*, vol. 20, no. 4, pp. 537–545, 2010.

[94] C. A. Albers, G. Lunter, D. G. MacArthur, G. McVean, W. H. Ouwehand, and R. Durbin, "Dindel: accurate indel calls from short-read data," *Genome Research*, vol. 21, no. 6, pp. 961–973, 2011.

[95] N. Homer and S. F. Nelson, "Improved variant discovery through local re-alignment of short-read next-generation sequencing data using srma," *Genome Biology*, vol. 11, no. 10, p. R99, 2010.

[96] P. Carnevali, J. Baccash, A. L. Halpern, I. Nazarenko, G. B. Nilsen, K. P. Pant, J. C. Ebert, A. Brownley, M. Morenzoni, V. Karpinchyk, *et al.*, "Computational techniques for human genome resequencing using mated gapped reads," *Journal of Computational Biology*, vol. 19, no. 3, pp. 279–292, 2012.

[97] T. Marschall, I. Hajirasouliha, and A. Schönhuth, "Mate-clever: Mendelian-inheritance-aware discovery and genotyping of midsize and long indels," *Bioinformatics*, p. btt556, 2013.

[98] . G. P. Consortium *et al.*, "A global reference for human genetic variation." *Nature*, vol. 526, no. 7571, pp. 68–74, Oct 2015. [Online]. Available: http://dx.doi.org/10.1038/nature15393

[99] Y. e. a. Shen, "A snp discovery method to assess variant allele probability from next-generation resequencing data," *Genome Research*, vol. 20, no. 2, pp. 273–280, 2010.

[100] B. Langmead, M. C. Schatz, J. Lin, M. Pop, and S. L. Salzberg, "Searching for snps with cloud computing," *Genome Biology*, vol. 10, no. 11, p. R134, 2009.

[101] K. Schneeberger, J. Hagmann, S. Ossowski, N. Warthmann, S. Gesing, O. Kohlbacher, and D. Weigel, "Simultaneous alignment of short reads against multiple genomes," *Genome Biology*, vol. 10, no. 9, p. R98, 2009.

[102] C. Thachuk, "Succincter text indexing with wildcards," in *Combinatorial Pattern Matching*. Springer, 2011, pp. 27–40.

[103] W.-K. Hon, T.-H. Ku, R. Shah, S. V. Thankachan, and J. S. Vitter, "Compressed text indexing with wildcards," *Journal of Discrete Algorithms*, vol. 19, pp. 23–29, 2013.

[104] X. V. Wang, N. Blades, J. Ding, R. Sultana, and G. Parmigiani, "Estimation of sequencing error rates in short reads," *BMC Bioinformatics*, vol. 13, no. 1, p. 185, 2012.

[105] J. M. Zook, B. Chapman, J. Wang, D. Mittelman, O. Hofmann, W. Hide, and M. Salit, "Integrating human sequence data sets provides a resource of benchmark snp and indel genotype calls." *Nature Biotechnology*, vol. 32, no. 3, pp. 246–251, Mar 2014. [Online]. Available: http://dx.doi.org/10.1038/nbt.2835

[106] A. McKenna, M. Hanna, E. Banks, A. Sivachenko, K. Cibulskis, A. Kernytsky, K. Garimella, D. Altshuler, S. Gabriel, M. Daly, *et al.*, "The genome analysis toolkit: A mapreduce framework for analyzing next-generation dna sequencing data," *Genome Research*, vol. 20, no. 9, pp. 1297–1303, 2010.

[107] X. Yu and S. Sun, "Comparing a few snp calling algorithms using low-coverage sequencing data," *BMC Bioinformatics*, vol. 14, no. 1, p. 274, 2013.

[108] X. Liu, S. Han, Z. Wang, J. Gelernter, and B.-Z. Yang, "Variant callers for next-generation sequencing data: A comparison study," *PloS One*, vol. 8, no. 9, p. e75619, 2013.

[109] G. Narzisi, J. A. O'Rawe, I. Iossifov, H. Fang, Y.-h. Lee, Z. Wang, Y. Wu, G. J. Lyon, M. Wigler, and M. C. Schatz, "Accurate de novo and transmitted indel detection in exome-capture data using microassembly," *Nature Methods*, vol. 11, no. 10, pp. 1033–1036, 2014.

[110] H. Li, "Aligning sequence reads, clone sequences and assembly contigs with bwa-mem," *arXiv preprint arXiv:1303.3997*, 2013.

[111] M. Ruffalo, T. LaFramboise, and M. Koyutürk, "Comparative analysis of algorithms for next-generation sequencing read alignment," *Bioinformatics*, vol. 27, no. 20, pp. 2790–2796, 2011.

[112] H. Li, "Towards better understanding of artifacts in variant calling from high-coverage samples," *Bioinformatics*, p. btu356, 2014.

[113] A. Cornish and C. Guda, "A comparison of variant calling pipelines using genome in a bottle as a reference," *BioMed Research International*, vol. 2015, 2015.

Appendix A

mDAG - A Web Tool for Analyzing and Visualizing Gene Expression Data

A.1 Introduction

mDAG is a web-based tool for analyzing and visualizing patterns of gene response to multiple treatments in microarray data. Patterns are represented using directed graphs, which capture precisely primary and subtle gene responses to all treatments. Genes with the same patterns in mDAG hypothetically share similar biological functions or are related on some ways as confirmed by several related works. Users can filter patterns to select how genes respond specifically to individual treatments (affected or not affected, up-regulated or down-regulated by a treatment). Users can also filter patterns based on probe-set ids, gene symbols, uni gene ids, or accession numbers. Gene lists and related necessary information can be downloaded in a proper format and might be further analyzed using tools that allows users to analyze gene funtions based on user-input gene lists. mDAG also incorporates several well-establised tools including DAVID, GeneMANIA, and GCAT to help users exploit these tools more conveniently. The tool will be useful in helping users verify various hypotheses about gene functions based on gene expression data.

The (server/personal/demo) software is freely available at http://cetus.cs.memphis.edu/mdag and its source code is available at http://github.com/namsyvo/mDAG. The software is configurable as a stand-alone application for individual usage, or as an online service on a server for group usage. At this release, the software assumes microarray data as inputs. Subsequent releases will allow data from other technologies that measure gene expression levels.

A.2 Using mDAG

Start your analysis by submitting a dataset. Please use the following format for your dataset. You can use one of the sample data sets while learning how to use the software. If you have already submitted a dataset, you can analyze it with proper

parameters. Go to result page to see whether your analyses are completed. If not, the system will show the approximate remaining time needed to finish the analyses. Otherwise, you can start to work on the analysis result of your datasets.

To understand more about the meaning of the result, you can take a tutorial about mDAG. Figure 32 shows a snapshot of an analysis result.

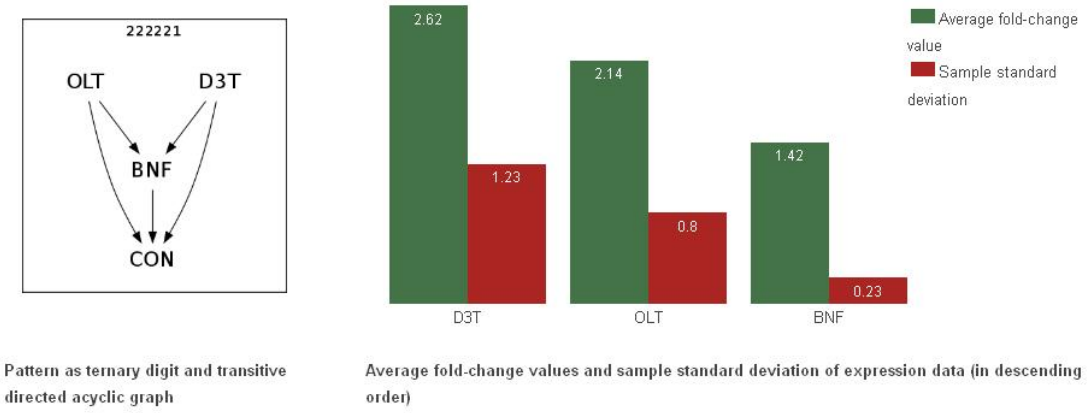

Figure 32: A snapshot of an analysis result from mDAG.

A.2.1 Online Registration

You can use the online version of mDAG without registering and logging in but people will see your data and results of analyses. In addition, mDAG will delete automatically every anonymous data/results which are older than 10 days or are out of 100 items in chronological order. You need to register and login if you do not want your data publically viewable or deleted after a time. For testing purposes, you can use the demo account with email ”mdagdemo@memphis.edu” and password ”mdagdemo”.

A.2.2 Personal Version

In addition to the online version, mDAG can be run as a stand-alone application on a personal computer for individual usage. You can download this software and install on your (Linux) computer to use it via a web interface. A version for Windows computer will be provided soon.

The followings are needed for a personal Linux-version of mDAG, please choose a proper package for your computer:

Python 2.6+ (for Linux)

Graphviz (for Linux)

mDAG (for Linux)

Installing mDAG:

Install Python, Graphviz.

Download the mDAG software and extract into a directory, for example /home/, the result will be the directory /home/mdag

Go to /home/mdag and run mDAG internal server as following: python web2py.py

Enter a password for this server and press ”start server”, the mdag website will display in your default web browser.

Follow the instruction in the website.

A.2.3 Data Format

Data must be stored in a tab-separated text file with Windows-format line ending.

The first row is the header that describes each column.

Each row, starting from the second, corresponds to each gene/probe.

Columns are separated by tabs. The best way to do this is select appropriate columns in your Excel files and save them as text.

First column, Probe Set Id

Second column, Gene Symbol

Third column, UniGene Id

Fourth column, Rep. Pub. Id

Fifth column, Gene Title and other information

Note: the first five columns may have missing values. Missing or even inaccurate values on the first five columns do not affect our analysis. Note, however, that missing or

inaccurate values may cause inaccurate analyses by external resources such as NCBI, GeneMANIA, DAVID, and GCAT.

Sixth column (and the rest) specifies the expression value of each replicate for a treatment.

Replicates for each treatment group must be consecutive. The first one must be Control group.

Expression values must be non-log values.

Figure 33 shows a snapshot of a sample dataset (shown in Excel to see better, but you must convert to text format before uploading).

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Probe Set	Gene Symbol	UniGene ID	Rep. Pub. ID	Gene Title, Chromosomal Location	CON1	CON2	CON3	BNF1	BNF2	BNF3	D3T1	D3T2	D3T3
2	1369943_at	Tgm2	Rn.10	NM_019386	transglutaminase 2, C polypeptide=ch	2177.697	1045.617	1941.678	1032.806	1688.829	1049.005	1195.683	1108.757	1353.181
3	1367657_at	Btg1	Rn.1000	NM_017258	B-cell translocation gene 1, anti-proli	8434.484	9747	7945.055	7135.597	11664.8	7243.845	8300.231	6238.039	5111.921
4	1368270_at	Apobec1	Rn.10002	NM_012907	apolipoprotein B editing complex 1=c	19.79208	19.51061	19.76153	18.66085	20.0334	14.83247	48.79503	22.01243	30.48052
5	1372094_at	Supt5h	Rn.100021	BG380556	suppressor of Ty 5 homolog (S. cerevi	996.1499	867.4282	1035.416	1095.092	985.2087	866.0309	963.4004	1065.601	1134.87
6	1399029_at	Usp48	Rn.100022	BI285965	ubiquitin specific protease 48=chr5q3	102.4236	118.1206	77.58885	130.8167	87.8831	103.5081	158.2999	89.92246	84.65049
7	1376849_at	Usp48	Rn.100022	BM384872	ubiquitin specific protease 48=chr5q3	217.9693	168.7466	149.8477	148.2196	145.8392	182.0112	179.688	101.7338	170.2565
8	1397005_at	Podxl2_predict	Rn.100042	BF393900	Podocalyxin-like 2 (predicted)=chr4q3	6.006983	6.366794	6.387213	7.211494	6.320073	7.459517	6.284494	6.598444	6.611019
9	1368627_at	Rgn	Rn.10006	NM_031546	regucalcin=chrXq11-q12	11479.95	7131.535	6869.644	8259.493	6571.767	6828.345	9313.828	8153.299	7816.688
10	1388858_at	Map2k3	Rn.100064	BI283843	mitogen activated protein kinase kina	1239.593	1518.014	1590.37	1596.66	1284.47	2328.313	1414.504	2150.621	1641.025
11	1398267_at	Slc22a7	Rn.10009 /// r	NM_053537	solute carrier family 22 (organic anior	1029.831	712.9044	906.3523	467.5999	613.8968	606.592	770.9209	621.2246	980.2907
12	1385837_at	Hoxd3_mappe	Rn.100101	AA956624	homeo box D3 (mapped)=chr3q23	7.942043	7.99069	7.999732	8.103678	7.995693	8.076715	7.942137	9.998881	9.076315
13	1371584_at	Trpc4ap	Rn.100109	BM390843	transient receptor potential cation ch	419.7459	388.7067	398.329	350.181	378.3355	335.3587	439.4473	433.1681	446.777
14	1391946_at	Selp	Rn.10012	BI296054	selectin, platelet=chr13q22	8.03969	8.102237	8.26353	8.205031	8.064649	8.132862	8.022334	8.790893	8.623053
15	1369813_at	Dnajc5	Rn.100120	U39320	DnaJ (Hsp40) homolog, subfamily C, n	57.42489	76.12687	59.24449	61.10849	86.04957	60.02731	105.4157	60.85005	60.31765
16	1371870_at	Anxa13_predic	Rn.100125	AI169493	Annexin A13 (predicted)=chr7q33	1011.147	804.3979	830.6408	834.6915	726.6166	777.3952	780.2942	899.1814	832.8329
17	1391346_at	Anxa13_predic	Rn.100125 ///	BG373537	Annexin A13 (predicted) /// Zinc-fing	563.6989	521.5713	570.9722	483.1054	615.4033	556.7912	786.0128	808.4399	977.9739

Figure 33: A snapshot of a sample dataset for mDAG.

Appendix B

RandAL - A Randomized Short Read Aligner

B.1 Introduction

RandAL (Randomized Short Read Aligner) is a tool for aligning DNA sequences to reference genomes. The tool is developed based on a new randomized algorithm with the distinction of having high performance across a wide range of read lengths and base error rates.

RandAL is implemented in C++; FM-index codes are adapted from an external library (http://code.google.com/p/fmindex-plus-plus). The tool has been tested with Debian Squeeze 6.0.6 and Mac OS 10.7.5.

B.2 Using RandAL

B.2.1 Directory Organization

Source Code of RandAL

./src: source code of RandAL.

See MANUAL for detailed information on how to use RandAL.

See also LICENSE, VERSION, and CHANGELOG for other information.

Sample Datasets for Testing RandAL

./data: several datasets for testing.

./data/genomes: store two reference genomes.

./data/genomes/Staphylococcus.fasta (bacterium, http://www.ebi.ac.uk/ena/data/view/Taxon:663951)

./data/genomes/Drosophila melanogaster chromosome 3R.fasta (eukaryote, http://www.ebi.ac.uk/ena/data/view/AE014297)

Genomes are taken from EBI (http://www.ebi.ac.uk). Users also can find reference genomes at NCBI (http://www.ncbi.nlm.nih.gov).

./data/reads: store simulated reads for above genomes.

./data/genomes/Staphylococcus: simulated reads for Staphylococcus.

./data/genomes/Drosophila3R: simulated reads for Drosophila melanogaster chromosome 3R.

Reads are generated with a simulator named wgsim (https://github.com/lh3/wgsim). 100,000 reads with length from 35bps to 400bps are generated with default settings. See https://github.com/lh3/wgsim for detailed information on how to generate simulated reads and evaluate the alignment results.

Supporting Scripts to Run and Evaluate RandAL

./scripts: scripts to support running and testing RandAL.

./scripts/do_exps.py: Python script to test RandAL with multiple simulated datasets.

Usage: python do_exps.py -r ref -l read_len -e error_rate -o result_file_name

Example: python do_exps.py -r Stap -r Dros -l 35 -l 51 -e 0.02 -e 0.04 -o overall_results.txt

./scripts/wgsim_eval.pl: Perl script to evaluate a SAM output and then produce result to screen. Original code from https://github.com/lh3/wgsim.

./scripts/wgsim_eval_tofile.pl: Perl script to evaluate a SAM ouput and then write results (including error mapped reads) to files, used in the script do_exps.py. Modified from wgsim_eval.pl.

B.2.2 Usage

Building RandAL from Source Code

Usage: make [index] [align] [debug] [clean]

index: to construct indexing module of RandAL.

align: to construct alignment module of RandAL.

clean: to clean remove redundant files in source code directory.

debug: to build a debug version of RandAL.

Indexing a Reference Genome

Usage: randal-index RefFileName.fasta [IndexFileName]

RefFileName : reference file name (in FASTA format).

[IndexFileName] : index file name (in randal's binary format), the reference file name will be used if not indicated. This name will be used for the following index files:

IndexFileName.ref

IndexFileName.rev

IndexFileName.bw

IndexFileName.fw

Aligning Reads to the Reference

Usage: randal-align IndexFileName ReadFileName MapFileName [OPTIONS]

IndexFileName : index file name (taken from preprocessing step 1.2. with randal-index).

ReadFileName : read file name (in FASTQ format).

MapFileName : map file name, store alignment result (in SAM format).

[OPTIONS] : a list of zero or more optional arguments, default values will be used if not indicated.

-mode number : distance mode (2: hamming distance; 3: edit distance; default: 3).

-th number : threshould for distance (integer, $>= 0$, default: calculated based on input).

-att number : number of attempts for the randomization (integer, > 0, default: calculated based on input).

-minws number : minimal length of common substring between read and reference (integer, > 0, $<$ read length, default: calculated based on input).

-maxcdt number : maximal number of candidates for alignment (integer, > 0, default: 2).

-err number : base error rate, the probability of each nucleotide being a SNP (float, > 0, default: 0.02).

B.2.3 Default Parameters

Although we do not know what d, the distance between r and its aligned substring r0, is, it can be estimated by the rates of SNP of the given genome. Let b be the probability of each nucleotide being a SNP; sequencing error rate can additionally be accounted for as well. Given read length m, we can assume that this SNP rate is being distributed by a binomial distribution with mean = mb and variance = mb(1 - b). Then parameters can be calculated with pseudo code as following:

$th = floor(error\_rate * read\_length + th\_para * sqrt(read\_length * error\_rate * (1 - error\_rate)))$

$att = att\_para * (th + 1)$

$minws = ceil(read\_length/(th + 1)) + 3$

where: $th\_para = 4, att\_para = 1$

Appendix C

IVC - An Integrated Variant Caller

C.1 Introduction

IVC (Integrated Variant Caller) is a tool for calling genomic variants from next-generation sequencing data. The tool is developed based on a novel approach to variant calling which leverages existing genomic variants to improve the accuracy of called variants, including new variants and close-by INDELs. By design, IVC integrates read alignment, alignment sorting, and variant calling into a unified process. The simplified workflow eliminates many intermediate steps and consequently reduces human intervention and errors.

Our method, IVC, was implemented in the Go programming language (see https://golang.org), which was designed to have high performance, garbage collection, and primitive support for concurrency. Although Go might not be very common currently and other languages such as C are also good choices for implementing high performance software like this, we think that Go is a better choice in this context. It is easier to write simple, reliable, and efficient software in Go, although the most optimal C code might be faster than those of Go. We also provided several pre-complied binary packages of the tool for several common platforms to help users run the tool without installing Go. These pre-complied binary packages were obtained by using the Go's built-in cross compilation feature.

The source code of IVC can be found at https://github.com/namsyvo/IVC. It currently supports Illumina paired-end reads. Other data formats will be supported soon.

C.2 Using IVC

C.2.1 Installation

1. Download IVC Source Code with Go

Pre-requirement: Go environment is already set up properly.

Get IVC source code:

go get github.com/namsyvo/IVC

After these steps, IVC source code should be in the directory $GOPATH/github.com/namsyvo/IVC

Then go to the IVC directory, from which IVC can be run as a Go program:

cd $GOPATH/src/github.com/namsyvo/IVC

2. Download IVC Source Code without Go

Get IVC source code with pre-compiled binary executable files of IVC (compiled on GNU/Linux 3.2.0-4-amd64 #1 SMP Debian 3.2.68-1+deb7u3 x86_64):

git clone https://github.com/namsyvo/IVC.git

cd IVC

Those binary executable files were obtained by compiling source code with Go:

go build main/ivc-index.go

go build main/ivc.go

The source code can be also downloaded from releases of IVC at https://github.com/namsyvo/IVC/releases

C.2.2 Usage

Example Commands

IVC comes with a test dataset which includes the following directories:

./test_data/refs: includes a reference genome and a corresponding variant profile for NC_007194.1 (Aspergillus fumigatus Af293 chromosome 1, whole genome shotgun sequence, see http://www.ncbi.nlm.nih.gov/nuccore/AAHF00000000).

./test_data/reads: includes a set of 10.000 simulated paired-end reads generated with DWGSIM (see https://github.com/nh13/DWGSIM).

1.1. Creating and Indexing Reference Genomes with Variant Profiles:

go run main/ivc-index.go -R test_data/refs/chr1_ref.fasta -V test_data/refs/chr1_variant_prof.vcf -I test_data/indexes

The command ”go run main/ivc-index.go” can be replaced by the command ”./ivc-index.go”.

1.2. Calling Variants from Reads and the Reference:

go run main/ivc.go -R test_data/refs/chr1_ref.fasta -V test_data/refs/chr1_variant_prof.vcf -I test_data/indexes -1 test_data/reads/chr1_dwgsim_100_0.001-0.01.bwa.read1.fastq -2 test_data/reads/chr1_dwgsim_100_0.001-0.01.bwa.read2.fastq -O test_data/results/chr1_variant_calls.vcf

The command ”go run main/ivc.go” can be replaced by the command ”./ivc.go”.

Commands and Options

2.1. Creating and Indexing Reference Genomes with Variant Profiles:

Required:

-R: reference genome (FASTA format).

-V: known variant profile (VCF format).

-I: directory for storing index.

2.2. Calling Variants:

Required:

-R: reference genome (FASTA format).

-V: known variant profile (VCF format).

-I: directory for storing index.

-1: the read file (for single-end reads) (FASTQ format).

-2: the second end file (for pair-end reads) (FASTQ format).

-O: variant call result file (VCF format).

Options:

-d: threshold of alignment distances (float, default: determined by the program).

-t: maximum number of CPUs to run (integer, default: number of CPU of running computer).

-r: maximum number of iterations for random searching (int, default: determined by the program).

-s: substitution cost (float, default: 4).

-o: gap open cost (float, default: 4.1).

-e: gap extension cost (float, default: 1.0).

-mode: searching mode for finding seeds (1: random (default), 2: deterministic).

-start: starting position on reads for finding seeds (integer, default: 0).

-step: step for searching in deterministic mode (integer, default: 5).

-maxs: maximum number of seeds for single-end reads (default: 1024).

-maxp: maximum number of paired-seeds for paired-end reads (default: 128).

-lmin: minimum length of seeds for each end (default: 15).

-lmax: maximum length of seeds for each end (default: 30).

-debug: debug mode (boolean, default: false)

C.2.3 Preparing Data and Performing Experiments

Simulated Data

IVC comes with a simulator which simulates mutant genomes based on the reference genome and its associated variant profile. Reads are then can be generated from the mutant genome using other simulators, such as DWGSIM.

Get the mutant genome simulator:

git clone https://github.com/namsyvo/ivc-tools.git

cd ivc-tools/genome-simulator

Then follow the instructions to generate simulated mutant genomes and evaluate the called variants.

Real Data

- Reference genome: the NCBI Genome Reference Consortium, GRCh37 p.13. http://www.ncbi.nlm.nih.gov/assembly/GCF_000001405.25/

- Known variant profile: the 1000 Genomes Project Consortium, Phase 1.

hftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/phase1/analysis_results/integrated_call_sets/

- Whole genome simulation: DWGSIM.
  https://github.com/nh13/dwgsim

- Mutant genome generation: our script.
  https://github.com/namsyvo/varcall-tools/tree/master/ivc-tools/genome-simulator

- Real data, Illumina HiSeq 2000 paired-end, sample ERR194147 (individual NA12878), read length 100, coverage ∼50x:
  http://www.illumina.com/platinumgenomes
  http://www.ebi.ac.uk/ena/data/view/ERR194147

- GAIB data: The Genome In A Bottle Consortium.
  ftp://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/release/NA12878_HG001/NISTv2.19

Commands and Settings

Scripts to run all tools are exist publicly in github:

- Scripts to run IVC on batch commands:
  https://github.com/namsyvo/varcall-tools/tree/master/ivc-tools

- Commands and parameters to run IVC (running with default parameters, version 0.8.1).

  ./ivc-index -R ref.fasta -V tdbSNP -I index-dir

  ./ivc -R ref.fasta -V dbSNP -I index-dir -O ivc-var-call.vcf -1 read1.fastq -2 read1.fastq