SCALABLE MAPPING AND COMPRESSION OF HIGH THROUGHPUT GENOME SEQUENCING DATA

by

Faraz Hach

B.Sc., Sharif University of Technology, 2004

M.Sc., Simon Fraser University, 2007

A Thesis submitted in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy

in the
School of Computing Science
Faculty of Applied Sciences

SIMON FRASER UNIVERSITY
Summer 2013

APPROVAL

Name:	Faraz Hach
Degree:	Doctor of Philosophy
Title of Thesis:	SCALABLE MAPPING AND COMPRESSION OF HIGH THROUGHPUT GENOME SEQUENCING DATA
Examining Committee:	Gregory Baker, Senior Lecturer Chair
	Dr. S. Cenk Sahinalp, Professor Senior Supervisor
	Dr. Colin Collins, Professor Dept. of Urologic Sciences, University of British Columbia Supervisor
	Dr. Peter Unrau, Associate Professor Dept. of Molecular Biology and Biochemistry, SFU Internal Examiner
	Dr. Bonnie Berger, Professor Computer Science and Artificial Intelligence Lab, Massachusetts Institute of Technology External Examiner
Date Approved:	July 29, 2013

Partial Copyright Licence

Abstract

The high throughput sequencing (HTS) platforms generate unprecedented amounts of data that introduce challenges for processing, downstream analysis and computational infrastructure. HTS has become an invaluable technology for many applications, e.g. the detection of single-nucleotide polymorphisms, structural variations. In most of these applications, mapping sequenced "reads" to their potential genomic origin is the first fundamental step for subsequent analyses. Many tools have been developed to address this problem. Because of the large amount of HTS data availability, much emphasis has been placed on speed and memory. In fact, as HTS data grow in size, data management and storage are becoming major logistical obstacles for adopting HTS-platforms. The requirements for ever increasing monetary investment almost signalled the end of the Sequence Read Archive hosted at the National Center for Biotechnology Information, which holds most of the sequence data generated world wide. One way to solve storage requirements for HTS data is compression. Currently, most HTS data is compressed through general purpose algorithms such as gzip. These algorithms are not specifically designed for compressing data generated by the HTS-platforms. Recently, a number of fast and efficient compression algorithms have been designed specifically for HTS data to address some of the issues in data management, storage and communication.

In this thesis, we study both of these computational problems, i.e., Sequence Mapping and Sequence Compression extensively. We introduce two novel methods namely mrsFAST and drFAST to map HTS short-reads to the reference genome. These methods are cache oblivious and guarantee perfect sensitivity. Both are specifically designed to address the bottleneck of multi-mapping for the purpose of structural variation detection. In addition we present Dissect for mapping whole trascriptome to the genome while considering structural alterations in the transcriptome. Dissect is designed specifically to map HTS long-reads

as well as assembled contigs. Finally, we address the storage and communication problems in HTS data by introducing SCALCE, a "boosting" scheme based on Locally Consistent Parsing technique. SCALCE re-orders the data in order to increase the locality of reference and subsequently improve the performance of well-known compression methods in terms of speed and space.

To My Angels
Sevil and Raheleh

"We know through painful experience that freedom is never voluntarily given by the oppressor; it must be demanded by the oppressed."

— *Martin Luther King, Jr.* Letter From Birmingham Jail, *1963*

Acknowledgments

First and foremost, I extend my utmost and sincerest gratitude to my senior supervisor, Dr. Cenk Sahinalp, who has supported me throughout my graduate studies with his encouragement, guidance and resourcefulness. In addition to technical skills in computer science, I have learned from him the requirements for being a good researcher.

I offer my regards to my supervisor, Dr. Colin Collins. I would like to sincerely thank Dr. Peter Unrau and Dr. Bonnie Berger, who graciously accepted to be my examiners and helped me with valuable discussions and comments. I give special thanks to Gregory Baker, who kindly accepted to be the chair of my examining committee.

I would also like to thank my collaborators Dr. Evan Eichler, Dr. Inanc Birol, Dr. Can Alkan, Dr. Fereydoun Hormozdiari, Dr. Iman Hajirasouliha, Dr. Phuong Dao, Ibrahim Numanagic, Deniz Yorukoglu, Iman Sarrafi, Yen-Yi Lin, Andrew McPherson, Lucas Swanson, Farhad Hormozdiari and Ermin Hodzic. I benefited greatly from these collaborations, and hope to continue working with them.

I would also like to thank all my dear friends: Mayssam Mohammadi, Sara Adineh, Kamyar Khodamordai, Amir Hedayaty, Mohammad Tayebi, Naser Ghazali, Michael Hartman and Pang Hartman.

Last, but not least, I heartily thank my family for the strong motivation that they gave me to follow my studies. Their support was invaluable to me.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The development of High Throughput Sequencing (HTS) technologies has changed the way genomics research is conducted since their inauguration in 2005 [82]. The first commercially available HTS technology was from Roche/454 Life Sciences [82] and was used to sequence the genome of James Watson [117]. It was followed by "second generation" sequencing platforms that generate orders of magnitude more data for a fraction of the cost, such as Illumina Genome Analyzer [10] and AB SOLiD [83]. HTS technologies continue to evolve as years progress, with the introduction of single molecule sequencing (i.e. HeliScope [98] and PacBio RS [21]) and more recently the nanopore sequencing.

Improvements in HTS technologies have empowered researchers in cataloging normal human genome variation [1, 2, 89], building *de novo* genome assemblies [36], finding disease causing mutations [97, 78] and detecting genes associated with diseases such as the Kabuki syndrome [94] and mental retardation [113].

Genomic variation between individuals or across species range from single nucleotide polymorphisms (SNPs) [71, 10], small insertions/deletions (indels) [66], larger size structural variation in the form of insertions, deletions and inversions [60, 65, 46, 16], segmental duplications as well as copy-number polymorphism [5] to larger chromosomal rearrangements [4].

Genomic structural alterations involving transcribed regions of the genome will appear in the associated transcript sequences. Although the whole transcriptome is much smaller than the whole genome, in the context of structural alterations, RNA-Seq data can be more difficult to analyze, partially due to splicing, which can produce several transcripts from the same gene. In comparison to the *wild-type* transcripts, post-transcriptional processes can

also introduce structural alterations into these sequences.

With the advent of High Throughput Transcriptome Sequencing (RNA-Seq), the problem of identifying structural alterations in the transcriptome is attracting significant attention. Several methods for detecting structural alterations using RNA-Seq have been developed [93, 67, 53, 35, 8, 74, 104, 84, 85, 86], mostly focusing on gene fusions, partially due to their abundance in cancer, but also due to the relative ease of identifying them computationally.

Although HTS has become an invaluable technology in these studies, the computational analysis of the data they generate is far from perfect. Continuous development and validation of novel algorithms for discovering genetic variation will be essential for the analysis of the growing volume of data generated by the HTS platforms. In the context of the 1000 Genomes Project for example, such algorithms are and will be used to catalog normal human genomic variation. Likewise, in ongoing cancer genomes projects, these algorithms are and will be employed to identify variations associated with disease states.

Analysis of genomic (or trascriptomic) variations using sequencing starts with mapping the randomly sheared and ideally uniformly sampled DNA (or RNA) fragments from the donor to the reference genome. Different properties and error models of sequence reads generated by these technologies require the development of specialized read mapping algorithms for each platform for accurate read alignment and characterization of genomic variants. This becomes more complicated for short reads: due to repeats and duplications in genomes, they can map to multiple locations with equal sequence identity. [5] reported that in a human resequencing study reads of length 36-bp can be mapped to about 1,800 locations on the average within two mismatches and/or indels.

Typically, the advantages and disadvantages of a read mapper is defined by the project needs, and the "best" tool should be selected depending on the biological question at hand [31]. Leveraging the high sequence coverage and randomly selecting one "best" location when a read cannot be unambiguously placed has proven to be effective in discovering SNPs and small indels in relatively non-complex areas of the genome [71]. However, structural variation detection sensitivity is shown to benefit from tracking "all" map locations of the reads including suboptimal alignments [46, 66, 89], and characterization of segmental duplications is extremely resistant against mapping the reads uniquely [5, 110]. Even with the increased read length, recent studies [46, 66, 5] suggest that ambiguity in read mapping is a bigger problem than was anticipated.

The first generation of mappers for short reads were simply based on a *brute force* approach; the main objective of these mappers was to map very short reads ($\approx$ 25bp) unambiguously. With the introduction of second generation of HTS platforms and longer reads, new algorithms have been designed to address the mapping problem. These read mappers can be broadly classified into two categories according to the method used to index the reference genome using either hash tables or suffix arrays (compressed through the Ferragina-Manzini index [29] with the use of the Burrows-Wheeler Transform [15]). Hash-based aligners such as MAQ [71], ZOOM [75], Mosaik [43], SHRiMP [100], mrFAST [5], BFAST [44], SOAP [72], RazerS [115], RazerS3 [116], YAHA [26], GSNAP [120], SRmapper [38], Gapped BLAST [6], BLAT [55], Exonerate [107], EST_GENOME [91], and GMAP [121] have poorer performance in comparison to suffix array-based aligners such as BWA [68], BWA-SW [69], Bowtie [63], Bowtie2 [62], TopHat [112], TopHat2 [58], SOAP2 [73] and GEM [80] when dealing with short reads. However, their relative performance increases considerably and surpasses the suffix array-based aligners when the read length, and thus, the number of errors (mismatches or indels) that need to be tolerated increase.

The accuracy of the structural variation discovery is directly correlated to the sensitivity and specificity of these mappings tools. The computational bottle-neck for a typical genome variation study is the mapping step [46]; a faster and more accurate mapping method than what is available today is going to be a valuable tool towards realizing the goals of personal genomics and medicine [81]. Since most structural variation occurs within repeat regions (both low copy repeats and common repeat elements), it is critical to consider all possible mapping locations for each read.

Although BWT-FM based methods accelerate read mapping significantly over hash based methods when the goal is to find *the best* matching location, these methods seriously limit the scope for structural variation studies as argued above. Furthermore, FM index is, by nature, for finding exact matches, and extending it to find mappings with mismatches is only achieved by some adhoc heuristics, which results in reduction of accuracy and efficiency. Unfortunately, for the above algorithms, the mapping time increases exponentially with the number of mismatches and indels allowed.

As HTS data grows exponentially in size, data management and storage have become major logistical obstacles for adopting HTS platforms. The requirements for ever increasing monetary investment almost signalled the end of the Sequence Read Archive hosted at the National Center for Biotechnology Information (NCBI), which holds most of the sequence

data generated world-wide. One way to solve storage requirements for HTS data is compression. Currently, most HTS data is compressed through general purpose algorithms such as gzip. These algorithms are not specifically designed for compressing data generated by the HTS platforms; for example, they do not take advantage of the specific nature of genomic sequence data which includes limited alphabet size and high similarity among reads. The need for improved performance has recently lead to the development of a number of techniques specifically for HTS data. Available compression techniques for HTS data either exploit (1) the similarity between the reads and a reference genome or (2) the similarity between the reads themselves. Once such similarities are established, each read is encoded by the use of techniques derived from classical lossless compression algorithms such as Lempel-Ziv-77 [124] (which is the basis of gzip) or Lempel-Ziv-78 [125] .

Compression methods that exploit the similarity between individual reads and the reference genome use the reference genome as a "dictionary" and represent individual reads with a pointer to one mapping position in the reference genome, together with additional information about whether the read has some differences with the mapping loci. As a result, these methods [50, 61] require (i) the availability of a reference genome and (ii) mapping of the reads to the reference genome. Unfortunately, genome mapping is a costly step time-wise, especially when compared to the actual execution of compression (i.e. *encoding* the reads) itself. Furthermore, these methods necessitate the availability of a reference genome both for compression and decompression. Finally, many large-scale sequencing projects such as the Genome 10K Project [42] focus on species without reference genomes.

Compression methods that exploit the similarity between the reads themselves either (i) reorder the reads to increase similarity and then apply bzip [18], or encode the differences of a read to another similar read [122]; or (ii) simply concatenate the reads to obtain a single sequence and then apply Huffman Coding [111, 19] or apply context modelling with arithmetic coding [54, 12].

Most of genomes sequenced from same species (i.e. human) are very similar to the ones already collected. This means that the amount of new sequenced information is growing slowly. [77] shows that this redundant information can be compressed in such a manner that the analysis can be done directly on the compressed data.

1.1 Contribution

In this thesis, we focus on two computational problems involving HTS data namely "HTS Sequence Mapping" and "HTS Sequence Compression". Our goal is to design efficient algorithms for sequence mapping in order to address the computational bottleneck for structural variation as well as providing an HTS specific compression tool to handle the large throughput of the these sequencing technologies. More specifically we present the following contributions:

- We introduce *mrsFAST* [39], a read mapping tool that utilizes specialized data structures which *mathematically guarantee* to (1) achieve optimal cache performance through the use of cache obliviousness paradigm, and (2) find *all* mapping locations for each read within a user specified error threshold. We show that *mrsFAST* has superior performance, both in terms of speed and accuracy in comparison to all popular read mapping algorithms, especially for longer read lengths. We also show that the mappings provided by *mrsFAST* coupled with state of the art structure variation detection algorithms [46] capture deletions that can not be detected by single mapping based approaches.

- We introduce a new SNP-aware read mapper developed for the Illumina platform, *mrsFAST-Ultra*, that improves mappability, mapping accuracy, and sensitivity by tolerating sequence variants that were previously reported as real variants and distinguishing them from likely sequencing errors. mrsFAST-Ultra provides full sensitivity, and it supports multi-mapping. We show that mrsFAST-Ultra is up to 4.5 times faster than its predecessor mrsFAST. In comparison to newly enhanced popular tools such as BWA (newest release) and Bowtie 2, it is more sensitive (it can report 60 times or more mappings per read) and much faster (5 times or more) in the multi-mapping mode.

- We present *drFAST* [48], a read mapper designed for di-base encoded "color-space" sequences generated with the ABI SOLiD platform. *drFAST* is specially designed for better delineation of structural variants including segmental duplications and is able to return *all* possible map locations and underlying sequence variation of short reads within a user-specified distance threshold. We show that *drFAST* is more sensitive in comparison to all commonly used aligners (for color-space reads) such as Bowtie,

BFAST, and SHRiMP. *drFAST* is also faster than both BFAST and SHRiMP and achieves a mapping speed comparable to Bowtie.

- We introduce two novel algorithmic formulations for identifying transcriptomic structural variants through aligning transcripts to the reference genome under the consideration of such variation. The first formulation is based on a nucleotide-level alignment model; a second, potentially faster formulation is based on chaining fragments shared between each transcript and the reference genome. Based on these formulations, we introduce a novel transcriptome-to-genome alignment tool, *Dissect* [123], which can identify and characterize transcriptomic events such as duplications, inversions, rearrangements and fusions. *Dissect* is suitable for whole transcriptome structural variation discovery problems involving sufficiently long reads or accurately assembled contigs.

- Lastly, we present *SCALCE* [40], a "boosting" scheme based on Locally Consistent Parsing technique which reorganizes the reads in a way that results in a higher compression speed and compression rate, independent of the compression algorithm in use and without using a reference genome. We show that SCALCE obtains significant improvements in both compression rate and running time over alternative methods.

In addition to our primary contributions to the sequence mapping and compression problems mentioned above, our other contributions to the field of Computational Biology can be found in [119, 64, 47, 76, 85, 49].

1.2 Organization of the thesis

The rest of the thesis is organized as follows: In Chapter 2, we describe sequence mapping and sequence compression problems. Then we present an overview of the existing related computational approaches and summarize the general issues related to previous work. In Chapter 3, we introduce *mrsFAST*, our cache oblivious mapper and present experimental results. In Chapter 4, we present an extension to *mrsFAST*, *mrsFAST ultra* that focuses on optimizing the search methodologies of *mrsFAST*. Chapter 5 describes two variants of *drFAST*, our color space mapping tool and its experimental results. In Chapter 6, we focus on *Dissect*, our trascriptome to genome mapping tool that considers the structural alterations during the mapping. Chapter 7 introduces *SCALCE*, our HTS compression tool

that uses Locally Consistence Parsing to increase the locality of HTS data in order to achieve higher compression ratio. Finally, in Chapter 8, we offer a summary and conclusion of our contributions to HTS technologies, as well as a discussion of possible directions for future work.

Chapter 2

Background and Related Work

In this chapter, we formally define the sequence mapping and sequence compression problems. For each problem, we provide the preliminaries followed by an overview of existing tools. Finally, we offer a general summary of the issues over the existing methods.

2.1 Sequence Mapping

2.1.1 Definitions

Sequence Mapping Problem: For a given reference sequence S, a set of read sequences R, a scoring function δ and error threshold e, find the set of locations (mappings) in S where the read sequence has a distance $\leq e$ under function δ. The common distance measures are Hamming Distance (mismatches) and Edit Distance (mismatches, insertions and deletions). For example in Hamming Distance case, δ function can be defined as: mismatch:+1, indel: $-\infty$ and match:0.

Best Mapping: For any given read $r \in R$, best mapping location is "one" location in the reference sequence that is the most probable location that r is originated from. The measures that can be used to define this likelihood is the "minimum" number of edit operations or the"minimum" sum of Phred quality [24] score of error locations in r. Phred quality scores Q is defined as a property which is logarithmically related to the base-calling error probabilities P ($Q = -10\log_{10} P$). For example, if Phred assigns a quality score of 30 to a base, the probability that this base is called incorrectly is 0.001.

All Mapping: For any given read $r \in R$, all the possible locations in the reference sequence S with a distance $\leq e$.

2.1.2 Burrows Wheeler Transform, Suffix Arrays, FM-index and Exact Matching

Let Σ be an alphabet, \$ a symbol not in Σ which is lexicographically smaller than all the symbols in Σ. Given a string $X = a_0 a_1 a_{n-1}$ of length n where $a_n = \$$, let $X[i] = a_i$ be the i-th symbol of X, $X[i, j] = a_i \dots a_j$ a substring of X and $X_i = X[i, n-1]$ a suffix of X. Suffix array [79] SA of X is a permutation of the integers $0 \dots n-1$ such that $SA(i)$ is the start position of the i-th smallest suffix.

For a substring W of X, the position of each occurrence of W in X will occur in an interval in the suffix array. Let R_l and R_h be the beginning and the end of this interval in suffix array. R_l and R_h can be defined as:

$$R_l = \min\{k : W \text{ is the prefix of } X_{S(k)}\} \tag{2.1}$$

$$R_h = \max\{k : W \text{ is the prefix of } X_{S(k)}\} \tag{2.2}$$

The naive way to find this interval requires $O(|W| \log |X|)$ time. [79] provided a faster solution to find this interval with time complexity of $O(|W| + \log |X|)$ by pre-calculating longest common prefix between W and the suffix array entries.

The BWT of X is defined as $B[i] = \$$ when $SA(i) = 0$ and $B[i] = X[SA(i) - 1]$ otherwise. The naive way to construct BWT of a given string is to add \$ to the end of string, then sort all the circular shifts of the string and use the last column as BWT. This approach is quadratic in time and space. It is efficient to construct the suffix array first and then generate BWT. Most algorithms for constructing suffix array require at least $n\lceil \log_2 n \rceil$ bits of working space. [45] gave an algorithm that uses n bits of working space and time complexity of $O(n \log n)$.

Let $C(a)$ be the number of symbols in $X[0, n-2]$ that are lexicographically smaller than $a \in \Sigma$ and $Occ(a, i)$ the number of occurrences of a in $B[0, i]$. Ferragina and Manzini [29] showed that if W is a substring of X:

$$R_l(aW) = C(a) + Occ(a, R_l(W) - 1) + 1 \tag{2.3}$$

$$R_h(aW) = C(a) + Occ(a, R_h(W)) \tag{2.4}$$

and that $R_l(aW) \leq R_h(aW)$ if and only if aW is a substring of X. This means that $O(|W|)$ time is required to check if W is a substring of X and also count the number of occurrences of W in X. This is called "backward search".

The algorithm described above needs to load the occurrence array Occ and the suffix array SA in the memory. Given a genome of size n, the occurrence array $Occ(.,.)$ requires $4n\lceil \log_2 n \rceil$ bits as each integer takes $\lceil \log_2 n \rceil$ bits and there are $4n$ of them in the array ($|\Sigma| = 4$). In practice, one can store in memory $Occ(., k)$ for k that is a factor of 128 and calculate the rest of elements using B on the fly. When two bits are used to represent a nucleotide, B requires $2n$ bits. The memory for backward search is thus $2n + n\lceil \log_2 n \rceil/32$ bits. Enumerating the position of each occurrence requires the suffix array SA. If the entire SA is put in memory, it would use $n\lceil \log_2 n \rceil$ bits. $SA(k)$ is stored in memory only for those k that are divisible by 32. For ks that are not a factor of 32, one can repeatedly apply "find previous character" $(C(B[i]) + Occ(B[i], i))$ for some iterations until SA is available (factor of 32). The memory requirement reduces to $2n + n\lceil \log_2 n \rceil/16$. This is about 1.2Gb for human genome.

2.1.3 Existing Methods

MAQ

MAQ [71] is a hash based best-mapper. Its criteria for best mapping is to minimize the sum of the quality values for all the mismatched bases. If there are multiple equally best positions, then one of them is chosen at random. MAQ guarantees to find mappings with up to two mismatches in the first 28 bp of the read, namely the seed section.

MAQ builds multiple hash tables to index the reads and scans the reference sequence against the hash tables to find the potential mapping locations. By default, six hash tables are used, ensuring that a sequence with two mismatches or fewer will be hit. The six hash tables correspond to six non-contiguous seed templates. Suppose that the seed size is 8 bp, the six templates are 11110000, 00001111, 11000011, 00111100, 11001100, and 00110011, where nucleotides at 1 will be indexed while those at 0 will not.

By default, MAQ indexes the first 28 bp of the reads, which are typically the most accurate part of the read. In the mapping phase, MAQ loads all reads into memory and then applies the first template as follows. For each read, MAQ takes the nucleotides at the 1 positions of the template and hashes the read. It, then, groups the reads with the

same hash value. When indexing is complete, MAQ scans the reference base by base on both forward and reverse strands. Each 28 bp subsequence of the reference will be hashed through the same template used in indexing and will be looked up in the six hash tables. If a hit is found for a read, MAQ will calculate the sum of the qualities of mismatched bases over the whole length of the read, extending out from the 28-bp seed without gaps MAQ will re-iterate this for all the templates.

SOAP

SOAP [72] is a hash based best-mapper. It allows either a certain number of mismatches (at most 2) or one continuous gap (1-3 bp) for mapping a read onto the reference sequence (no mismatches are allowed in the flanking regions). Its criteria for best mapping is either the minimum number of mismatches or a smaller gap.

To allow two mismatches, a read is partitioned into four fragments, the two mismatches can exist in at most two of the fragments at the same time, this will generate six combinations of the two fragments as seeds. SOAP loads reference sequences into memory and builds the seed index tables for the reference sequence. Then for each read, it creates seeds and searches the corresponding index table for candidate hits. Finally, it performs alignment and report the results.

SOAP uses a look-up table to accelerate the alignment. To reduce the space, both the reads and the reference sequences are converted into 2-bits-per-base encoding. A read will do exclusive-OR comparison with the reference sequence to find the number of different bases.

Since mismatches are not allowed in gapped hits, SOAP uses the enumeration algorithm which tries to insert a continuous gap or delete a fragment at each possible position in a read.

SRmapper

SRmapper [38] is a hash based best-mapper. SRmapper starts by building an index on the reference genome. The index takes the form of a hash table. To construct the hash table, SRmapper takes k consecutive bases from reference and encodes each base with 2 bits. The resulting number is the hash value for the k-mer. This hash value points to a bucket containing the locations on the genome that share the same k-mer. SRmapper builds

a collision free hash table with $O(1)$ look up. This means that two different k-mer cannot have the same hash value. This results in 4^k entries. This requires huge amount of memory for a mammalian genome like human (approx. 16GB). To reduce the memory requirement, SRmapper divides a genome into non-overlapping sequences containing k bases and stores the locations of these sequences in the corresponding buckets. For a reference sequence of length R, k is chosen according to $k = \lfloor \log_4 R \rfloor$. In case of human genome, k is 15. R/k locations are stored in the hash table. SRmapper loads 1/4 of the index at a time and then iterates over the four parts. This is done to reduce the memory usage.

SRmapper uses seed and extend strategy to find the best mapping location for the read. In the first step, it calculates the hash value for the first k bases (1 through k), and then retrieves the possible candidate locations from the index. In the second step, the remaining bases are compared with their corresponding bases in the reference. A mapping is considered to be proper if, on comparing all bases against the reference, the number of mismatches is less than or equal to the allowed number of mismatches. If a mapping is found, the error threshold e is then decreased to the number of mismatches in that mapping, and the location of the alignment and the number of mismatches is stored. SRmapper iterates these two steps until $(e + 2)k$ bases are covered. To justify this approach, suppose there is no error in the read, then SRmapper needs to use at least k consecutive bases as starting point for k-mer since the genome is being indexed with non-overlapping k-mers. This means that it requires to cover $2k$ bases. If there is one error in the read, it will happen at most in k of the k-mers. Then we need to continue at least $2k$ more bases to be able to find any possible mappings.

To speed up the mapping process SRmapper introduces two heuristics: (i) SRmapper reduces the error threshold by 1 if two mappings with an equal number of mismatches are found for a read. This modification has no effect on the number of confident alignments achieved; (ii) SRmapper limits the number of locations that should be verified from each bucket. This obviously will reduce sensitivity in trade for speed.

GSNAP

GSNAP [120] is a hash based best-mapper. Searching involves the steps of generating, filtering and verifying candidate genomic regions. Similar to SRMapper, GSNAP reduces the memory size by indexing 12-mers every 3 nt in the genome.

Depending on the read length L and the error threshold e, a multiway merging process can be used in two different ways to generate and filter genomic regions for verification. For

smaller values of e, GSNAP uses a multiway merging based on "Spanning Set" of k-mers which filters genomic regions based on the number of k-mers. For larger values of e, GSNAP employs a merging process based on "Complete Set" of k-mers, which filters genomic regions based on the pattern of k-mers that support the region. Both methods provide lower bounds on the number of mismatches present in a read that can be used for filtering.

Spanning Set. A spanning set is a minimal set of 12-mers that covers the read. This structure exploits the pigeonhole principle that the number of non-supporting 12-mers (elements), those that fail to contain a given position in their corresponding position list, provides a lower bound on the number of mismatches in the read. Because of the indexing scheme that indexes the genome every 3 nt, GSNAP constructs six spanning sets, one for each shift of 0, 1 and 2 nt in both the forward and reverse complement directions. These sets of 12-mers are non-overlapping, so the pigeonhole principle now holds where k non-supporting elements implies a lower bound of k mismatches, and the region may be filtered out if $k > e$. Although it is possible to use all spanning set elements to generate candidates and then proceed to the verification step, GSNAP assigns some of the elements for generating candidates and reserves the others for a secondary filtering step. In short, GSNAP performs a multiway merge on the position lists of generating elements. For each position, it counts the support. If the support is high enough to allow at most e non-supporting elements, then it will go through another step to check the supporting elements. If the candidate has more than e non-supporting elements, then it will be eliminated, otherwise, it will be verified as a potential mapping position. Allocation of N elements between generating and filtering depends on allowed number of mismatches (e). At least $(e + 1)$ elements must be generating to guarantee that at least one generating element has support for a candidate region while the e other generating elements do not. $N - e - 1$ elements can be used as filtering elements. Usually it is better to choose the longer list for filtering. To illustrate this, consider a read of length 120 bp. The number of elements (non-overlapping 12-mers) is 10. If the mapping locations with less than 4 mismatches are desired, GSNAP can assign 6 elements as generating elements and 4 as filtering elements. If a region has a support of at least 2 then it will go for filtering step. If one of the filtering elements does not contain this region, it will be filtered.

Complete Set. GSNAP employs a method based on the complete set of overlapping 12-mers to allow more mistmaches. This approach works for any e as long as there is a

consecutive 14 bp region shared between the read and the genome. Candidates are generated by performing a multiway merge of position lists for all read locations in a single forward and single reverse complement. During this merge, locations of the 12-mers that support each candidate region is stored (i.e. location t is supported by number of 12-mers that start at locations 1, 3, 9, ...). The pattern of the supporting 12-mers provides a lower bound on mismatches in the read. If the supporting 12-mers have read locations separated by Δp, then the minimum number of mismatches between them is $\lfloor(\Delta p + 6)/12\rfloor$. Over the entire read, GSNAP can sum these lower bounds to evaluate if the reads should be verified.

YAHA

YAHA [26] is a hash based best-mapper. Similar to GSNAP, YAHA builds a genome index. (It can handle different k-mer size and different distance between two consecutive k-mers). YAHA uses a hash table index to locate the set of locations (seeds) where each k-mer in the query sequence appears as a subsequence of the reference. YAHA next joins these seeds together to form extended seeds ("fragments" of contiguous matching bases between the query and the reference). To find extended seeds, many aligners collect all N seeds for a query into an array and sort them to collocate seeds to be placed in a fragment. However, since the seeds for each k-mer are pre-sorted in the genome index, YAHA does L-way merging (L is read length).

After obtaining the extended fragments, YAHA tries to find the best potential mapping in each region of the reference by combining the fragments that contribute to the highest estimated alignment score in that region. YAHA calculates the estimated score for each collection of fragments with affined gap penalties; fragments are scored as matches, while differences between fragment diagonals are scored as a single indel. YAHA builds a graph where each node in the graph represents a fragment and each edge represents the cost of one fragment succeeding another one. Fragments earlier in the query can only be succeeded by fragments later in the query, thus the graph is directed and acyclic. Finding maximum scoring path requires visiting each edge only once in the proper (topological sort) order. By placing the nodes in an array and sorting them by starting position in the read, YAHA can perform the graph algorithm without ever forming the edges explicitly. Each node is visited sequentially while checking against all nodes above it in the sorted array. If an edge is allowed between two nodes, YAHA scores the edge, updating the best score and best path. It, then, takes each potential alignment, and calculates the full alignment. Finally, it uses a

modified version of Smith-Waterman to find the portions of the alignment that fall between fragments, and to find the best forward and backward extensions for the alignment.

RazerS and RazerS3

RazerS [115] and RazerS3 [116] are both seed-and-extend all-mappers. RazerS family is based on q-gram. The q-gram lemma states that two sequences of length n with hamming distance e share at least $t = n + 1 - (e + 1)q$ common substrings of length q (q-gram). The lemma can be applied to edit distance if n is the length of longer sequence.

In order to find the potential match regions of a read in the genome, RazerS uses SWIFT algorithm which is based on the following two observation. Suppose that two sequences share t q-grams. The dot plot of the sequences consists of at least t contiguous diagonals of length q. If the two sequences have edit distance e then there are at most $e+1$ consecutive diagonals covering these t q-grams. In the case of hamming distance, there is a single diagonal that covers the t q-grams.

For any read of length r, each dot plot parallelogram of dimension $r * (e + 1)$ with at least t q-grams contains a potential match. To speed up, instead of counting q-grams for each possible parallelogram separately, it is enough to count them in overlapping $|r| * w$ parallelograms ($w > e + 1$) that have an overlap of e. This guarantees that every $r * (e + 1)$ is contained in a $r * w$ parallelogram. In a linear scan over the reference sequence, the number of common exact q-grams between read and the reference subsequence is counted for each parallelogram. Parallelograms that contain a sufficient number of common exact q-grams are considered as candidate regions for verification.

RazerS3, in addition to SWIFT filtering, is equipped with another method of filtering based on pigeonhole principle. For error e and arbitrary length reads, the minimal q-gram size is selected and then reads are indexed. Then the index is scanned in linear fashion over the genome and potential match locations are verified. This method is used when e is small and SWIFT algorithm is used when e is large.

Bowtie, Bowtie2

Bowtie [63] is a BWT-FM based best-mapper[1]. FM Index is devised for exact-matching. To handle sequencing errors/genetic variations, Bowtie modifies this algorithm by introducing

[1]Bowtie's best mapping mode is 2 to 3 times slower than its default mode.

two extensions: (i) a quality-aware backtracking algorithm that allows mismatches; (ii) and "double indexing", to avoid excessive backtracking. Its criteria for mapping allows a limited number of mismatches and prefers mappings where the sum of the quality values at all mismatched positions is low. Bowtie has a MAQ-like policy where it allows up to 2 errors in seed (first 28 bases) as well as the maximum acceptable quality distance of the overall mapping.

Bowtie conducts a backtracking search to find mappings that satisfy the mapping policy above. The search proceeds similarly to the exact backward search, calculating R_l and R_h ranges for successively longer query suffixes. If the range becomes empty then Bowtie may select an already matched query position and substitute a different base there, introducing a mismatch into the mapping. Then it resumes backward search from just after the substituted position. Bowtie selects only those substitutions that are consistent with the mapping policy and which yield a modified suffix that occurs at least once in the text. If there are multiple candidate substitution positions, then the algorithm greedily selects a position with a minimal quality value.

The algorithm described so far may encounter sequences that cause excessive backtracking. Bowtie reduces excessive backtracking with introduction of "double indexing". Two indices of the genome are created: one containing the BWT of the genome ("forward index"), and the other containing the BWT of the genome with its character sequence reversed (not reverse complemented) called the "mirror index". To illustrate the benefits of such indexing, consider a matching policy that allows one mismatch in the alignment. A valid mapping with one mismatch falls into one of two cases according to which half of the read contains the mismatch. Bowtie proceeds in two phases: In Phase 1, it loads the forward index into memory and invokes the algorithm with the constraint that it may not have a substation in the query's right half. Phase 2 uses the mirror index and invokes the aligner on the reversed query, with the constraint that the aligner may not substitute at positions in the reversed query's right half (the original query's left half). The constraints on backtracking into the right half prevent excessive backtracking, whereas the use of two phases and two indices maintains full sensitivity.

Bowtie uses the first 28 bases on the high-quality end of the read as 'seed'. The seed consists of two halves: the 14 bp on the high-quality end (usually the 5' end) and the 14 bp on the low-quality end, called the 'hi-half' and the 'lo-half', respectively. Assuming the default policy (two mismatches permitted in the seed), a reportable mapping will fall into

one of the following four cases: no mismatches in seed (case 1); no mismatches in hi-half, one or two mismatches in lo-half (case 2); no mismatches in lo-half, one or two mismatches in hi-half (case 3); and one mismatch in hi-half, one mismatch in lo-half (case 4). Bowtie alternates between forward and mirror indices to cover all the cases in three phases.

Note that if one or more exact matches exist for a read, then Bowtie is guaranteed to report one, but if the best match is an inexact one then Bowtie is not guaranteed to find the highest quality alignment.

Unlike Bowtie, Bowtie2 [62] supports gapped alignment. In doing so, Bowtie2 extracts seeds from the read and its reverse complement. It, then, employs FM-index to extract all the ranges related to these seeds and then prioritize them. Finally, it uses an extended version [25] of Smith-Waterman [109] algorithm to to align these ranges to the read.

BWA, BWA-SW

BWA [68] and BWA-SW [69] are BWT-FM based best-mappers. For a given read R, BWA calculates $D(i)$ where $D(i)$ is the lower bound on the number of differences between prefix $R[0, i]$ and the reference sequence. This means that $R[0, i]$ cannot be mapped to the reference with less than $D(i)$ errors. To calculate D, BWA uses BWT of the reverse reference sequence (not complemented) to test if a substring of R is also substring of the reference. It finds the maximal match for the prefix of R. If there are remaining bases then it tries to find a maximal match for the remaining bases and so on and so forth. Every time BWA resets the maximal search, it introduces an error in D. To illustrate this, suppose the reference is $GOOGOL$ and the query is LOL. Obviously L is the maximal prefix substring of query that can be found in reference because LO does not exists in reference, thus $D(0) = 0$ and $D(1) = 1$, now BWA resets and looks for maximal prefix match of OL, OL exists in reference and thus $D(2) = 1$.

BWA uses a recursive algorithm to search for the suffix array intervals of substrings of the reference that match the query string R with no more than e differences (mismatches or gaps). It uses D to reduce the search space.

GEM

GEM [80] is a BWT-FM based all-mapper. GEM employs a filtration-based approach to approximate string matching: all relevant candidate matches are extracted from a FM index

by suitable pigeonhole-like rules and refined by dynamic programming [92] in bit-compressed representation.

Partitioning the reads into equally-sized segments may not provide the results efficiently. Some of the segments may yield to many candidate matches that need to be verified, thus leading to inefficient alignment.

GEM picks a threshold t, which describes the maximum number of candidates that should be considered for each filtering segment. For a given read, GEM scans the read backward (from right to left) with FM-index adding one character more to the current region each time and calculating the number of candidates in the reference that correspond exactly to the string being formed. Any time the number of rows (candidates) falls below the threshold, it starts a new region.

Note that by definition such a procedure guarantees that the number of candidates to be considered per filtering segment will always be less than the threshold. However, the method does not give any guarantee about the number of regions that will be identified. This eliminates any mathematical guarantees for exhaustive mapping scheme by enforcing pigeonhole-like rules. In practice, for vast majority of the reads this method works well.

SOAP2

SOAP2 is a BWT-FM based best-mapper. SOAP constructs a hash table to accelerate search for the location of a read in BWT reference index. Because of the hash, very few search interactions are sufficient to identify the exact location inside the block. For inexact (both mismatch and indel) alignments, it partitions the read into segments. To allow one mismatch, a read is partitioned into two fragments. Similarly, it partitions a read into three fragments to search for hits that allow two mismatches.

2.2 Sequence Compression

2.2.1 HTS data format

Typically, high throughput sequencing data sequencing data consists of three parts: Read name, read sequence and quality scores. Read name is a unique string that specifies an HTS read. There are a few naming conventions for this part of the data but there is not any universal standard. Read sequence is the nucleotide bases. Quality scores are the

base-calling error probabilities. There is one quality value per sequence base.

2.2.2 Popular Encodings: Huffman, Golomb, Gamma, Delta and Arithmetic Coding

Huffman Coding

Huffman coding [52] is a lossless data compression method. Huffman coding is used when the frequencies of the symbols are known. Huffman coding generates "prefix free code" meaning that the code word for a symbol cannot be prefix of any other symbol. To generate the huffman code, a binary tree is built as follows. Create a leaf node for each symbol and add it to the priority queue. Remove the two nodes with highest priority (smallest frequencies) from the queue and create an internal node with these two nodes as children with frequency equal to sum of the frequencies of these two nodes. Add this new node to the priority queue. Continue these steps until there is one node left in the queue. Assigning the code words are simple as marking the left edges with 0 and right edges with 1. A code word for a symbol is the concatenation of the edge symbols from the root to the corresponding leaf.

2.2.3 Golomb Coding

Golomb coding [37] is a lossless data compression method. It is suitable when occurrence of small values in the input are more probable than larger values. Golomb has a tunable parameter M which divides the input value N into two parts: $q = N/M$ and $r = N$ modulo M. Golomb encodes N as follows: it encodes the q as unary coding (outputs q 1s followed by one 0) and encodes r as truncated binary coding (Let $b = \lceil \log_2 M \rceil$, if $r < 2^b - M$, output r in binary representation using $b - 1$ bits, otherwise output $2^b - M + r$ in binary using b bits).

Gamma Coding

Gamma coding [22] is a lossless data compression method. It is used when the largest value is not known ahead of time and the small values are much more frequent. Gamma code is pretty simple. To encode N in gamma coding, output $\lceil \log_2 N \rceil - 1$ 0s at the beginning and append the binary representation of N to it.

Delta Coding

Delta coding is for storing data in the form of differences between sequential data rather than absolute values. For example, instead of storing 14043, 14045, 14050, we can store 14043, +2, +5.

Arithmetic Coding

Arithmetic encoding [99, 118, 103] is a lossless entropy encoding scheme. The main idea is to represent the whole input sequence as a fractional number n, where $0 \leq n \leq 1$. For each input symbol in the input, arithmetic encoder needs to know a probability assigned to that symbol in order to successfully create the output. In order for data to be decoded, both encoder and decoder have to use same probability distributions for the input symbols. The technique of assigning probabilities to the symbols is usually referred as *data modelling*.

Performance of arithmetic encoder directly depends on the underlying data model. In fact, it is well known result that arithmetic encoder can achieve optimal compression performance (i.e. the compressed size is close to the entropy of the input data) with appropriate model. Unfortunately, modelling problem is not uniform, since different data sources usually have different optimal models. The most commonly used models are simple order-n models, where for calculating the probability of the input symbol p_i, we also take into account the last n symbols which occurred before p_i, namely $p_{i-1}, \ldots, p_{i-n}$. These symbols are also known as a *context* of the symbol p_i. These models can be either stationary or adaptive. For stationary models, probabilities of input symbols are calculated and known in advance of the encoding process, while in the case of adaptive models, probabilities are constantly being updated as the new symbols arrive to the coder.

2.2.4 Existing Methods

CRAM

CRAM [50] is a reference based lossless/lossy compression method. It requires to have the mapping locations for the reads. The mappings should be sorted by their genomic location. For every read, CRAM stores the starting position of the read with respect to the reference genome, the strand and a flag that shows if the read is mapped perfectly. Positions of the reads are encoded with Delta coding and the resulting value is stored as Golomb code.

Strand and match flags require one bit each. In the case of a non-perfect match, CRAM stores a list of variations. Every variation is stored as its position on the read, the variation type (substitution, insertion, deletion), and additional information (the base change in the case of a substitution, the inserted bases, or the length of the deletion). The positions of the variations are Delta and Golomb codes. Type of variation requires 1 or 2 bits. Positions of the variations are Delta and Golomb encoded. The variation type (substitution, insertion, or deletion) is encoded in 1 or 2 bits. Given the reference base, any substitution to A, C, G, T, or N (other than the reference base) can be encoded in 2 bits. Inserted bases are encoded in 2 or 3 bits. Length of deletions are encoded with Gamma coding. If the read length is variable in the input, they are encoded with Huffman coding. For unmapped reads, CRAM assembles them to contigs and then maps the unmapped reads back to this new references and repeats the same procedure explained above. For quality scores, CRAM only keeps the quality scores for the bases that show variations. These quality scores are then compressed with Huffman coding.

SlimGene

SlimGene [61] is a reference based lossy/lossless compression method. Similar to CRAM, SlimGene requires the mapping location for the reads. SlimeGene compresses the read sequence and quality scores but not the identifiers (names) of the reads. These mappings should be sorted with respect to their genomic loci. To compress the starting position of the reads, SlimGene uses a binary array POS. This means that for each position on the genome, SlimGene uses 1 bit. $POS[i]$ is 1 if at least one read maps to the position i. For all the reads that maps to the position i, SlimGene keeps a refinement vector. For each read, the refinement vector consists of three bits (More copies, Strand and Flawless) and an Offset array in case of any errors. If there are multiple reads map to the same position, "More copies" bit is set to 1 except for the last read on the same genomic location. "Strand" bit is set to 1 if it maps to forward strand and 0 otherwise. "Flawless" bit is set to 1 if it is a perfect match. In the case that "Flawless" is set to 0, Offset array is also added to the refinement vector which shows both the differential error location (offset) in the read and type of the error (Opcode) . Usually the error location is calculated from the right hand side of the read.

ReCoil

ReCoil [122] is a lossless read sequence compression method. It does not compress quality scores and read names from HTS data. ReCoil reorders the read sequences in order to increase the similarity. ReCoil constructs a similarity graph between reads that is an undirected weighted graph. In the similarity graph, for each read there is a node and there is an edge between two nodes if the two reads share at least one k-mer. The weight of the edges is equal to the number of k-mers shared between two reads.

Building such a graph for HTS reads requires huge amount of memory which is not practical. ReCoil constructs this graph in external memory. To do so, it gets all the k-mers from a read and stores the pair (k-mer, read_id) on the disk. It (externally) sorts them based on the k-mer value. After this step, all the reads that share the same k-mer fall in the same range. For each k-mer, ReCoil gets all the read_ids and adds the anchors (a, b) to another file (a and b are read_ids of the two reads). In the next step, it (externally) sorts them based on the anchors. At the end of this step, similarity graph is built.

Then, ReCoil (externally) calculates the Maximum Spanning Tree (MST) of the similarity graph. This tree catches the highest similarity between the reads and is unrooted.

Finally, ReCoil selects an arbitrary node in the MST as a root and outputs the the corresponding read. Then, it traverses the MST using BFS and encodes each node's read using maximal exact matches (MEMs) between that node's read and its parent's read. MEMs are calculated by using a modified version of Smith-Waterman algorithm to calculate the edit transcript. ReCoil can use a general purpose method to further improve the compression.

BEETL

BEETL [18] is a lossless read sequence compression method. BEETL constructs a modified Burrows Wheeler string on a collection of reads. BEETL passes this modified version to a general purpose Burrows Wheeler based compression method (i.e. bzip2, 7zip) for compression. Note that the general purpose Burrows Wheeler based compression methods always work with a fixed size window. Thus, doing a BWT on the full collection will increase the locality.

To construct the BWT on a collection of read sequences: (i) append $\$_1, \$_2, \ldots \$_n$ to the end of the reads (one symbol per read); (ii) generate all the circular shifts of all the reads. (iii) sort them lexicographically. BWT of the collection will be the last column from this

sorted list. Note that $\$_1, \$_2, \ldots, \$_n$ do not exists in the alphabet and are lexicographically smaller than any symbol in the alphabet. BEETL constructs this BWT on the collection externally since for HTS reads, it requires more than $100G$.

To understand the BEETL, it would be nice to mention that BWT of a string can create easily compressible data. Consider transforming a long English text frequently containing the word "the". Sorting the rotations of this text will often group rotations starting with "he " together, and the last character of that rotation (which is also the character before the "he ") will usually be "t", so the result of this transform would contain a number of "t" characters along with the perhaps less-common exceptions (such as if it contains "Brahe ") mixed in. Consider the range "he" in the suffix array, on the BWT of a single string you cannot swap the rows to make the "t"s follow each other because the suffixes will never be the same. In BWT of a collection when the order of the string does not matter, you can change the order of the these suffixes if they are the same.

BEETL constructs another array named SAP (Same As Previous). $SAP[i]$ is 1 for the suffix i if it is equal to the previous suffix without considering the ending character. BEETL, then, sorts the BWT of a range that has the same suffix. This procedure is equivalent to the reverse lexicographical sorting of the reads.

G-SQZ

G-SQZ [111] is a Huffman coding-based compression method. It compresses data without altering the relative order. G-SQZ scans the data and calculates the frequencies per pair of *(base, quality)*. It generates the unique Huffman code per pair. It, then, encodes the pairs and writes them to a file. For meta information (read names), it compresses them with Delta encoding meaning that meta-characters ('@', '+', '>', '_', ':', etc.) are stored only once and the differences between successive read names are stored. The compressed file consists of a fixed-length header followed by a sequence of blocks, one block per read.

DSRC

DSRC [19] is a lossless compression method. It divides the reads into blocks and divides each block into three streams and compresses them independently. DSRC uses heuristics to compress the read names. It uses different delimiters to find the segments in the read names and then analyzes these segments. If it is fixed (i.e. instrument name), then it stores them

once. If it is numeric, depending on the distance of the consecutive numbers, it either keeps them packed or compresses them with Delta coding. DSRC uses two different methods to compress the read sequences: (i) DSRC packs every four bases into one byte and then uses Huffman coding to compress them. This method is fast but has a lower compression ratio. (ii) DSRC uses an LZ like compression method on the packed reads. It compresses the uncompressed portion with a backward pointer. DSRC compresses the quality scores using Huffman Coding.

Quip, Fastqz and Fqzcomp

Fastqz, Fqzcomp [12] and Quip [54] are lossless compression methods based on context modelling and arithmetic coding. Context modelling is just providing the probabilities for arithmetic coder. Similar to DSRC and G-SQZ, read names are compressed through Delta coding and AC. For read sequence compression, Quip uses order-12 AC, fqzcomp can use a configurable k-order AC (order-7 seems to work best to find sequence biases and motifs) and fastqz uses 6 models (order-0 to order-5). For quality score compression, Quip uses AC order-3, fqzcomp uses order-2 AC and fastqz uses a mixture of three different models.

2.3 Conclusion

In this chapter we presented an overview of the existing tools for sequence mapping and sequence compression. Sequence mapping tools can be classified into two categories based on indexing method they use: (i) hash based and (ii) BWT-FM based tools. BWT-FM based methods provide a fast way to map a collection of HTS reads to the reference genome, but they acquire this speed by sacrificing sensitivity using ad-hoc heuristics. On the other hand, hash-based methods can reach full sensitivity if designed properly, but they suffer from speed and memory usage. Most of the available hash-based methods use some kind of heuristics to reduce the search space, and by doing so, they lose sensitivity.

Sequence compression methods can be classified into two categories. The first category exploits the similarity between the reads and a reference genome while the second category exploits the similarity between the reads themselves. The first category provides an efficient way to compress the HTS reads, but they suffer from availibity of the reference genome and time required for mapping HTS reads to the reference genome. The issue with the second category is that finding a competitive model can be very complicated.

Chapter 3

A Cache-Oblivious Algorithm for Mapping

As mentioned in Chapter 1, High Throughput Sequencing (HTS) technologies produce unprecedented amounts of data that demands very fast, efficient, and accurate read mapping algorithms. Although improvements in CPU speeds have been helping to improve the performance of such algorithms, the time they spend for accessing the memory is increasingly dominating their running time and providing a significant bottleneck. New and popular read mapping tools index sequence data in main memory to obtain good performance, but the (1) data structures they use are typically not optimized for cache performance; as a result they suffer from frequent cache misses and the costly access to main memory. Furthermore (2) these "memory efficient" data structures are capable of handling only short read lengths and at most 2 to 3 mismatches or indels. In addition, (3) for purposes of structural variation discovery, it is of utmost importance that such mapping algorithms find *all* mapping locations of a given read on the genome - these data structures can not provide such guarantees. As a result, mapping longer reads with higher error rates to all locations will provide a significant computational bottleneck, whose resolution will be essential to the full realization of personalized genomics. To address this need, many mapping algorithms [71, 75, 43, 100, 5, 44, 72, 115, 116, 26, 120, 38, 68, 69, 63, 62, 73, 80] have options to report read multiplicities, however most of this tools do not return the underlying sequence variation.

"Cache obliviousness" is a new paradigm for massive database search, whose potential

contributions to the efficiency of the short read mapping algorithms have not yet been explored. A cache oblivious algorithm utilizes the available cache hierarchy in a systematic manner so as to reduce the number of cache misses, and thus improves the performance without any specific knowledge of the existing cache sizes or structure. A standard example of a cache oblivious algorithm, is one that compares each element of a given list L_1 with every element of an other L_2. A naïve algorithm for this task may use two nested loops to compare each element of L_1 with all of the elements of L_2, requiring a total of $O(|L_1| \cdot |L_2|)$ comparisons. A cache oblivious algorithm for the same task, partitions the two lists recursively until the subproblems can fit in the cache hierarchy, and compare the sublists with each other. Although the number of comparisons stay the same, the order in which they are performed will mathematically guarantee that the number of cache misses will be minimized. Because all available short read mapping tools spend a significant amount of execution time handling cache misses, the cache obliviousness paradigm has a potential to improve their performance drastically.

In this Chapter, we introduce mrsFAST, a cache oblivious short read mapping algorithm. mrsFAST rapidly finds all mapping locations of a collection of short reads in the reference genome through indexing both the reference genome and the short read collection and performing a simple all-to-all list comparison via the cache oblivious algorithm sketched above.

mrsFAST is guaranteed to report *all* possible read mappings and the underlying sequence variation within a specified number of mismatches and indels, for the purposes of discovering segmental duplications and estimating absolute copy numbers of duplicated genomic segments [5] by read depth and structural variation by end-sequence placements [46]. Although the false discovery rate may be higher for this approach, application of mrsFAST has the ability to sample structural variation more comprehensively and identify many variants missed by other approaches.

mrsFAST is a seed-and-extend algorithm. It works by first placing a k-mer (seed) from a read by interrogating the index (in the form of a hash table for all k-mers and their respective loci) of the reference genome, and then extending them by allowing at most a user specific number of mismatches or indels. During the execution of the algorithm, the operating system copies the information related to the seed locations from the main memory to the much faster levels of cache memory, and the extension step is performed using the information stored in the cache. In a naïve execution of such a seed-and-extend algorithm,

the seed mapping locations to be compared to the read would be streamed through the cache. Since the capacity of the cache is very limited, before such read locations can be used for another read, they will be overwritten. mrsFAST reorders the comparisons to be made in order to ensure that the cache memory is split evenly between reads that share a k-mer and seed mapping locations. This facilitates multiple comparison of a seed mapping location while it is resident in the cache.

We have compared mrsFAST with popular mapping tools available for the purpose of finding all the mapping locations of reads on the reference genome within a user defined number of mismatches. Our tests indicate that our method is not only significantly faster but also more sensitive than these methods. Furthermore, the mappings provided by mrsFAST, when used in combination with state of the art structural variation detection algorithms like VariationHunter [46] can capture deletions which can not be detected by methods that rely on best mapping locations provided by MAQ, BWA or Bowtie.

The source code for mrsFAST is available at `http://mrsfast.sourceforge.net/`

3.1 Methods

Given a read from a high throughput sequenced donor genome, a mapping algorithm aims to find locations on the reference genome such that the read can be aligned exactly, or within a small number of mismatches (or indels). Here, we introduce mrsFAST (Micro Read-Substitutions only- Fast Alignment and Search Tool) which finds *all* mapping locations of each read of a given length r on the reference genome within a user specified, e mismatches. mrFAST-CO extends mrsFAST in a way that it finds all mapping locations of a given read on the reference genome within e mismatches and indels. mrsFAST is a seed-and-extend type algorithm: it creates a collision free hash table for each length-$\lfloor r/(e+1) \rfloor$ substring of the reference genome and then map the reads to the genome using this hash table. A key contribution of mrsFAST is that it introduces the cache obliviousness paradigm to genome mapping - and, in fact, to computational genomics, through which it achieves superior running time in comparison to popular mapping algorithms.

The description below focuses on the data structure used by mrsFAST . mrFAST-CO data structure is very similar to that of mrsFAST so rather than describing it separately we only note its differences when necessary.

3.1.1 Indexing the Reference Genome

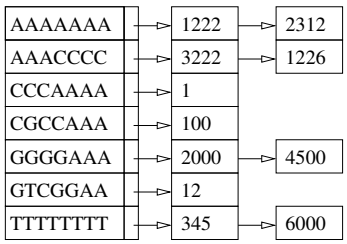

Figure 3.1: Excerpts from a conceptual reference genome index GI.

mrsFAST creates an index array (see Figure 3.1) for the reference genome, denoted GI (Genome Index), in a way that for any *unique* subsequence of size $k = \lfloor r/(e+1) \rfloor$ of the reference genome there exists an entry in GI. The j^{th} item of GI, namely $GI[j]$, is a 2-tuple (s, L), where s is a subsequence of size k from the genome and L is a list of all positions of the genome starting with this subsequence, and are respectively denoted as $GI[j].s$ and $GI[j].L$. The items $GI[j]$ are maintained in GI in lexicographically sorted order with respect to their subsequences $GI[j].s$. If the length of the reference genome sequence is n, an upper bound for the size of GI would be $O(n \cdot k)$; however, because of the highly repetitive nature of genome sequences, the typical size of GI is much smaller in practice.

3.1.2 Indexing the Donor Genome

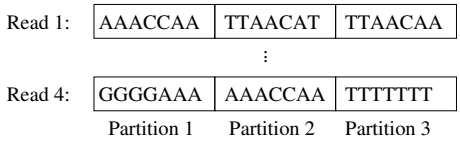

Figure 3.2: Sample read partitioning for $e = 2$: each read is partitioned into 3 equal length blocks

Given a collection R, of length r reads from the donor genome (the number of reads in R denoted by $|R|$) we partition every read to $e + 1$ non-overlapping blocks of length $k = \lfloor r/(e+1) \rfloor$ each. The *pigeon hole principle* guarantees that if a read is mapped to a specific location in the reference genome with at most e mismatches (i.e. within hamming distance e), then at least one of these blocks should map to the reference genome location with no mismatches. We create an index array, RI, as shown in Figure 3.3 for these

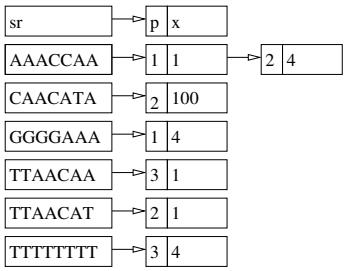

Figure 3.3: Excerpts from a conceptual donor genome (i.e. read) index RI - for the reads given in Figure 3.1.

$(e+1)|R|$ blocks, such that for any specific block from a read, there is a corresponding entry in RI. Each entry of RI, denoted $RI[i]$ is a 2-tuple (sr, PR) again, where sr is the sequence corresponding the block, and PR is a list of read ids that include sr as a block and the specific block location of sr in that read, respectively denoted $RI[i].sr$ and $RI[i].PR$. RI again maintains entries $RI[i]$ in lexicographical order with respect to $RI[i].sr$.

3.1.3 Search

mrsFAST compares GI and RI to find the locations of the reference genome where a read from the donor genome can be mapped with at most e mismatches. For each block of the read, mrsFAST first finds the locations of the reference genome that have a matching (i.e. identical) subsequence. Among all such locations, it then reports the ones whose Hamming Distance to (i.e. the number of mismatches with) the read is at most e. The order of comparison is as follows: Each $GI[i]$ gets to be compared to only $RI[j]$ for which $GI[i].s = RI[j].sr$ through a simple loop that scans the lexicographically sorted arrays GI and RI from left to right. Given an item $RI[i]$, there is indeed an item $GI[j]$ such that $GI[i].s = RI[j].sr$ then all entries in the list $GI[i].L$ is a candidate mapping location for each read entry in $RI[j].PR$; thus the entire list $GI[i].L$ needs to be compared to $RI[j].PR$.

The novelty of mrsFAST is that this "all-to-all" list comparison is performed through the recursive divide-and-conquer strategy, which guarantees cache obliviousness; i.e. the number of cache misses in any level of the cache hierarchy during this very costly all-to-all comparison step is mathematically guaranteed to be minimum [33]. As a result, mrsFAST not only guarantees to find all locations of the reference genome that can be mapped to by a particular read within e mismatches , but it guarantees to find them in minimum possible time by *any* seed-and-extend algorithm.

Lemma 1 *The number of cache misses by mrsFAST, at any level of the cache hierarchy, is optimal among all family of seed-and-extend algorithms for mapping, within a factor of* 2 *in the worst case.*

Proof: Consider a cache level that can accommodate a total of γ items from the lists L_1 and L_2 which need to be compared - this is the essence of a seed-and-extend type algorithm. Note that a seed-and-extend type algorithm will need to either partition the reads from the donor reads to non-overlapping blocks of length m and extract all substrings of length m from the reference genome, or vice versa - in theory it is possible to do a combination of the two. Independent of the option chosen, the number of comparisons that will need to be performed will be the product of the lengths of the two genome sequences divided by m.

Now consider a "hypothetical" cache optimal (as opposed to cache oblivious) seed-and-extend algorithm. Without loss of generality, consider the case in which the lists are of equal length, ℓ; if they are not one can partition the longer one into smaller sublists, each of length ℓ, which can be compared to the shorter list independently. Starting with an empty cache, this hypothetical cache optimal algorithm will bringing in some α items from list L_1 and β items from list L_2 to the cache before any comparison is performed. In fact for any cache optimal algorithm that does not fill up all cache positions at each step of the all-to-all comparison procedure, there is an alternative cache optimal algorithm which works in alternative stages of (i) bringing in some items from L_1 and others from L_2 so as to fill up some cache positions and (ii) do an all-to-all comparison for the items from the two lists. In order to minimize the number of cache misses, such an algorithm necessarily maximizes the number of pairwise comparisons in each execution of step (ii). This necessarily implies that, (1) the cache is always kept full, i.e. $\alpha + \beta = \gamma$, (2) in order to maximize the number of comparisons, $\alpha \cdot \beta$, under the constraint $\alpha + \beta = \gamma$, we have to set $\alpha = \beta = \gamma/2$, and (3) in the execution of step (i), either all the items from L_1 or all the items from L_2 or both need to be fully replaced by new items from the same list(s). Note that the total number of comparisons this cache optimal algorithm will perform is ℓ^2. Due to observation (2) above, at most $\gamma/2 \cdot \gamma/2$ comparisons can be performed in each iteration; thus the total number of iterations will be $(2 \cdot \ell/\gamma)^2$. Due to observation (3) above, at the end of each iteration, at least $\gamma/2$ items will be replaced in the cache; thus the total number of cache misses in all iterations will be at least $2\ell^2/\gamma$.

The cache oblivious all-to-all list comparison algorithm on the other hand, will always

maintain that the number of items from L_1 kept in the cache is equal to that from L_2. In level $h = \lceil \log_2 \ell - \log_2 \gamma \rceil + 1$ of the recursive call for divide-and-conquer method will aim to compare two sublists A and A' from L_1 of length $\lceil \ell/2^h \rceil$ each, to two sublists B and B' from L_2 of the same length. Note that $\gamma/4 < \ell/2^h \leq \beta/2$, thus the sublists to be compared can easily fit in the cache. After the comparison of each pair of sublists, at most $\lceil \ell/2^h \rceil$ items will be replaced in the cache through a correct ordering of the recursive calls. As a result, the total number of cache misses per recursive call in level h will be $4 \cdot \ell/2^h$. Because the total number of recursive calls at this level is $2^{2(h-1)}$, the total number of cache misses will be $2 \cdot \ell \cdot 2^{h-1}$ in level h.

The above calculation simply implies that the ratio between the number of cache misses by the cache oblivious algorithm and the hypothetical "optimal" algorithm is at most $\frac{2\ell 2^{h-1}}{2\ell^2/\gamma}$ (at level h), which is upper bounded by the constant 2 as $2^{h-1} \leq \ell/\gamma$ by definition. Thus, even in the worst case, the cache oblivious algorithm is optimal with respect to the cache misses within a factor of 2. ∎

Note that in order to guarantee that we report each mapping location for a read only once (there are $e + 1$ blocks of a read - each block can "re-discover" a location detected earlier by another block) we make sure that blocks to the left of the current blocks each have at least one mismatch with the potential mapping location in the reference genome. If this constraint is not satisfied, the potential mapping location is no longer considered.

3.2 Additional Features

mrsFAST is designed to handle not only single read mapping but also paired end mapping. For this purpose mrsFAST initially maps each of the ends of every paired end reads independently to each chromosome. Then, it produces the mapping locations of each paired end read by combining the independent mapping locations so as to satisfy all user defined constraints such as the minimum and maximum distance between the ends. Because independent mapping locations with respect to all reads are obtained in location sorted order, these constraints are easily satisfied without the need for using additional data structures.

3.3 Results

We explore the performance and accuracy of mrsFAST in comparison to several popular alternative mapping tools in depth below.

Software Benchmarked: We compared mrsFAST with:

- Bowtie (version 0.10.0) from [63],
- BWA (version 0.5.0) from [68],
- MAQ (version 0.7.1) from [71],
- RazerS (version1.0 20090710) from [115],
- mrFAST-CO (an extended version of mrsFAST that can handle indels - equivalently a doubly indexed, cache oblivious version of mrFAST).

We used the following parameter settings for the above programs:

- Bowtie - error threshold, $v = 2/3$ (for different runs); number of mappings: a (indicating "all").
- BWA - for alignment: error threshold, $n = 2/3/4/6$ (for different runs); number of mappings: N (indicating "all")
  for reporting: the maximum number of locations to be reported for each read, $n =$ 300,000,000. In addition, we also benchmarked BWA with the default setting that report a unique map location.
- MAQ - for alignment: error threshold, $n = 2/3$ (for different runs), number of mappings, $C = 1000$ (which reports all mappings).
- RazerS - error threshold, i=94; sensitivity (set to 100%), rr=100; number of mappings for any read m=100 (it was not possible to set m to a value higher than 100 because of its significant memory requirements)
- mrsFAST and mrFAST-CO: number of mappings for any read, $n = \infty$ (i.e. all mappings to be returned); error threshold, $e = 2/3/4/6$ (for different runs).

We carried out two independent experiments to benchmark the above software tools.

- We performed a general comparison of the speed and accuracy of the above mentioned mapping tools against mrsFAST.
- We calculated the average cache miss per instruction and instructions per cycle (IPC) in mrsFAST and all of the above mapping tools.

Data, Reference Genome and Computing Power. We carried out our experiments on (i) randomly picked 1 million $36 - bp$ reads from an earlier whole genome resequencing study on the NA18507 (Yoruban) genome[10]) as well as (ii) 1 million 100-bp reads from the same genome provided to us by Illumina, which we also rendered into shorter reads of length 50-bp and 75-bp. We used the human genome build 36 (non-repeat masked) as the reference genome in our experiments. All our experiments are performed on a 64 bit Intel(R) Xeon(R) 2G with 4GB of RAM.

Time and Accuracy. We provide the comparison results for all these methods considering the time and accuracy of mapping (to ensure fairness we compared all these methods when their relative parameters were set to find and output all the locations a read can map to). We compare the accuracy of each method by computing number of reads each method was able to map to reference genome with number of mismatches less or equal to 2 for the read size 36bp, 3 for the read size 50bp, 4 for the read size 75bp and 6 for the read size 100bp (BWA also detects some mappings with indels). Note that neither Bowtie nor MAQ is currently capable of handling 4 mismatches.

The speed and sensitivity results are shown in Table 3.1. As can be seen, the number of false negatives for MAQ increases significantly when the read size increases to 50bp. In fact, for that read size MAQ misses almost half of the map locations mrsFAST discovers.

We remind that mrsFAST is *guaranteed* to find *all* map locations within a specified number of mismatches. As a result, the number of mappings reported for mrsFAST is the *correct* number of mappings one should obtain through any algorithm reporting mapping locations with mismatches only.

For read length 36bp, observe that MAQ returns more map locations than mrsFAST: this is due to the fact that MAQ returns a non-trivial number of false positives - i.e. maps reads to loci with more differences than the user defined error threshold. Also observe that BWA generally reports more map locations than mrsFAST. This is due to the list of map locations returned by BWA including (1) many repetitions (the same locus reported

multiple times - sometimes due to several alignment scripts with the same number of errors), (2) sequences involving the symbol `N` (i.e. *aNy nucleic acid*) -BWA assumes that `N` correctly maps to any symbol from the four letter DNA alphabet - these, which constitute about %10 of the mapping locations returned by BWA, are avoided by mrsFAST. In addition, when the read length is 100bp, mrsFAST is up to 3 times faster than BWA even when BWA is used with the default options to report a unique map location, where mrsFAST is used to return all map locations. Finally, note that, because for read length 50bp, MAQ discovers much fewer mapping locations than BWA, it spends less time.

Cache Utilization. For a second set of experiments, we used the *OProfile* tool (which can be found at `http://oprofile.sourceforge.net`) to profile the system performance. OProfile is a system-wide profiler for Linux/Unix systems, capable of profiling all running code at low overhead. We used OProfile to calculate *L2 MPI* (level 2 cache misses per instruction) which is the ratio of L2 cache misses to overall retired instructions (i.e. instructions which are successfully completed). L2 MPI demonstrates the amount of cache misses of a program (lower values indicate better cache performance of the program); we have calculated the amount of L2 MPI for mrsFAST and MAQ: see Table 3.2. In addition, we calculated *IPC* (instruction per cycle) which is the ratio of overall retired instructions (completed) to CPU clock unhalted (i.e. number of CPU clocks outside of halt state). IPC demonstrates the average instructions completed per cycle, thus higher this value indicates better performance. The IPC values for mrsFAST, and MAQ are shown in Table 3.2. Please note that for these experiments we mapped the same set of reads as before (1 million reads) to a portion of human reference genome (profiling of these programs on full human genome would have been very costly and time consuming).

Fewer cache misses and reduced idle cycles obtained by mrsFAST (in comparison to others) results in higher IPC: see Table 3.2. Bowtie and BWA all have similar cache utilization figures, however MAQ has a significantly higher percentage of idle cycles and lower cache utilization.

The Need for mrsFAST and mrFAST-CO for Comprehensive Structural Variation Detection and Copy Number Variation. Our final set of results demonstrate the need for mrsFAST and mrFAST-CO and in general, the necessity of finding all mapping locations of each paired end read, in structural variation detection and CNV discovery. It is now well established that a significant portion of structural variations are observed

Table 3.1: Mapping one million reads of indicated read lengths and within the given number of errors, to the human reference genome HG18 build 36 by indicated algorithms.

Dataset		Software	Time	% of reads	Mappings
Read Length	Error		(hh:mm)	Mapped	Reported (mil)
36	2	Bowtie	5:14	91.65	1,404
		BWA[c]	3:10	92:05	1,581
		MAQ[c]	6:45	90.91	1,609
		RazerS[a]	10:17	91.79	<100
		mrFAST-CO	6:12	92.18	1,486
		mrsFAST	2:00	91.79	1,411
		BWA[b]	0:10	92.05	<1
50bp	3	Bowtie	3:13	92.73	610
		BWA[c]	10:23	93.38	729
		MAQ[c]	10:05	89.25	458
		RazerS[a]	12:17	92.91	<100
		mrFAST-CO	9:21	93.39	663
		mrsFAST	1:55	92.91	613
		BWA[b]	0:15	93.38	<1
75bp	4	Bowtie	NA	NA	NA
		BWA[c]	59:35	90.16	212
		MAQ	NA	NA	NA
		RazerS[a]	12:00	89.35	< 100
		mrFAST-CO	11:32	90.22	193
		mrsFAST	2:00	89.35	177
		BWA[b]	0:25	90.16	<1
100bp	6	Bowtie	NA	NA	NA
		BWA	67:38	87.91	42
		MAQ	NA	NA	NA
		RazerS[a]	25:10	87:27	<100
		mrFAST-CO	17:54	88.55	155
		mrsFAST	2:49	87.27	138
		BWA[b]	7:04	87.91	<1

[a] Because of RazerS's high memory requirement, we could not run it for read multiplicities >100; [b] BWA's default settings; [c] Total number of mapping locations is higher for MAQ or BWA than for mrsFAST or mrFAST-CO because MAQ often returns mapping locations with an error rate higher than the user-specified rate and BWA returns certain mapping locations multiple times.

Table 3.2: L2 MPI and IPC

Software	L2 MPI	IPC
Bowtie	0.0016	0.94
BWA	0.0016	0.93
MAQ	0.0060	0.56
mrsFAST	0.0008	1.24

in repeat regions of the human genome [87]. As a result, structural variation detection methods focusing on unique (non-repeat) regions (e.g. [10]) fail to detect almost half of the deletions (43 out of 92) and the majority of the inversions validated by fosmid studies [56] on NA18507 [10]. Algorithms that focus on the repeat regions such as VariationHunter have been shown to detect some of these deletions (an additional 9 strongly supported and 6 reasonably well supported deletions, on top of the ones predicted by unique mappers in NA18507 reported in [10]) as well as inversions [46]. The remaining question is whether it is possible to capture these deletions (in the case of NA18507, $9 + 6 = 15$ deletions) as well as other structural variations occurring in repeat regions by considering the best mapping location (or the best few locations) provided by available methods such as BWA or Bowtie - we omit MAQ from this comparison as its results are consistently inferior to BWA results. Equivalently, we would like to check whether it is necessary to find all (or at least the majority) of the mapping locations of each read (which are provided by mrsFAST faster and more accurately than any competing software) for reliable structural variation prediction.

Among the 15 additional fosmid validated deletions found using VariationHunter algorithm (and not by [10]) through the mappings provided by mrFAST [46, 5] in NA18507, we found 5 deletions, which can only be detected when all the mappings of the paired-end reads are considered. The "best" (or first) mapping option for paired-end reads with BWA or Bowtie do not map the majority of the reads supporting these deletions to the correct loci - failing to provide the necessary support to make a call for these deletions. As a result, none of these 5 (out of 15) deletions, given in table 3.3, can be discovered through unique mappings.

Table 3.3: 5 fosmid validated deletions in NA18507 which can only be found using multiple mappings

Deletion Loci
chr10:26723208-26737813
chr7:6846864-6901718
chr2:89876618-89935540
chr14:105395536-106097514
chr2:89238825-89279544

3.4 Conclusion

Applications employing high throughput sequencing technologies, typically need to find the mapping locations of all short reads on the reference genome. A fast method with high sensitivity (which is capable of finding all the locations a read can map to with a small number of mismatches) is highly desirable. In this chapter we introduced a novel mapping tool, mrsFAST, that guarantees 100% sensitivity, while ensuring optimal performance among the seed-and-extend type algorithms through the use of the cache obliviousness paradigm. Our experiments show that minimizing the number of cache misses through better memory management can have significant results in the efficiency of mapping methods; see Tables 3.1 and 3.2. As a result, mrsFAST is faster than all competitive methods, especially for longer read lengths, as shown in table 3.1. As read length and throughput increase, algorithmic advances such as mrsFAST will be critical in allowing 1000s of genomes to be analyzed within individual labs.

Chapter 4

SNP-aware Mapping

Available mapping tools require a (typically user defined) upper bound on the number of "errors" it can tolerate per read mapping, and treat real variants and sequencing errors identically - this reduces the mappability of a significant number of reads substantially. A mapper capable of distinguishing real variants from sequencing errors, will be able to map more reads to the reference, effectively providing an increased accuracy and sensitivity. Unfortunately, as each read is mapped independently from the others, and the genomic variants are detected only after the mapping process is complete, real variants cannot be known *a priori*. However, many of the 3 to 4.5 million SNPs in a human genome (in comparison to a reference genome) are shared among individual genomes [2], and have been collected and indexed in the dbSNP database. Therefore, a read mapper which utilizes the common SNP information in dbSNP (or any other genomic variation databases) can improve the signal-to-noise ratio in alignments.

In this chapter, we introduce a new SNP-aware read mapper developed for the Illumina platform, that we call mrsFAST-Ultra, which improves the (i) mappability, (ii) mapping accuracy, and (iii) sensitivity by tolerating common, previously reported sequence variants and distinguishing them from likely sequencing errors. Given a user defined error threshold, mrsFAST-Ultra reduces the number of reads that could NOT be mapped by any available mapper by 19%.

mrsFAST-Ultra achieves this while providing full sensitivity, i.e. it guarantees to find all mapping loci of each read within a user defined error threshold. As mentioned earlier, this feature is essential for accurate structural variant detection techniques (e.g. VariationHunter [46, 49]). As a result, mrsFAST-Ultra has a significantly higher sensitivity compared to

Bowtie2 and BWA (the latest version) in the "all mapping mode" where mrsFAST-Ultra reports at least 30 times more mappings per read.

mrsFAST-Ultra introduces several additional improvements over mrsFAST (See Chapter 3), such as (i) requiring a substantially smaller reference genome index file (which also improves its cache performance), (ii) introducing new filters to improve search space, and (iii) supporting multi-threading. More specifically, mrsFAST-Ultra improves on the storage requirement of the mrsFAST, the first cache-oblivious HTS read mapper, by a factor of 10. The index size was one of the limiting factors of the original mrsFAST in large scale sequencing projects. As mentioned above, the compactness of mrsFAST-Ultra's index structure also reduces the overall number of CPU operations and the I/O needs, resulting in a factor of 4.5 improvement in the running time in comparison to the original mrsFAST.

Finally, mrsFAST-Ultra introduces new features such as (1) the ability to retrieve the single best mapping loci, (2) the ability to retrieve all reads which map to at most (a user defined) k unique loci (within a user defined number of mismatches), and (3) automatic parallelization if multiple cores are available in the computing environment.

4.1 Methods

Similar to mrsFAST, mrsFAST-Ultra is a seed and extend aligner in the sense that it works in two main stages: (i) it builds an index from the reference genome for exact "anchor" matching and (ii) it computes all anchor matchings for each of the reads in the reference genome through the index, extends each match to both left and right and checks if the overall alignment is within the user defined error threshold.

4.1.1 Compact Indexing of the Reference Genome

In the indexing step, mrsFAST-Ultra slides a window of size $k = l/(e+1)$ (where l is read length and e is the user defined error threshold e) through the reference genome and identifies all occurrences of each k-mer present in the genome. For small values of k, mrsFAST-Ultra's genome index is an array of all possible k-mers in lexicographic order. For each k-mer, the index keeps an array of all locations the k-mer is observed in the reference genome. In case the value of k is prohibitively large, only a prefix of user defined size ℓ (for each k-mer) is used for indexing. For each such ℓ-mer, its locations on the reference genome are then sorted with respect to the $k-\ell$-mers following it. (In fact, for most applications, even

keeping track of all $k - \ell$-mers following a particular ℓ-mer is not necessary: we just hash these $k - \ell$-mers via a simple checksum scheme.)

For further compacting the index, the reference genome itself is first converted to a 3 bit per base encoding. The genome sequence is stored in 8 byte long machine words implying that each machine word contains 21 bases. In addition, the index of the reference genome actually does not keep every occurrence of each k-mer, but rather keeps how many occurrences of each k-mer is present in the genome. Everytime we perform a search in the reference genome, the locations are computed on the fly. This reduces the I/O requirements of mrsFAST-Ultra significantly. One may think that such a set up would increase the overall running time of the search step but the savings from I/O reduction significantly offsets the cost of recalculating the k-mer locations on the fly. Overall, the storage requirement of the index we construct for the reference genome is 2GB, including the reference genome sequence itself. This represents a 10 fold improvement in the index storage requirement of the original mrsFAST.

4.1.2 Search.

In this step, mrsFAST-Ultra processes the reads from an input HTS data set and computes "all" locations on the reference genome that can be aligned to each read within the user-defined error threshold e. mrsFAST-Ultra is a fully sensitive aligner meaning that it guarantees to find and report all mapping locations of a given read within e mismatches. mrsFAST-Ultra achieves this by partitioning the read into $e + 1$ non-overlapping fragments of length k for a given error threshold e. Due to the pigeon hole principle, at least one of these fragments should have an exactly matching k-mer of the reference genome in each location the read can be mapped to. The search step then validates whether each location of the reference genome with an exact k-mer match of the read is indeed a mapping location.

In order to perform the search step as fast as possible, mrsFAST-Ultra loads the genome index (see above) to the main memory and computes the locations of each k-mer on-the fly - for significant savings in I/O. For each k-mer, the number of locations in the reference genome is already stored in the index, thus we can preallocate the required memory for each array that keeps the locations of a given k-mer. Once this extended reference genome index is set up in the main memory, the remaining memory is allocated for the reads. At each subsequent stage, mrsFAST-Ultra retrieves sufficiently many (unprocessed) reads that can fit in the main memory and searches them in the reference genome simultaneously.

(Alternatively, the user can specify an upper bound on the memory usage.) These reads are also indexed with respect to the $e + 1$ non-overlapping fragments of size k it extracts from each read. Basically, for each possible fragment of length k, the read index keeps the read ID, the fragment number and the direction the fragment is observed in the read. Once the read index is set, it is compared to the reference genome index, in a divide and conquer fashion as per mrsFAST, in order to achieve cache obliviousness.

Because mrsFAST-Ultra aims to be fully sensitive, it needs to verify whether each reference genome location and each corresponding read that have the same k-mer have indeed an alignment within the user defined error tolerance. Note that, the value of k, set to $l/(e+1)$ can be too big for creating an index that has an entry for every possible k-mer from the 4 letter DNA alphabet. Thus the primary indexing is performed on a prefix of length $\ell = 12$ for each k-mer and all locations/reads that share this prefix are further sorted according to the $k-\ell$-mer succeeding this prefix. This is achieved by hashing the $k-\ell$-mer through a simple checksum scheme. As a result, the divide-and-conquer comparison of reference genome locations and reads is performed on those entries that have the same ℓ-mer and the same checksum value for the succeeding $k-\ell$-mer. The comparison for each genomic location and a read involves the calculation of the Hamming distance between the read and the k-mer location in the genome, extended by the appropriate length towards left and right. Before calculating the Hamming distance, mrsFAST-Ultra applies another filter that compares the number of As, Cs, Gs and Ts in the read and the genomic locus; if the total number of symbol differences is more than $2e$, then we do not need to compute the Hamming distance explicitly as it will be at least $e+1$ - above the error threshold. In comparison to the original mrsFAST, this search strategy reduces the number of Hamming distance calculations, the main bottleneck for the search step, by a factor of 5. When combined with reduced I/O (due to compact index representation), a two stage index (for both the reference genome and the reads), and the introduction of new filters, this implies a 4.5 factor reduction in the overall running time of search.

4.1.3 SNP awareness

The user has the option of setting mrsFAST-Ultra to tolerate known SNP locations in the mappings: i.e. in this mode, SNPs in an alignment location simply do not contribute to the error count in the Hamming distance computation. For that, mrsFAST-Ultra parses dbSNP and generates a compact structure that it uses for mapping. Although conceptually simple,

this feature is highly desired by users as it significantly reduces the number of reads that can not be mapped to anywhere in the reference genome. In this mode mrsFAST-Ultra reports the number of SNPs in addition to the number of mismatches per each mapping location.

4.2 Additional Features

Limited mapping. mrsFAST-Ultra provides the user the option of returning a single best mapping locus per read - which it performs much faster than computing all mapping loci. As per BWA, Bowtie2, SRmapper and others, a best mapping location (on the reference genome) is considered to be one which has the smallest number of differences with the read. In addition, mrsFAST-Ultra has the option to return only mapping loci of reads which map to at most n locations within the user defined error threshold. These features help the users to control the mapping multiplicity - which can grow prohibitively for further downstream analysis.

Parallelization. mrsFAST-Ultra is designed to utilize the parallelism offered by contemporary multicore architectures. The mapping task is simply partitioned into independent threads each of which is executed by a single core. For efficiency purposes, the only locks used by the threads are for allocating memory and I/O.

4.3 Results

We report on experiments we performed on a single PC, equipped with an Intel(R) Xeon(R) CPU with 4 cores and 12GB of RAM. We benchmarked a number of read mapping software with parameters set as below - unless otherwise stated.

- mrsFAST v2.5.0.4 (-e 6, for error threshold)
- mrsFAST-Ultra v3.0.0 (-e 6, for error threshold, –threads 1 for using a single CPU)
- BWA v0.6.2 from [69] (-n 6 for error threshold; -N for disabling iterative search and reporting all mapping locations where required)
- Bowtie2 v2.0.2 from [62] (-k 100 for reporting up to 100 mappings for each read, -a for reporting all mapping locations where required [1])

[1] It was impractical to run Bowtie2 for all mapping location

- GEM v1.367.beta from [80] (-m 6 for error threshold 6; -d for reporting all mapping locations where required)
- RazerS3 v3.1.1 from [116](-i 94, provides %94 similarity for allowing 6 errors in reads of length 100, -rr 100 for full sensitivity)
- GSNAP 2013-01-23 [120] release (-m 6 for error threshold 6)
- SRmapper v0.1.5 [38] (-m 6 for error threshold 6)

Our results below are based on mapping 2 million Illumina reads of length 100bp from NA18507 (Sequence Read Archive ID: SRR034939) individual genome to the "hg19" version of the human reference genome. We carried out a few experiments to evaluate the performance of the above software. In the first experiment, we mapped the reads with an error threshold of 6% (i.e. 6bp). Table 4.1 depicts the results of this experiment. As can be seen, mrsFAST-Ultra reports about 308M mapping locations for these reads, which is at least 30-times more than the number of mapping locations reported by any of the competing methods. In the SNP-aware mode where we provided mrsFAST-Ultra with dbSNP32, the percentage of reads which could not be mapped drops by 19% (roughly 3% of all reads).

In the second experiment, we set the appropriate parameters in each method to report 100, 1000 and all mappings locations per read. Table 4.2 shows the running time of all the methods. Although the running time for GEM is better than the other methods, it misses many mapping locations as per BWA and Bowtie2.

In the third experiment, we ran all the tools in the "best mapping" mode with various error thresholds. Although mrsFAST-Ultra is not as fast as some of the other tools, it has higher sensitivity than the others as shown in Table 4.3.

In the final experiment, we compare mrsFAST-ultra and GSNAP in their SNP-tolerant best mapping mode. The results are given in Table 4.4.

Table 4.5 demonstrates the memory footprint of all tools we benchmarked on 2M reads.

Finally, we show the effectiveness of mrsFAST-Ultra filters. In Figure 4.1, we calculated the expected number of the locations that should be verified given varying k-mer values. As expected, when the length of k-mer increases the expected number of locations that should be verified decreases. We also plot the average number of locations verified by mrsFAST-Ultra for various k-mer + checksum length values. As shown in the figure, using checksum

Table 4.1: Mapping 2M reads from NA18507 to hg 19 with e ≤6. BWA and GEM are set to report all mapping locations. Bowtie2 is impractical to run if it is set to report all mappings. Other mappers do not provide such option.

Software	Time	# mappings (millions)	% of reads mapped
mrsFAST	6h 2m	308.302	90.55
mrsFAST-Ultra	1h 18m	308.302	90.55
mrsFAST-Ultra (SNP-aware)[a]	1h 52m	354.691	93.17
BWA	7h 16m	4.133	91.57
Bowtie2[b]	2h 52m	5.130	91.52
GEM	16m	9.078	90.78
RazerS3[c]	10h 17m	10.631	92.00
GSNAP	3h 11m	5.388	86.37
SRmapper[d]	7m	0.32	10.46

[a] Note that the SNP-aware mrsFAST-Ultra employs dbSNP32 for this task; [b] For Bowtie2, we report the time when it is set to return at most 100 mappings per read - without this bound it does not complete the task in 24 hours; [c]Please note that RazerS3 cannot finish the task within 12hrs and we reduced its sensitivity to 99% (i.e. setting parameter -rr to 99); [d] SRmapper crashes on the full human genome. Results are shown only for mapping the reads to chr1.

Table 4.2: Running time for reporting n mapping locations per read.

Software	n=100	n=1000	n=∞
mrsFAST-Ultra	53m	57m	78m
BWA	438m	436m	436m
Bowtie2[a]	172m	NA	NA
GEM	14m	13m	16m
RazerS3	538m	604m	617m
GSNAP	196m	191m	191m
SRmapper[b]	7m	7m	7m

[a] Bowtie2 can not complete the task in 24 hours for n≥1000. [b] SRmapper crashes on full human genome. Results are shown only for mapping the reads to chr1.

Table 4.3: Mapping of 2M reads in the best mapping mode, with an error threshold of 2, 4 and 6. No indels/gaps allowed in any method. We report on both the running time and the percentage of reads mapped.

Software	$e \leq 2$		$e \leq 4$		$e \leq 6$	
	Time	% of reads mapped	Time	% of reads mapped	Time	% reads mapped
mrsFAST-Ultra	12m	80.97	21m	87.63	61m	90.55
BWA	4m	80.97	11m	87.52	18m	90.22
Bowtie2	10m	80.97	10m	87.52	10m	89.77
GEM	4m	80.97	6m	87.18	13m	89.33
RazerS3	13m	80.97	59m	87.63	325m	90.55
GSNAP	156m	77.78	180m	83.70	184m	86.16
SRmapper[a]	3m	7.29	5m	8.89	7m	10.46

[a] SRmapper only running on chr1

Table 4.4: Comparing mrsFAST-Ultra and GSNAP in SNP-tolerant best mapping mode.

Software	Time	% of reads mapped
mrsFAST-Ultra	82m	93.17
GSNAP	207m	85.30

filtration mimics the use of longer k-mer. In the figure we demonstrate the average number of the locations verified after we incorporated the 1-gram filtration method.

Table 4.5: Memory footprint of the tools on 2M reads.

Software	Memory Footprint (GB)
mrsFAST-Ultra	2.5
BWA	3.2
Bowtie2	3.2
GEM	4.1
RazerS3	1.4
GSNAP	4.6
SRmapper[a]	0.2

[a] SRmapper footprint only on chr1

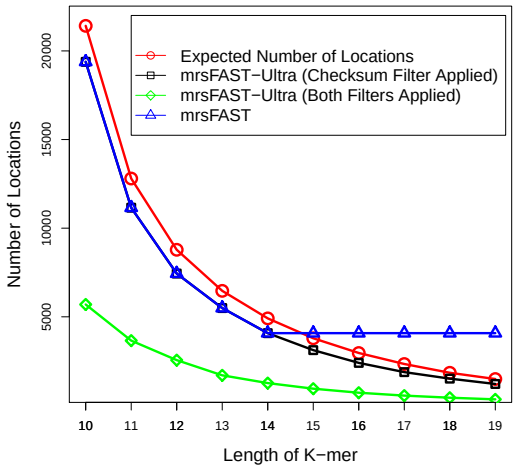

Figure 4.1: Average Number of locations verified per k-mer extracted from each read, as a function of k-mer length.

Chapter 5

Sensitive and Fast Mapping of di-base Reads

As stated in Chapter 1 discovering variation among high throughput sequenced genomes relies on efficient and effective mapping of sequence reads. The speed, sensitivity and accuracy of read mapping are crucial to determining the full spectrum of single nucleotide variants (SNVs) as well as structural variants (SVs) in the donor genomes analyzed.

In this chapter, we describe a hash-based read mapping algorithm named "di-base read fast alignment search tool" (*drFAST*) designed for the di-base encoded color-space reads generated with the SOLiD platform. The main advantage of di-base encoding is increased base call accuracy due to each base being represented by two "colors". This helps in differentiating base calling errors (color-space errors) from real sequence variance, thereby increasing the reliability of detected genomic variants. We show that mapping speed of *drFAST* is higher than other SOLiD-enabled hash-based read mappers BFAST [44] and SHRiMP [100], and comparable and comparable to suffix array-based aligner Bowtie [63]. In addition, *drFAST* was able to map more reads than all the tools we benchmarked. Furthermore, *drFAST* achieves 100% sensitivity if the maximum allowed distance is less than L/k, where L is the sequence length, and k is the length of the k-mers stored in the hash table (k is set to 12 by default). Coupled with its ability to find all map locations within a user-specified distance threshold and it paired-end (PE) mapping capabilities, *drFAST* can be used to characterize segmental duplications [5, 110], and increase sensitivity of structural variation discovery using VariationHunter [46, 49].

The source code for drFAST is available at `http://drfast.sourceforge.net/`

5.1 Methods

For each read sequenced from a donor genome, a mapping algorithm aims to find locations in a "reference genome" where the read can be aligned exactly or within a small number of errors in the form of substitutions or insertions/deletions (indels). *drFAST* is a read mapper designed for color-space reads generated with the SOLiD platform, and finds *"all"* possible map locations for each read of length r in the reference genome within a user-specified e mismatches.

drFAST is a seed-and-extend type algorithm and it builds an index of the reference genome by creating a collision-free hash table for all subsequences of length k (k-mers) of the reference genome. To map the reads, it first partitions each read to $(e+1)$ k-mers and searches for each of these k-mers in the hash table. For each *hit* in the hash table, it then tests if the remainder of the read can be "extended" by aligning to the reference genome starting at the determined hit location.

How exactly this is done is described below: In the following subsections, we provide the detailed description of the algorithm.

5.1.1 Genome transformation

The sequence data produced with the SOLiD platform are in *color-space* format ($S = \{0,1,2,3\}^*$), where the reference genome sequence is in *letter-space* (i.e. $R = \{A,C,G,T\}^*$). Each color encodes two adjacent base pairs in the read, and each base pair is represented by two colors. Transformation of reads from color-space to letter-space before mapping may result in generating incorrect reads where base call errors exist, as depicted in Figure 5.1. To avoid such incorrect decoding of reads, we translate the reference genome to color-space and use this transformed genome to create the index.

5.1.2 Indexing the Reference Genome

drFAST creates a collision-free hash table for all k-mers in the reference genome. Each entry of this index is a 2-tuple $\tau = (s, L)$, where s is a k-mer from the genome ($k = 12$ by default) and L is a list of all positions of the genome starting with this subsequence. The index

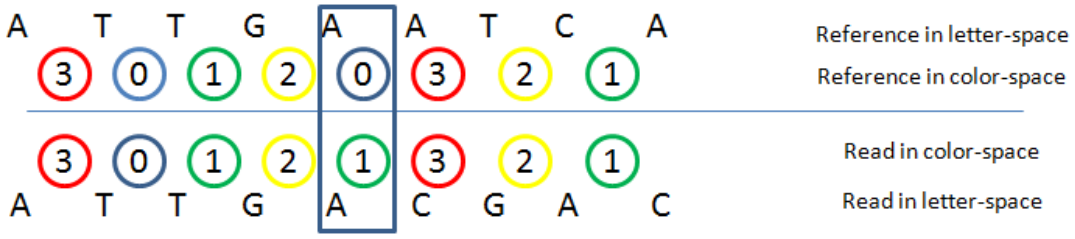

Figure 5.1: Translating the read from color-space to letter-space may result in a new sequence different from the original read if there exists a color-space error.

is maintained in lexicographically sorted order with respect to their subsequences. For a reference genome of length n, the upper bound for the size of its index is $O(n.k)$; but due to the repetitive nature of genome sequences, the index size is smaller in practice.

5.1.3 Indexing the Donor Reads

drFAST partitions each read of length r into $e+1$ non-overlapping blocks of length k where e is the user-specified maximum Hamming distance allowed for mapping. In the case where $k \leq \lfloor r/(e+1) \rfloor$ the pigeon hole principle guarantees that at least one of these blocks maps to the reference genome with no errors. Similar to the indexing described in Section 5.1.2, *drFAST* creates an index of blocks computed from all reads in 2-tuples $\tau_r = (s, L_r)$, where L_r denotes the list of reads that include the k-mer s.

5.1.4 Searching

drFAST compares the reference genome index keys with read index keys to find the locations in the reference genome where a read can be mapped with at most e errors. For each partition of the read, *drFAST* first finds the locations of the reference genome with the identical subsequence (same keys). It then tries to extend the location through sequence alignment of the reads to the genome, and reports those locations where the Hamming distance of the alignment is at most e. A simple loop scans both indices (both are lexicographically sorted); if the keys of the indices are the same (same subsequence) for entries $\tau = (s, L)$ in the reference and $\tau_r = (s, L_r)$ in the read index. Then all entries in L are candidate map locations for each read entry in L_r, thus the entire list L should be compared to L_r (extending step).

Similar to mrsFAST (See Chapter 3), *drFAST* performs *"all-to-all"* list comparison using a recursive divide-and-conquer strategy that guarantees cache obliviousness; i.e. asymptotically minimizing the number of costly cache misses [33].

5.1.5 Extending

The final step is to verify if each read can be aligned to candidate map locations within the user-specified error threshold e. *drFAST* aims to align the color-space read (S_c) to the letter-space sequence (S_l). The aligning process can be considered as finding a letter-space read S'_l that aligns to S_l, and highly similar to S_c if transformed to color-space:

$$argmax_{S'_l} \left(Sim(S_l, S'_l) + Sim(S_c, CCG(S'_l)) \right) \tag{5.1}$$

where CCG is the function that transforms the letter-space to color-space as defined by the SOLiD technology, and Sim is the similarity function.

Maximizing the similarity between two sequences is equivalent to minimizing their distance. We use Hamming distance (i.e. the number of mismatches) as the distance measure between two sequences.

$$argmin_{S'_l} \left(Diff(S_l, S'_l) + Diff(S_c, CCG(S'_l)) \right) \tag{5.2}$$

To address the problem, *drFAST* introduces two efficient methods.

Method I: Dynamic Programming.

Let $\sum = \{A, C, G, T\}$, and $\sigma, \sigma' \in \sum$, and let $Score(i, \sigma)$ indicate the optimal alignment of two subsequences $S_l[1..i]$ and $S_c[1..i]$ (from the first to the i^{th} character) while σ is the last character of S'_l. We then define

$$Score(i, \sigma) = d(S_l[i], \sigma) + min_{\sigma'}\{Score(i-1, \sigma') + d(S_c[i], CCG(\sigma'\sigma))\} \tag{5.3}$$

where $d(a, b) = 1$ if $a \neq b$, and $d(a, b) = 0$ otherwise.

The detailed version of equation 5.3 is as follows:

$$Score(i,'\texttt{A}') = d(S_l[i],'\texttt{A}') + min \begin{cases} Score(i-1,'\texttt{A}')+d(S_c[i],'\texttt{0}') \\ Score(i-1,'\texttt{C}')+d(S_c[i],'\texttt{1}') \\ Score(i-1,'\texttt{G}')+d(S_c[i],'\texttt{2}') \\ Score(i-1,'\texttt{T}')+d(S_c[i],'\texttt{3}') \end{cases}$$

$$Score(i,'\texttt{C}') = d(S_l[i],'\texttt{C}') + min \begin{cases} Score(i-1,'\texttt{A}')+d(S_c[i],'\texttt{1}') \\ Score(i-1,'\texttt{C}')+d(S_c[i],'\texttt{0}') \\ Score(i-1,'\texttt{G}')+d(S_c[i],'\texttt{3}') \\ Score(i-1,'\texttt{T}')+d(S_c[i],'\texttt{2}') \end{cases}$$

$$Score(i,'\texttt{G}') = d(S_l[i],'\texttt{G}') + min \begin{cases} Score(i-1,'\texttt{A}')+d(S_c[i],'\texttt{2}') \\ Score(i-1,'\texttt{C}')+d(S_c[i],'\texttt{3}') \\ Score(i-1,'\texttt{G}')+d(S_c[i],'\texttt{0}') \\ Score(i-1,'\texttt{T}')+d(S_c[i],'\texttt{1}') \end{cases}$$

$$Score(i,'\texttt{T}') = d(S_l[i],'\texttt{T}') + min \begin{cases} Score(i-1,'\texttt{A}')+d(S_c[i],'\texttt{3}') \\ Score(i-1,'\texttt{C}')+d(S_c[i],'\texttt{2}') \\ Score(i-1,'\texttt{G}')+d(S_c[i],'\texttt{1}') \\ Score(i-1,'\texttt{T}')+d(S_c[i],'\texttt{0}') \end{cases}$$

The minimum value of $Score(|S_l|,'A')$, $Score(|S_l|,'C')$, $Score(|S_l|,'G')$, and $Score(|S_l|,'T')$ is the score of the best translation of S_c to S'_l.

Figure 5.2 shows an example of aligning a letter-space and a color-space sequence using the dynamic programming described in equation 5.3. The minimum value in the last column represents the score of the best alignment. Using the backtracking pointer, we can then recover the best alignment sequence.

Remark 1 *The dynamic programming formulation in Equation (5.3) will find the optimal solution to the objective function in Equation (5.2) if the costs of mismatches and read errors are equal to one.*

Remark 2 *Dynamic programming described in (5.3) can be modified to handle any cost function for mismatches and read errors.*

Note that the equation (5.3) uses Hamming distance but it can be easily generalized for edit distance to allow indels.

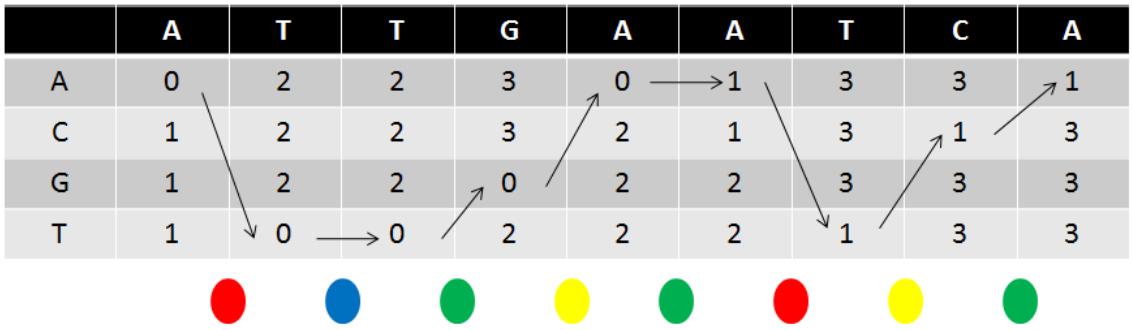

Figure 5.2: The dynamic programming table generated to align ATTGAATCA and 30121321 (0=blue, 1=green, 2=yellow, 3=red). The arrows represent the best alignment between the two sequences.

$$Score(i,j,\sigma) = min \begin{cases} Score(i-1,j-1,\sigma')+d(S_c[j],CGG(\sigma\sigma'))+d(S_l[i],\sigma) \\ Score(i-1,j,\sigma')+d(S_l[i],\sigma) \\ Score(i,j-1,\sigma')+d(S_c[j],CGG(\sigma\sigma')) \end{cases}$$

Method II: Transformation Based Detection.

The second method is based on the theoretical design aspect of color-space reads [83]. A string of colors $c_1c_2c_3\ldots c_k$ can also be treated as transformations. For example, $C102$ can be written as $f_2(f_0(f_1(C)))$ where the transformation of the colors is applied one after the other. This specific transformation converts C to G, acting as color 3 ($C102 = C3 = G$). For any other base pair, color string 102 will behave exactly as color 3.

The set of color operations is isomorphic to the "Klein Four Group" [7, 83]. The Klein Four Group is the symmetry group of a rectangle, which has four elements: the identity, the vertical reflection, the horizontal reflection, and a 180 degree rotation. In other words, given the four bases in the corners of a rectangle, each color operation has a one-to-one correspondence with one of the Klein Group elements (see Table 5.1). The Klein Four Group is closed under its elements meaning that if a and b are two elements of this group $a \oplus b$ and $b \oplus a$ ($a \oplus b$ means a followed by b) is also an element of the this group. It also has associative, identity, reverse and commutative properties. This means that any sequence of color operations can be considered as one color operation.

We use this property of the color-space reads to detect mismatches. Let two sets of

color operations of the same length exist ($c_1 \dots c_k$ and $r_1 \dots r_k$) with different starting color ($c_1 \neq r_2$). For both sets, if any two consecutive colors are replaced with their equivalent (closure property) starting from left hand side, you will end up with one at the end. If the last color matches with no intermediate matching colors then these two operations show a mismatch of length $k-1$. To illustrate this, consider two color operations 313 and 100. For simplicity, we also consider a leading base C. After applying the color operations, strings GTA and AAA will be generated respectively. It can be seen that the last base pair generated using both operations is A and intermediary base pairs are not matching. These two sets of operations have the same transformation, thus although they generate different sets of base pairs in middle, the final "product" is the same character.

Theorem 2 *Let $c = c_1c_2c_3c_k$ be a k-color substring of a read aligned with the corresponding color-space reference $r = r_1r_2r_3r_k$. Then c encodes an isolated $(k-1)$-base change if and only if the base position preceding c is not a variant, and the following two equations hold under the Color Addition Table 5.2:*

$$\sum_{j=1}^{k} c_j = \sum_{j=1}^{k} r_j$$

For all i from 1 to $k-1$:

$$\sum_{j=1}^{i} c_j \neq \sum_{j=1}^{i} r_j$$

We use Theorem 2 as the basis of our validation function (i.e. extending step). If there is a color mismatch between the read and the reference genome, we consider the next $e+1$ colors to test if there exists any same color transformation of size at most $e+1$ between the read and the genome. Considering a window of limited length, this sometimes may cause incorrect classification of a long stretch of mismatches as two independent read error calls. We refine such calls at the final step.

5.2 Additional Features

Parallelization. An *embarrassingly-parallel* wrapper for drFAST can be easily written to split the reads into smaller chunks (∼1-5 million reads per file) and align on cluster nodes.

Table 5.1: Applying color transformation '3' (a) is the same as applying 180^o rotation (b).

(a)

A	C
G	T

(b)

T	G
C	A

Table 5.2: Addition Table Code for Strings of Colors

$\oplus$	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

This approach the best practice because:
1) drFAST requires $<$ 700MB to load the genome and its index and only $\sim$ 1.2GB of memory to map 1M reads to the genome.
2) Mapping of each read is independent from mapping the others (except in the case of paired-end sequences where both ends need to be processed in the same chunk).

Paired-end Mapping. SOLiD, like most other HTS technologies can generate paired-end (PE) sequences. A pair of PE sequence are generated from the prefix and suffix of the same sheared DNA fragment, thus they can be used to increase mapping accuracy, and discover structural variation [4, 89]. Current implementation of drFAST supports tracking the paired-end information, enabling direct use of VariationHunter for structural variation [46] and transposon insertion [49] discovery, as well as NovelSeq [41] for characterization of novel sequence insertions.

5.3 Results

To measure the performance of *drFAST*, we compared its two variants to popular color-space read mappers currently available.

Benchmarked Software:

- *drFAST*-DP (Dynamic programming variant) (version 0.0.0.2);

- *drFAST*-CT (Color transformation variant) (version 0.0.0.0);
- BFAST [44] (version 0.6.4);
- Bowtie [63] (version 0.12.0);
- SHRiMP [100] (version 2.0.1);
- SOCS [96] (version 2.0.3);
- Mapreads [83] (version 2.4.1); and
- PerM [16] (version 0.3.3);

Parameters: We used the following parameter settings for these mappers:

- *drFAST*: e=2,3 (error threshold for different runs).
- BFAST: Parameters recommended in the BFAST manual.
- Bowtie: n,v=2,3 (error threshold for different runs); -a (for reporting all); -S (output in SAM format); -C (color-space mapping).
- SHRiMP: -m 1 (score 1 for match); -i -1 (score -1 for mismatch) -x -1 (score -1 for color-space error); -U (ungapped alignment) -o 10000 (maximum number of alignments for a read); -N 1 (number of threads); -h 96% ($\geq$96% alignment identity).
- SOCS: -x 0 (number of bases to trim); -s 2 (mismatch sensitivity); -t 4 (mismatch tolerance); -m 0 (maximum number of alignments for a read, 0 indicates to report all); -T 1 (number of threads); -N 1 (number of nodes); -l yes (consider the lower case bases in genome).
- Mapreads: S=0 (color-space mapping); M=2 (number of mismatches allowed); A=2 (count adjacent mismatches as one mismatch); Z=10000 (maximum number of alignment for a read).
- PerM: seed F2 (full sensitivity for 1 SNPs); -v 2 (number of mismatches); -k 1 000 000 (maximum number of alignment for a read); -A (report all possible mapping for a reads).

We used the same parameters (for reporting "all" mapping locations) when available to ensure a fair comparison.

Note that BWA and MAQ are not considered here since they ignore the first two characters of SOLiD reads.

Data, Reference Genome and Computing Power: We used both simulated and real data sets for comparisons. We simulated three sets, each with 4 million reads of length 50 bp sampled randomly from chromosome 1 of human reference genome (NCBI build 35) as follows:

- **Set 1:** We transformed the reads to color-space with no color errors and no mismatches.
- **Set 2:** Reads are transformed with two color errors. To achieve this, we transformed the reads to color-space and then changed the color of two arbitrarily selected non-consecutive positions. Note that if two color errors are consecutive, this might make it impossible to distinguish a read error from a SNP.
- **Set 3:** Generated with no color errors but one SNP.

In addition, we randomly selected 1 million (50bp long) reads from publicly available color-space reads generated from the genomes of NA18507 [83] (Accession Number: SRX004555), NA10847 (Accession Number: SRX008164), and NA12156 (Accession Number: SRX001969). We used the human reference genome (NCBI build 35, unmasked) as the reference genome in all our experiments. The benchmarking results we report are performed on a server with 64-bit Intel Xeon processor and 8 GB of RAM.

Time, Accuracy and Sensitivity Results: We give the comparison results for all the mappers above with respect to the proportion of the reads that have at least one map location on the reference genome (sensitivity), total number of map locations found (comprehensiveness), and time needed to map the reads.

Table 5.3 shows the results on simulated data sets with error threshold of 2 (color errors and mismatches). *drFAST* maps *all* of the reads from simulated data sets back to the reference genome very efficiently. The closest competitor to *drFAST* appears to be Bowtie, which is, in general, slower than *drFAST*-CT and is not 100% sensitive. Although Bowtie with a parameter setting of v=2 seems to map each read to more locations than *drFAST*,

Table 5.3: Performance results of all tested color-space read aligners on simulated data with error threshold of 2 mismatches.

Dataset	Mapper	Time (min)	Map Locations	Reads Mapped (%)
Set 1	*drFAST*-DP	65	138,715,908	**100**
	drFAST-CT	40	137,483,484	**100**
	BFAST	88	8,803,840	96.1
	Bowtie v=2	17	25,581,176	99.4
	Bowtie n=2	67	168,307,651	99.4
	SHRiMP	414	13,961,155	99.8
	SOCS	45	13,357,519	**100**
	Mapreads	50	55,569,848	**100**
	PerM	17	46,854,056	**100**
Set 2	*drFAST*-DP	42	37,652,313	**100**
	drFAST-CT	26	36,458,468	**100**
	BFAST	101	8,098,581	98.0
	Bowtie v=2	13	9,738,234	60.8
	Bowtie n=2	31	57,550,920	61.9
	SHRiMP	519	11,977,512	99.8
	SOCS	90	12,909,860	**100**
	Mapreads	31	21,749,155	**100**
	PerM	15	17,290,574	**100**
Set 3	*drFAST*-DP	47	76,588,622	**100**
	drFAST-CT	32	75,970,911	**100**
	BFAST	105	8,982,132	97.4
	Bowtie v=2	16	11,030,554	49.4
	Bowtie n=2	43	70,508,835	51.66
	SHRiMP	472	11,859,215	99.8
	SOCS	96	9,780,960	**100**
	Mapreads	37	29,799,473	**100**
	PerM	15	24,525,864	**100**

Reads are simulated from human reference genome build 35 (chromosome 1). Set 1: no errors, Set 2: color errors, Set 3: substitutions.

when no substitutions are present (Set 1), or a single color error is added (Set 2), this is simply due to Bowtie not being stringent on the number of errors it permits disregarding the parameter setting; we noticed that there are mapping locations with more than five color errors.

When the reads involve a nucleotide substitution (Set 3), the number of mapping locations are lower than that of *drFAST*. What is more interesting is the number of reads that can be mapped to the reference genome. It seems like Bowtie can map at most 61.9% of the reads even when they include a single color error (Set 2), in contrast, *drFAST* (both variants) map 100% of the reads. When the errors are in the form of nucleotide substitutions, the proportion of reads mapped by Bowtie drops to 51.66%.

Since Bowtie was the closest competitor to *drFAST*, we performed another experiment on the same data sets by increasing the error threshold to 3 (Table 5.4). Interestingly for this setting, the proportion of reads mapped by Bowtie is 99.4%, almost matching the 100% mapping sensitivity of *drFAST*. However, both in terms of time and the number of map locations *drFAST* (both variants) perform better than Bowtie, especially when errors (Set 2 for color errors and Set 3 for nucleotide errors) are present.

As all three sets are generated from chromosome 1 with at most two errors added, a sensitive mapper should be able to map all reads to chromosome 1 when the error threshold is set to 2. In order to experimentally check the accuracy of all locations found by *drFAST*, we simulated the corresponding Illumina reads (letter-space) and aligned to chromosome 1 using mrsFAST. As seen in Table 5.5 for Sets 1 and 3, *drFAST* finds slightly more mapping locations than mrsFAST, where the sensitivities of both aligners are 100%. The reason *drFAST* can find more mapping locations for SOLiD reads compared to the corresponding Illumina reads is because *drFAST* could map a read like `T0000` to two different positions with base pair contents `TTTT`, and also `CCCC` when one color error is "corrected". This is not the case with letter-space reads generated by a platform like Illumina Genome Analyzer. Although it would not be correct to arbitrarily select one "version" above the other, or returning both alignments as possibilities, we propose to correct such artifacts by incorporating the base pair quality values. This problem will arise only in polyN regions, thus, we propose to disable error correction of polyN reads where the base quality value of the first base is sufficiently high (i.e. $q > 30$).

BFAST and SHRiMP results are not presented for the three real data sets (Table 5.6) due to: (i) in our experiments, BFAST terminated with error in indexing step, (ii) SHRiMP

Table 5.4: Performance comparison on simulated data sets between *drFAST*-DP, *drFAST*-CT and Bowtie where error threshold is set to three mismatches.

Dataset	Mapper	Time (min)	Map Locations	Reads Mapped (%)
Set 1	*drFAST*-DP	88	364,601,231	**100**
	drFAST-CT	75	363,472,241	**100**
	Bowtie v=3	60	56,407,732	99.4
	Bowtie n=3	101	252,735,117	99.4
Set 2	*drFAST*-DP	42	118,053,818	**100**
	drFAST-CT	45	113,349,741	**100**
	Bowtie v=3	45	23,365,015	99.4
	Bowtie n=3	50	111,931,387	99.4
Set 3	*drFAST*-DP	47	215,274,860	**100**
	drFAST-CT	50	215,261,940	**100**
	Bowtie v=3	48	28,321,746	99.4
	Bowtie n=3	75	137,015,425	99.4

Values in bold show alignments with 100% sensitivity, higher value implies more sensitivity in the reported alignment.

requires 16 GB of main memory for alignment. Furthermore, both programs are much slower than *drFAST* or Bowtie. As a result, we compared *drFAST* with Bowtie, SOCS, Mapreads, and PerM with an error threshold of 2 (see Table 5.6) and an error threshold of 3 (See Table 5.7). In five out of the six cases, *drFAST* maps significantly more reads, and to substantially more locations, in comparable time. The performance of the two programs are comparable only for NA18507 for n=2, in terms of mapped reads; however, *drFAST*-CT is slightly faster on this data set.

We also compared the amount of memory used by each program when mapping 1 million reads to the human reference genome assembly (Table 5.8).

One important issue to note is that the *drFAST* aligner is aimed at SV/CNV inference and it does not return mapping quality values, which are still essential for accurate SNP detection. However, *drFAST* also returns "best" map locations for paired-end and matepair reads in addition to all possible discordant configurations where "best" is defined as the mapping with the lowest Hamming distance and with span size closest to the library average. Future releases of *drFAST* will have the capability of returning mapping quality for these

Table 5.5: Number of mapping locations reported by mrsFAST for the same set of simulated reads in letter-space.

Dataset	Mapper	Time (min)	Map Locations	Reads Mapped (%)
Set 1	mrsFAST	20	135,450,193	100
Set 3	mrsFAST	20	75,115,629	100

Table 5.6: Performance comparison on real data sets between *drFAST*-DP, *drFAST*-CT, Bowtie, SOCS, and PerM on 1 million randomly selected reads from three different sequencing experiments. We set the error threshold to 2 bp for all aligners.

Dataset	Mapper	Time	Map Locations	Reads Mapped (%)
NA18507	*drFAST*-DP	114	189,276,027	36.8
	drFAST-CT	54	149,362,540	35.6
	Bowtie v=2	21	64,092,233	27.8
	Bowtie n=2	63	202,948,323	35.2
	SOCS	320	21,081,941	35.3
	Mapreads	80	17,032,680	37.3
	PerM	76	51,012,126	35.2
NA10847	*drFAST*-DP	200	667,928,813	47.1
	drFAST-CT	100	512,599,230	45.9
	Bowtie v=2	84	280,928,112	38.0
	Bowtie n=2	91	270,996,634	36.0
	SOCS	420	53,668,622	44.8
	Mapreads	140	39,589,079	48.5
	PerM	100	132,417,348	44.6
NA12156	*drFAST*-DP	136	491,158,791	33.5
	drFAST-CT	91	440,317,111	32.5
	Bowtie v=2	99	329,916,108	25.0
	Bowtie n=2	99	318,621,596	23.7
	SOCS	400	38,246,530	31.2
	Mapreads	110	22,182,469	35.1
	PerM	140	64,821,620	31.1

Table 5.7: Performance comparison on real data sets between *drFAST*-DP, *drFAST*-CT and Bowtie on 1 million randomly selected reads from three different sequencing experiments. We set the error threshold to 3 bp.

Dataset	Mapper	Time (min.)	Map Locations	Reads Mapped (%)
NA18507	*drFAST*-DP	154	309,994,599	41.5
	drFAST-CT	61	302,237,779	40.5
	Bowtie v=3	63	145,473,423	37.1
	Bowtie n=3	78	290,357,005	36.35
NA10847	*drFAST*-DP	300	1,121,281,408	52.1
	drFAST-CT	141	1,092,259,727	51.6
	Bowtie v=3	182	565,114,739	47.8
	Bowtie n=3	76	270,885,799	35.8
NA12156	*drFAST*-DP	310	655,648,865	42.4
	drFAST-CT	120	639,667,174	39.5
	Bowtie v=3	187	585,191,747	34.6
	Bowtie n=3	98	318,527,434	23.6

Table 5.8: Memory required by each software to map 1 million 35-base reads to human reference genome. The memory requirement increases with the number of reads and/or the read length, this increase is typically linear with the increase in the number of base pairs in the data set.

Mapper	Memory Usage
drFAST-DP	1.3 GB
drFAST-CT	1.3 GB
BFAST	$\geq$ 10 GB
Bowtie	4.5 GB
SHRiMP	16 GB
SOCS	$\geq$ 5 GB[a]
PerM	2 GB
Mapread	$\geq$ 8 GB[b]

[a]SOCS memory usage is a parameter set by user, we used 5 Gigabytes of memory in our experiments; [b]Mapreads memory usage is a parameter set by user, we used 8 Gigabytes of memory in our experiments.

best map locations, which will effectively increase the appeal of *drFAST*, and users will be able to use it for both structural variation discovery through multi-mapping paired-end and matepair reads, and SNP discovery. Until this feature is available in *drFAST*, one may need to run other aligners in parallel to discover SNPs.

5.4 Conclusion

This is an exciting time for genomics research. The amount of available [1] and soon-to-be-generated [42] sequence data now arms us to expand our understanding human variation, disease susceptibility and genome evolution. Although the inherent accuracy and bias problems associated with different sequencing platforms [108], we can also leverage the different "strengths" of these technologies to increase confidence and comprehensiveness of SNP [95] and structural

For species where a reference genome is available such as human, mapping sequence reads to this reference assembly is the first step in genome analysis. Sensitivity, accuracy, as well as the speed of read alignment are crucial for precise characterization of genomic variants. To this end, many mapping algorithms were developed [71, 72, 68, 5, 44, 100] focusing mainly on the Illumina Genome Analyzer data, and very little effort was devoted to analyze color-space reads generated with the SOLiD platform [83, 100, 44]. The main limitation of the SOLiD-aware read aligners is that they were not optimized for structural variation detection (except for SHRiMP [100] which is more powerful in mapping to more complex areas of the genome), and they are unusable for segmental duplication analysis due to their unique mapping approach [5]. On the other hand, by tracking all possible map locations and underlying sequence variation, *drFAST* provides an opportunity to better access and increase "mappability" in repeat and duplication rich areas of the genome that are known to harbour much structural variation [56]. Although the **sensitivity** of *drFAST* is higher than the other aligners, we also demonstrate **speed enhancements** of both dynamic programming and color transformation versions. Through its readiness to be integrated to VariationHunter [46] for more sensitive SV discovery, and to NovelSeq [41] to characterize novel sequence insertions, and usability for segmental variation detection [5] *drFAST* is an important step forward for recovering additional genetic variation from di-base encoded color-space sequencing.

Chapter 6

Transcript to Genome Alignment

Computational identification of genomic structural variants via High Throughput Sequencing is an important problem for which a number of highly sophisticated solutions have been recently developed. The transcriptome refers to the complete collection of RNA sequences transcribed from portions of the genome; these include not only mRNAs but also non-coding RNAs.

In order to analyze structural variation within transcriptomic high-throughput sequencing (HTS) data (RNA-Seq) one typically needs to find the most likely transcript-to-genome alignment under the possibility of structural alteration events (Figure 6.1) such as: (1) internal duplications, which result in two separate segments in the transcript sequence aligning to the same segment of the genomic sequence; (2) inversions, which result in a segment of the transcript sequence aligning to the opposite strand of the genome than the rest of the transcript in an inverted fashion; (3) rearrangements, which result in a change of ordering of the aligned segments; and (4) fusions, which result in the transcript sequence aligning to two genes that are on two different chromosomes or far apart on the same chromosome. Note that an inversion can be of the type (1) suffix-inversion (or prefix-inversion), which involve a single breakpoint, where a suffix of the transcript sequence aligns to the strand opposite of that of the corresponding prefix, and (2) internal-inversion, which involves a pair of breakpoints, where the portion of the inverted transcript sequence aligns to the strand opposite to that of the flanking portions.

As HTS technologies progress, the length of the read-sequences they produce grows dramatically, and is expected to continue growing to over 1000bp per read. A longer read has a greater possibility of containing segments from more than two exons, complicating the

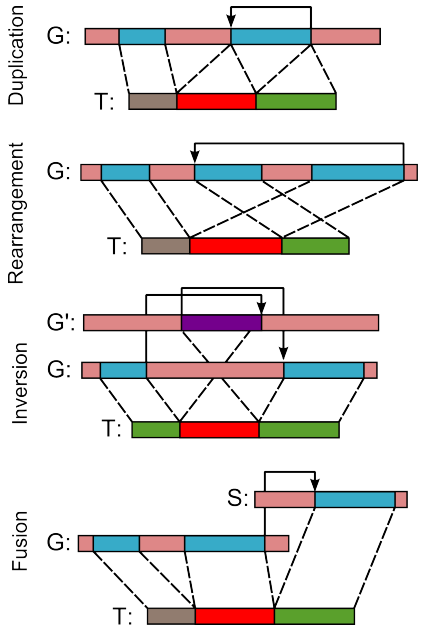

Figure 6.1: Structural alteration events considered in this chapter. T represents the transcript, G and S represents two genomic regions. G' is the complementary strand for G. Boundaries between red and green blocks indicate event breakpoints; arrows represent corresponding genomic transitions in the alignment. Apart from the event types shown in the figure, duplication events can appear as non-tandem and fusions can be between two different strands.

process of mapping such a read. Existing structural alteration detection tools are limited by their reliance on read-mapping tools designed for the short read sequences produced by the original HTS technologies; they cannot take advantage of increasing read length. Furthermore, the increase in read length improves the accuracy of *de novo* transcriptome assembly tools, leading to opportunities for the analysis of full transcript sequences.

Methods that have been proposed for spliced transcriptome-to-genome sequence alignment [6, 55, 107, 91, 121] implicitly assume the transcript sequences are devoid of structural alterations and use a seed-and-extend alignment strategy combined with a fragment chaining method. In the context of genomic sequences, alignment under structural alterations has been considered since the early 2000s. From a theoretical perspective, Cormode et al. introduced block edit distance [17] as the minimum number segment deletions, duplications and translocations in addition to single nucleotide insertions, deletions to transform (and

align) one sequence to another. Block edit distance in its most general setting is NP-hard to compute; [23] investigated many variants of the block edit distance under several restrictions that make the alignment problem computationally tractable. One such variant involving *1-monotonous* alignments was implemented in the context of genome-to-genome alignments [13]. Unfortunately no such method exists for transcript-to-genome alignments.

In this chapter, we introduce novel algorithmic formulations of the transcript-to-genome alignment under structural alterations problem and describe solutions for several gap penalty models. Our first formulation is a nucleotide-level alignment model that assumes the transcript sequence is a chain of *unidirectional* copies of segments taken from the genome - as investigated in [23]. We show how to sparsify the alignment table using a convex gap penalty model; we also show how to incorporate splice signal scores to the model. Our second formulation aims to reduce the running time and memory cost of the initial nucleotide-level alignment problem by sacrificing the sensitivity upon structures shorter than a user defined threshold value. In this formulation, we assume each alignment unit is a short segment shared between the transcript and the genome, possibly containing some mismatches. We then aim to find the *fragment chain* that gives the best overall alignment score based on the penalties described in the first formulation.

We introduces a novel computational tool, *Dissect* (Discovery of Structural alteration Event Containing Transcripts), suitable for high throughput transcriptome studies. To the best of our knowledge, Dissect is the first comprehensive stand-alone software for detecting and characterizing novel structural alterations in RNA-Seq data, and capable of direct global alignment of long transcript sequences to a genome. We report experimental results obtained by Dissect on a simulated mouse transcriptome database containing nucleotide-level and structural noise, as well as assembled RNA-Seq reads from the human prostate cancer cell line C4-2.

We tested Dissect on simulated transcripts altered via structural events, as well as assembled RNA-Seq contigs from human prostate cancer cell line C4-2. Our results indicate that Dissect has high sensitivity and specificity in identifying structural alteration events in simulated transcripts as well as uncovering novel structural alterations in cancer transcriptomes.

Dissect is available at `http://dissect-trans.sourceforge.net`

6.1 Methods

In this chapter we introduce a generalization of the transcriptome to genome spliced alignment problem, which allows the detection of transcriptional aberrations such as duplications, rearrangements and inversions. The model we use for our formulation corresponds to the *restricted asymmetric* variant of the block edit distance [23], in which the transcript sequence is represented by substrings extracted from the genome sequence, which are stitched together in various formations. Even though such an alignment model ignores the intermediate evolutionary steps of genomic modifications, it can still act as a realistic model of the transcription process with structural alterations, allowing the alignment of chimeric transcripts containing introns/deletions, novel insertions, duplications, rearrangements and inversions. One major caveat of aligning two genomic sequences using this approach is the omission of duplications in one of the aligned sequences [13], whereas the same approach does not necessarily pose a limitation for transcriptome to genome alignment since transcriptional structural alterations involve duplications on the transcript side, yet genomic duplications that appear in the transcript are not crucial for our analysis.

We further generalize our formulation for handling the special case of fusions that correspond to the alignment of a single transcript to two independent genomic sequences such that there can only be a single transition from the first sequence to the second and no transition from the second to the first. Finally, we incorporate additional score models for canonical splice signals into our formulation in order to represent a realistic model of the transcriptional machinery biased on canonical splice sites.

Below we describe a nucleotide-level transcriptome to genome alignment formulation for detection of structural alterations within the transcript. Based upon this approach, we also describe a number of algorithmic formulations for "chaining" fragments shared between the transcript and the genome sequences (within some small number of mismatches), considering alternative structural formations of the resulting fragment chain. Our formulation considers a number of genomic gap penalty models, in the form of general, convex and log-scale cost functions and transcriptional insertion penalties with a constant gap penalty model.

6.1.1 Nucleotide-level transcriptome to genome alignment under structural alterations.

Given $T = t_1 t_2 \dots t_N$, a transcript sequence and $G = g_1 g_2 \dots g_M$, a genomic sequence (including a gene), let the complementary genomic sequence to G be $G' = g'_1 g'_2 \dots g'_M$, where g'_i represents the complement of g_i. Also let $S = s_1 s_2 \dots s_L$ be a secondary genomic sequence (e.g. another gene) independent from G, and let $S' = s'_1 s'_2 \dots s'_L$ be its complement. S represents the potential fusion partner for G in the context of T.

We define an *alignment* of T to $\{G, G', S, S'\}$ under specific set of *structural alterations* A_s (which will be clarified later in the text) to be a mapping f from the nucleotides of T to those of G, G', S, S', as well as ϕ, representing a single nucleotide gap on the genome side of the alignment (note that f is not an invertible function). In order to prevent multiple fusions within the alignment, we restrict A_s such that if a nucleotide in T is aligned to S or S', none of the subsequent nucleotides in T can be aligned to G or G' - ensuring that T can be obtained by fusing at most two genes and there can be at most one "transition" from G/G' to S/S'.

We now define the score of an alignment with structural alterations A_s as

$$Score(A_s) = \sum_{i=1}^{N} S_m(t_i, f(t_i)) \quad - \sum_{\substack{1 \le i < j \le N \\ f(t_i), f(t_j) \neq \phi \\ \forall k, i < k < j, f(t_k) = \phi}} (P_s(t_i, t_j) - J_s(t_i, t_j))$$

Here S_m denotes the alignment score of each t_i to $f(t_i)$ - which is higher if $t_i = f(t_i)$. The second contribution to the alignment score is due to the penalties for all genomic transitions in the alignment, i.e. segments in T which are all mapped to ϕ, indicating a gap. Now we assign P_s as:

$$P_s(t_i, t_j) = \begin{cases} H_n(Gdist(t_i, t_j) - 1) & \text{forward transition} \\ H_b(Gdist(t_i, t_j) + 1) & \text{backward transition} \\ H_i(Gdist(t_i, t_j)) & \text{inversion transition} \\ C_f & \text{fusion transition} \end{cases}$$

Here $Gdist(a, b)$ denotes the genomic distance, namely, $|f(a) - f(b)|$. Note that forward transitions are genomic transitions between a pair of aligned positions that lie on the same

sequence (i.e. G, G', S, or S') such that the order of alignment is preserved with respect to the alignment direction (e.g. when aligned to complementary strand order will be reverse as well). The penalty function for forward transitions is a user defined function H_n (see the note below). In the presence of duplication and rearrangement events, aligned positions will be in reverse order with respect to their alignment direction and these types of backwards transitions are penalized by the user defined function H_b. Another type of transition considered is the inverted transitions penalized by the user defined function H_i; here, one alignment lies on the original sequence and the other on the complementary sequence. Finally, a constant gap penalty C_f is applied to fusion transitions in which t_i aligns to G/G' and t_j aligns to S/S'. Note again that although H_n, H_b and H_i are user defined functions, H_b and H_i (which correspond to structural alterations in the transcript) should be costlier than H_n (which corresponds to the regular genomic gap, presumably spanning an intron).

The second component of the transition penalty, J_s, is an additional score deducted from the penalty depending on the existence of canonical splice signals at the junction sites:

$$J_s(t_i, t_j) = \begin{cases} 0 & Gdist(t_i, t_j) < min\_intron \\ \frac{C_s}{2} * (J_b(t_i) + J_e(t_j)) & otherwise \end{cases}$$

Here min_intron corresponds to the minimum considered length of an intron (anything shorter is treated as a deletion) and J_b, J_e are binary functions that indicate whether the beginning and ending splice sites of the transition correspond to canonical splice signals. Note that the above formulation assumes a single canonical splice signal pair (e.g. GT-AG) and the penalty is additive with respect to the beginning and ending sites.

Based on the definitions above, observe that in the special case of an *insertion* in T, such that $t_{i+1} \dots t_{j-1}$ are all aligned to ϕ and $f(t_i)$ and $f(t_j)$ are consecutive nucleotides in G, G', S, S', the penalty is zero.

An Efficient Algorithm for the Nucleotide-Level Transcriptome to Genome Alignment with Structural Alterations Problem.

Given a limited variant of the alignment and the score function above, where only forward transitions are considered without splice signal scores on a single genomic sequence G, there is a simple algorithm (for arbitrary H_n) with running time $O(NM^2)$. This algorithm constructs an alignment table, X_G, of size $N \times M$ and initializes its first row as $max(S_m(t_1, g_j), S_m(t_1, \phi))$ for all $j \in [1, M]$. For each of the remaining rows, the transitions

from the previous row are evaluated representing genomic gaps between valid transcript position pairs. The gaps in the transcript are calculated as vertical transitions between two adjacent rows in the alignment table. This allows the construction of each row in $O(M^2)$ time, yielding the above running time.

[34] and [88] independently showed that when a restricted distance penalty scheme is assumed, the running time needed to construct a row can be reduced to $O(Mlog(M))$ using *sparse dynamic programming*. This restriction assumes a convex gap penalty function, $h: \mathbb{Z}^+ \cup \{0\} \rightarrow \mathbb{R}$ such that $h(x) - h(x-1) \geq h(x+1) - h(x) \geq 0$. [34] further reduced the running time to $O(M)$ for simple convex gap penalty functions, with the condition that for every $x_1, x_2 \in \mathbb{Z}$, $x_1 < x_2$ and $r \in \mathbb{R}$, the smallest integer value y that satisfies $y > x_2$ and $h(y - x_1) - h(y - x_2) - r \leq 0$ can be calculated in constant time. Log-scale distance penalty functions (in the form $a + b * log(distance)$), which model gap penalties quite realistically (as the intron lengths are geometrically distributed [14]), satisfy the simple convexity property. This allows the exact transcript to genome alignment without structural alterations to be performed in $O(NM)$ time.

We now show how to extend the above algorithms to handle splice signal scores and transcriptional structural alterations in the form of duplications, rearrangements, inversions and fusions.

Even if genomic distance penalties can be chosen as convex functions, the contribution from J_s may violate the convexity. To resolve this issue, we construct each row of the alignment table by the use of two independent processes; the first process calculates the genomic transitions from the previous row for the positions that constitute a canonical splice starting site, and the second process does the same for the positions that do not constitute a canonical splice starting site. For each of these processes, since J_b remains constant for a fixed position in the previous row, the set of forward transitions between two rows satisfy the convexity property. In order to obtain the optimal entries in a given row, we take the higher of the two values.

Even though the above formulation is for forward transitions only, we can perform all sparse dynamic programming operations in reverse order (with switched indices); as a result we can split each of the two processes described above into further two processes, one for forward and another one for backward transitions. This way we can capture duplication and rearrangement events that require a backwards transition in the alignment.

For handling inverted transitions we use a second alignment table, $X_{G'}$ for aligning T

with G', which is initialized in a similar fashion to X_G (naturally S_m values are obtained for the complementary nucleotide). Since the original sparse dynamic programming solution is designed for any arbitrary score values taken from the previous row, obtaining the row from table $X_{G'}$ will still be valid. Therefore we can further split the four processes described above for handling inverted and non-inverted transitions; each entry will then be assigned to the maximum valued result out of the eight computed. Computing a row in table $X_{G'}$ can be carried out similarly: non-inverted transitions involving forward/backward and canonical/non-canonical processes will be computed using the previous row in table $X_{G'}$, where the remaining inverted transitions will be computed using the table X_G.

For fusion cases, we use two new alignment tables X_S and $X_{S'}$ to our formulation (for aligning T with both S and S'). Even though the processes for constructing the rows of $X_S/X_{S'}$ are identical to $X_G/X_{G'}$, the initialization step requires handling potential fusion transitions. Before starting the row construction process we first identify the optimal fusion transition to each row in $X_S/X_{S'}$. For k^{th} row, the highest scoring entry within the first $k-1$ rows in X_G or $X_{G'}$ constitutes the highest scoring fusion transition score, combined with the constant fusion transition penalty C_f. Since fusion transition penalties are independent from genomic position, the highest scoring table entry gives the optimal fusion transition for any of the cells in the row with the same transition score.

The above algorithmic formulation provides solutions for arbitrary, convex and simple convex penalty formulations for handling structural alterations and splice signal scores within respective running times of $O(NM^2)$, $O(NMlog(M))$, and $O(NM)$.

6.1.2 Fragment chaining for transcriptome to genome alignment under structural alterations.

The problem of transcriptome to the genome alignment with structural alterations can be optimally solved in polynomial time under the assumption that the transcript sequence is composed of substrings copied from the genome sequence. For high-throughput transcriptome to genome alignment studies, however, running time and memory requirements of nucleotide level alignment will be costly even for log-scale gap penalty functions.

In this section we describe the algorithmic formulation for a "lower resolution" solution to the transcript to genome alignment with structural alterations. Given a set of "fragments" between the transcript and the genome sequences, this approach aims to find the *optimal* chain of fragments within certain constraints that will give the maximum alignment score

with respect to the fragment "qualities" and transition penalties. If the fragment length is one, this formulation reduces to the nucleotide-level formulation given.

Given a transcript sequence T and a pair of genomic sequences G and S (and their complementary sequences G' and S'), a fragment F is a segment of G, S, G' or S' which is also present in T within a small number of mismatches. Associated with F, we have (i) the starting position in T, (ii) the starting position in one of the genome sequences G, G', S, S', (iii) the length of the fragment, and (iv) the similarity score for the fragment, respectively denoted by $F.ts$, $F.gs$, $F.len$, and $F.score$. Similarly $F.ge$ and $F.te$ will denote the ending position of the respective sequences; e.g. for forward alignments, $F.ge = F.gs + F.len - 1$ and $F.te = F.ts + F.len - 1$. Fragments from G or S are aligned to T in the forward direction; the fragments from G' or S' are aligned to T in the reverse direction with complementary nucleotides. The score of the fragment is a function of its length and the number of mismatches between itself and its occurrence in T.

In the algorithmic formulation below, we are given a set (F_{set}) of K fragments shared between T and genomic sequences, G, G', S, and S', which are at least of a user specified length and have an alignment score higher than a user specified threshold (we describe how we obtain F_{set} later in the text). A pair of fragments can overlap in the transcript or in the genome sequence. However, for the description below, we do not consider a fragment in F_{set}, which is a sub-fragment of (fully included in, and in the same direction with) another fragment in the genome sequence and the transcript.

We define a *valid* disjoint fragment chain C as an ordered subset of F_{set} involving $k \leq K$ fragments, $(F_1, F_2, \ldots, F_k)$, such that (1) for each pair of subsequent fragments F_i, F_{i+1} (subsequent fragments are said to be *chained*) we have $F_i.te < F_{i+1}.ts$, and (2) if F_i is aligned to S/S', no F_j for $j > i$ is aligned to G/G'.

Our goal here is to find the valid disjoint fragment chain C_d (of length $B \leq K$) over F_{set} with the highest possible "score" with respect to the scoring function f_{score} and transition penalty function $f_{penalty}$ given as

$$\begin{aligned} f_{score}(C_d) = & \sum_{i=1}^{B} F_i.score - \sum_{i=1}^{B-1} f_{penalty}(i, i+1) \\ & - P_t(F_1.ts - 1) - P_t(N - F_B.te) \end{aligned}$$

$$\begin{aligned} f_{penalty}(x, y) = & \; P_t(F_y.ts - F_x.te - 1) + P_s(F_x.ge, F_y.gs) \\ & - J_s(F_x.ge, F_y.gs) \end{aligned}$$

Here, the "transcript gap penalty function" (according to the constant gap penalty scheme described in our nucleotide-level alignment formulation) is set to be $P_t(dist) = C_{gap} * dist$. The original genomic transition penalty function P_s and canonical splice signal scoring function J_s are as per the nucleotide-level alignment formulation.

It is possible to solve the problem described above by going through the fragments in F_{set} according to their starting position in T, computing the best scoring chain ending with each fragment via dynamic programming (see the description of the Dissect method). For any pair of user defined functions P_s and J_s this algorithm can find the optimal disjoint fragment chain in F_{set} in $O(K^2)$ time. Although it may be possible to improve the running time for restricted genomic gap penalty models involving, e.g., convex and simple convex cost functions, this algorithm is easy to implement and has proven to be sufficiently fast on the data sets we experimented with.

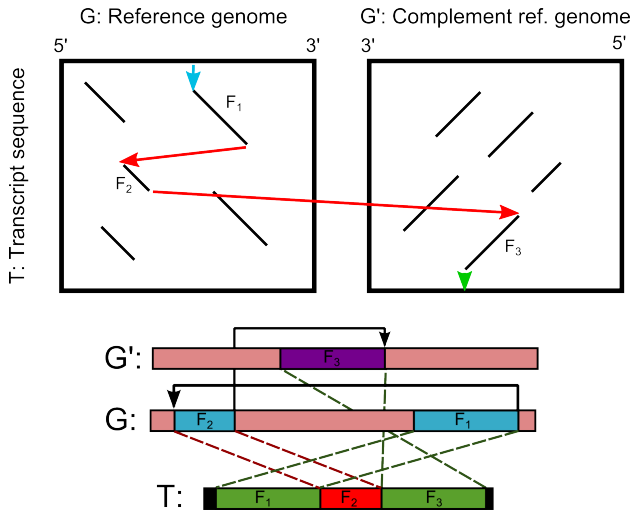

Figure 6.2: Fragment chaining in the presence of a rearrangement and an inversion. The fragments involved include two segments from T associated with segments from G and another segment from T associated with a segment from G'. The figure depicts how the fragments reveal themselves in the alignment tables and how they can be chained to get the overall alignment.

Fragment chains with overlapping fragments.

In a real-life experimental setting, a chain of fragments should be allowed to overlap to handle situations involving highly similar flanking sequences of a pair of chained fragments. Here we develop a generalized version of the formulation given above which allows the chaining of a prefix of a fragment F_i to a suffix of another fragment F_j so that the chosen prefix and suffix do not overlap in the transcript. First we redefine the concept of a valid fragment chain and then investigate different overlap resolution schemes.

Let a valid fragment chain with overlaps, C, be a sequence of $k \leq K$ fragments, $(F_1, F_2, \ldots, F_k)$, such that (1) the starting and ending positions of the fragments in the transcript increase throughout the chain and (2) if F_i the chain is aligned to S/S', F_j can not be aligned to G/G' for any $j > i$.

Our goal is to find the optimal overlapping fragment chain C_o (of length $B \leq K$) with a modified score function that differs from the original score function in only the transition penalty function, $f_{penalty}$, described below:

$$f_{penalty}(x,y) = \begin{cases} P_t(F_y.ts - F_x.te - 1) & \\ \qquad + P_s(F_x.ge, F_y.gs) & \text{disjoint pairs} \\ P_o(F_x, F_y) & \text{overlapping pairs} \end{cases}$$

Here $P_o(x,y)$ represents the special transition penalty for overlapping fragment pairs. Notice that the splice signal score function, J_s, is omitted from the penalty function. This is due to the complications that can be caused by the integration of splice signal scores and overlapping fragments.

Given a valid overlapping fragment chain, $(F_1, \ldots, F_k)$, an *overlap split position* between chained fragments F_i and F_{i+1}, is a position $r \in [F_{i+1}.ts - 1, F_i.te]$ indicating the modified ending position of F_i, and $r + 1$ indicating the modified starting position of F_{i+1} in the transcript. Below we show how to obtain the overlap split positions, and effectively resolve overlaps between a pair of chained fragments.

In a simple overlap resolution model, the penalty P_o can be set to the sum of the length of the overlapping interval and the penalty of the genomic gap: for an overlapping fragment pair F_i and F_{i+1}, we define $P_o = P_s(F'_i.ge, F'_{i+1}.gs) + (F_i.te - F_{i+1}.ts + 1)$, where F'_i and F'_{i+1} represent the updated (shortened) fragments. Since the overlap length is known, we simply have to find the overlap split position r that minimizes P_s.

Notice that for fragment pairs that are aligned in the same direction, the genomic distance between a pair of fragments increases with the length of the overlap - independent of the overlap split position; thus any overlap split position will do. For fragments that are aligned in different directions however, the overlap split position has an effect on the genomic distance between the two fragments. There are three different scenarios to consider in this case: (1) first fragment is located "earlier" in the genome, (2) first fragment is located "later" in the genome, (3) the overlapping regions of both fragments overlap in the genome as well. For each of the above scenarios, respectively selecting the "latest" overlap split position, the "earliest" overlap split position, and the overlap split position that makes the two updated fragments closest in the genome minimizes the genomic transition penalty - under the assumption that inversion transition penalty increases with the genomic distance. Since all these three cases can be handled in constant time, computing the optimal overlap split position for any pair of fragments can be performed in constant time. As a result, we can employ an algorithm quite similar to the one described for disjoint fragment chaining. The only difference is, when selecting the valid chains ending at each fragment F_i, the algorithm will need to also consider the fragments that overlap with F_i, but do not start earlier than $F_i.ts$ in T or end earlier than $F_i.te$. As per the algorithm for disjoint fragment chaining, this variant of the fragment chaining with overlapping fragments method needs $O(K^2)$ time.

For further improving the accuracy, one needs to consider the (eliminated) mismatches within the overlapping region. The penalty function, P_o, can now be defined as: $P_o(F_i, F_{i+1}) = P_s(F_i'.ge, F_{i+1}'.gs) + (F_i.score - F_i'.score) + (F_{i+1}.score - F_{i+1}'.score)$. The optimal split position of an overlapping fragment pair is that which minimizes the sum of the contributions from the updated genomic distance and the number of mismatches retained in the updated fragments. A naive method to handle this variant of the problem checks each position within the overlap to compute the minimum value for P_o. However, if there are no mismatches in either fragment, the problem reduces to that described above, and thus will have an efficient solution. Furthermore, if each fragment in F_{set} can only have a constant number of mismatches (one can enforce this in the definition of a fragment), a simple brute force search will compute the optimal split position for each pair in constant time, preserving the running time of $O(K^2)$.

The algorithmic formulation described above provides the theoretical underpinnings of the computational method we devised for identifying structural alterations leading to a transcript. However, as will be described in the next section, we need to take further steps

to make the fragment chaining solution efficient and its results close to those implied by nucleotide-level formulation towards a practical solution.

6.1.3 Whole genome analysis and discovery of novel transcriptional structural alterations with *Dissect*.

In this section, we describe some of the details of our computational tool *Dissect* (DIScovery of Structural alteration Event Containing Transcripts). Dissect has three main stages:

Genomic region inference

This stage begins by sampling anchors from the transcript sequence and mapping to reference genome within a user defined mismatch threshold. Although there are a number of "spliced" alignment methods in the literature, they either perform a local [55] or an anchor specific [121] analysis of anchor mappings. Because we aim to detect structural alterations, the order or direction of anchors are not necessarily preserved within the alignment. Thus, a global region inference approach oblivious to order or direction is more suitable.

For our purposes, an anchor is a substring of constant length L_A, of the transcript sequence T of length N. We generate the set of anchors from the transcript by sampling a user defined number of equally distanced anchors of length L_A, the first and last anchors corresponding to the beginning and the ending of the transcript respectively. Then we find all possible mappings of an anchor in the genome through the cache-oblivious short read mapper, mrsFAST (See Chapter 3), eliminating the anchors that have more mappings than a user specified threshold.

Within a set of anchor mappings $S_{map} = \{m_1, m_2, \ldots, m_K\}$ of size K, each mapping m_i (to a genomic region) is represented as $m_i = (m_i.t,\ m_i.g,\ m_i.score)$, which respectively correspond to the starting position of the anchor in the transcript, the starting position of the mapping in the genome, and mapping (similarity) score. Given the complete set of anchor mappings, we determine a "genomic region of interest" by finding all intervals within the genome (or two disjoint intervals for the fusion cases) to which a high number of distinct anchors are mapped - with high alignment scores. Since the region is preferred to be compact, our aligner only searches among the intervals that start at the starting position of an anchor mapping and end at the ending position of a mapping. This condition removes all intervals that have unnecessary extensions at each end and reduces the number of possible

genomic regions to $O(K^2)$.

For our purposes, a genomic region R is an interval that locally maximizes the following score:

$$Score(R) = c_N \times M(S_{map}, R)^{c_\alpha} / (\text{region length} + c_L).$$

Here $c_N \geq 1$, $c_\alpha \geq 1$ and $c_L > 0$ are user defined normalization parameters for adjusting relative significance of the number of anchors contained within the region to the length of the inferred region. M, on the other hand, is the function defined to be the sum of best mapping scores of all anchors mapped to the region.

Our genomic region inference method initially sorts all anchor mappings according to their genomic position. It then goes over each mapping position and calculates the above score for all possible genomic intervals starting at that position and commits to the one with the highest score- in linear time via dynamic programming.

A second type of genomic region inference needs to find fusion regions that appear as intervals separated by long inter-genic regions on the same genomic sequence (e.g. intra-chromosomal fusions) or possible different genomic sequences (e.g. inter-chromosomal fusions). Since Dissect does not utilize gene annotation for region inference, we do not differentiate between single gene alignments and inter-genic fusions between closely located genes. This step essentially corresponds to the inference of two separate regions and a transcript cut position that yield the highest double region inference score, which is the sum of scores of the two regions such that the first region score is only calculated over the anchor samples taken from upstream of the transcript cut position and the second region score is only calculated upon the samples taken from downstream.

Instead of looking for combinations of regions that give the optimal double region score, Dissect scans over all possible anchor split positions and search for single regions for both ends of the transcript independently. Optimal double region score for this anchor split is the sum of optimal single regions that cover anchors on each side. Given c_A anchors sampled from the transcript, we find the optimal double region pair in $O(c_A * K^2)$ time.

Even though the region inference methods described above search for the highest scoring genomic region/region pair, Dissect processes all regions whose scores are within a constant factor of the highest score (and within a user defined maximum number of disjoint regions threshold). Each of these reported genomic regions/region pairs are analyzed separately within the downstream alignment pipeline.

An important issue to address is the relation between inferred single regions and double

regions. Even though we can efficiently find the highest scoring single/double regions from a given anchor set and its mappings, it is not possible to objectively compare scores of inferred single/double regions. For instance, if there is a long intron in the alignment of the transcript, the double region that spans the exons on each side of the intron might score higher than a single region that encompasses the entire transcript alignment if c_α is 1. Furthermore, a distance threshold would require prior knowledge of fusions within the genome that is analyzed. Dissect uses a double layer inference step as a workaround to this single/double region score comparison issue. In the first inference layer, we find a set of highest scoring single regions. If any of these regions cover a user determined percentage of the sampled anchors in the transcript, a double region is not searched for. If there is no such single region however, our method searches for high scoring double regions. If there is no high quality single or double region, our method does not report a region and considers the given transcript sequence as not represented within the genome with high similarity.

As the final step of this stage, regions that are overlapping or close to each other are combined into single regions (these include fusion regions that are relatively close to each other). In addition, the region boundaries are extended allowing flanking sequence from the beginning/ending anchors that are not represented in the set of mappings. The resulting genomic intervals are passed on to the second stage of Dissect which finds the optimal fragment chain with structural alterations between the transcript and inferred regions.

Fragment set construction and chaining

In the fragment set construction stage, we construct a set of fragments shared between the transcript and the genomic region(s) inferred in the previous stage. For that purpose we modified mrsFAST alignment method, to identify "seeds" (of a user specified length - which can be overlapping) in the transcript and maps them to both strands of the genomic sequence. After obtaining all possible seed mappings, the modified mrsFAST extends each fragment on both ends under certain extension constraints. These constraints are defined in the form of thresholds that limit total number of errors, number of consecutive errors that can appear in the fragment and minimum sequence similarity required for each k-mer of the fragment. After the fragment set is constructed, we employ the overlapping fragment chaining algorithm once using forward and a second time using reverse splicing signals (GT-AG and CT-AC). In the case of a single region inferred in the previous stage, the chaining solution will only consist of G/G' sequences. If a pair of regions are inferred, the chaining

solution will consider all G/G' and S/S' sequences allowing fusion transitions.

A key difference between our original formulation and the chaining method used in Dissect is the use of splice signal scores. Even though splice signal scores were omitted in our original overlapping chaining formulation, in practice it would be useful to have a two layered chaining/post-processing approach that constructs the chain that does not attempt to determine the exact overlap split positions, but performs a more accurate overlap analysis together with splice signals as a post-processing step after the optimal chain is obtained. The optimality condition here is also modified in the sense that splice signal score is incorporated when two disjoint fragments are chained together, yet omitted for overlapping fragments.

At the end of fragment chaining step, a tentative chain is obtained that represents the general structure of the alignment, yet overlap split positions are not exactly specified in the resulting chain. Dissect detects the exact split position through a post-refinement method described below.

Post-refinement of the fragment chain

In the post-refinement stage, we adjust the boundaries of fragment pairs that potentially contain minor misalignments due to the limitations introduced in the fragment construction step. In order to resolve these, we implemented several post-refinement steps that (1) combine fragments which are separated by a single nucleotide indel or a mismatch in their alignment, (2) classify and modify short overlaps in the genome, (3) fill in short gaps in the transcript between adjacent fragments in the chain, and (4) find optimal split position for overlapping fragment pairs.

In the case of two adjacent fragments being separated by an indel or a mismatching transcript-genome nucleotide, the two fragments are simply combined into a single fragment that contain an error in between. Clearly, if the fragments are separated by an indel, the combined fragment will also contain indels. Even though this is against the original fragment construction constraints; as a post-processing method, it only affects the resulting chain and not the chaining formulation.

When there is an adjacent fragment pair with a short overlapping region in the genome, Dissect does not directly report the overlap region as a duplication as this can also be an inserted region that displays sequence similarity to one of the flanking sequences. Since it is difficult to differentiate between the two, we allow the user to define lower and upper thresholds on length. Overlaps that are shorter than the lower threshold are treated as

insertions, whereas overlaps that are longer than the upper threshold are reported as regular duplicated regions in the transcript. The overlap regions that fall in between are reported as an ambiguous insertion/duplication region should be further analyzed by the use of gene annotations.

We have two additional refinement methods that require efficient implementations: (1) If two adjacent fragments have a gap in between them in the transcript, the formation might indicate a novel insertion of the size of the gap. (2) Alternatively, the region of the gap might belong to one of the exons represented by adjacent fragments (or both), yet may have relatively low similarity to the corresponding genomic region. In order to test the latter case, we perform a double-sided semi-global alignment on the flanking sequences of the fragment pair in the genome and the gap sequence. This alignment scheme aims to optimize the sum of the semi-global (overlap-detection) alignment scores on each side of a fixed split position in the transcript gap region. The method we apply is analogous to the *Sandwich DP* method proposed for GMAP spliced alignment algorithm [121]. The key difference in our application is the consideration of alignment with various structural alterations and their effect of fragment directionality. Within the alignment table, we also mark the GT-AG/CT-AC splice signals and take them into account for the computation of the new transition penalty between the updated fragments. This allows a fair alignment score comparison between the chains that go through this post-processing step and the chains that do not. Note that this gap refinement scheme has a running time of $O(l^2)$ per refinement (l is the length of the transcript gap).

As mentioned in the description of the fragment chaining stage; the exact overlap split position for overlapping fragments is not determined during the execution of the dynamic programming method but is left for a more detailed analysis in the post-refinement stage. In this stage, Dissect searches for the optimal overlap resolution according to an extended version of the overlap resolution scheme described in the previous section. In this version of the overlap resolution scheme, we combine splice signal scores with the original accurate overlap resolution that considers mismatch retention and updated genomic distance for the overlap penalty. Even though this extension was not feasible during fragment chaining, if the overlaps are resolved in an iterative fashion from the fragment pairs at the beginning of the chain towards the end, the number of splice sites that need to be checked in the genome stays within $O(N)$. Incorporating splice site scores in this manner also allows us to have a fair comparison basis when comparing the relative scores of the fragment chains for various

regions inferred in the first stage.

When two adjacent overlapping fragments have near perfect sequence similarity and no splice signals to identify the exact splicing position, there can be multiple overlap split positions with equal updated fragment scores. In such cases, our aligner reports the earliest split in the transcript but also provides an output field indicating equivalent split positions. This additional output can be used to reconstruct all optimal overlap splitting selections.

6.2 Results

We first report the performance of Dissect on simulation datasets derived from NCBI *RefSeq* transcript sequences and *Known Gene* gene structure database [51], which are subjected to nucleotide level substitution/indel noise with varying frequencies, novel oligonucleotide sequence insertions, and structural alterations at different length distributions, including exon duplications, inversions, rearrangements and transcript-transcript fusions. To demonstrate the performance of Dissect on *real* human transcriptome data, we report on an RNA-Seq dataset comprising 50bp reads from the prostate cancer cell line C4-2, assembled through Trans-AbySS transcriptome assembler [11].

Wild-type transcripts with novel insertions and nucleotide-level alterations We first evaluate Dissect's false discovery rate through the use of a data set comprised of wild-type transcripts. For that, we used NCBI RefSeq mRNA annotation dataset including the whole mouse transcriptome (build of July 18, 2011). This annotation dataset is (presumably) composed of wild-type transcripts that do not contain structural alterations. We evaluated Dissect's false event discovery rate by aligning all transcripts from this data set to the mouse reference genome (build mm9). Since most of these sequences have very close matches to genes in the mouse genome, we also used this dataset to evaluate the accuracy of Dissect alignments (obtained through fragment chaining) at nucleotide-level resolution.

After the removal of poly-A tails, the entire data set containing 28060 RefSeq sequences of average length of 2848 nucleotides, were aligned to the mouse reference genome using the default parameters of Dissect. We report on the highest scoring alignment for each transcript sequence. Among them, 27922 (99.51%) did not contain structural alterations, 49 (0.17%) were reported to contain a structural event, and 38 (0.14%) were identified to contain a short ambiguous insertion or duplication event. The remaining 51 sequences had no high

similarity alignments. On a standard single core processor, aligning the entire dataset of 28060 sequences with Dissect took less than 80 minutes.

Next, we aimed to observe how Dissect's false event discovery rate varies as a function of sequence divergence (between the transcript and the genome) - and thus sequencing error rate. For that, we modified the original RefSeq sequences by adding nucleotide-level substitution/indel errors. These errors were added independently at random in each position of a transcript sequence: based on a recent study on error rates in Illumina sequencing [90], single nucleotide indel to substitution error ratio was set to 1/150 and insertion to deletion ratio was set to 1/10. When the sequencing error rate was set to 1%, 27917(99.4%) sequences out of 28060 were aligned as wild-type transcripts without no alterations. For 52 sequences, a high similarity alignment was not reported. Structural alterations were reported for only 57 transcripts and short ambiguous duplication/insertions were detected for 34 transcripts. When the sequencing error rate was set to 4%, the number of transcripts with a wild-type transcript alignment was reduced to 27756(98.9%). Among the remaining 304 sequences, 188 did not produce a high similarity alignment.

Finally, we used the latest RefSeq mRNA annotation dataset for whole human transcriptome (build of July 18, 2011) for the purpose of evaluating Dissect's false event discovery rate in the presence of short-to-medium size novel insertions.

In order to simulate a realistic sample of novel human genome insertions, we sampled substrings of varying length from the set of insertion sequences reported in a novel insertion characterization study [57], and inserted them to the transcript sequences at random exon breakpoints. Our data set included 33460 sequences that were devoid of structural alterations and had nucleotide-level accurate alignments (after the removal of poly-A tails). We equally partitioned this dataset into four subsets, each subject to a specific insertion size. To obtain realistic novel insertion sequences, we used 2363 known novel insertion sequences [57], from which we randomly picked a position in each sequence and extracted the sequence of the required length.

Table 6.1 depicts the false event detection rate for novel insertions shorter than 35 nucleotides. The higher rate of false positives for longer insertion sizes is caused by Dissect's high sensitivity to sequence similarity. Since the insertion sequences are obtained from a real novel insertion study for the human genome, there might be sequences highly similar to the insertion nearby the aligned gene loci, which increase the risk of identifying false rearrangements.

Table 6.1: Alignment results of Dissect for the simulated wild-type transcriptome dataset with novel insertions. Rows represent the length interval of the novel insertion distributions (e.g. insertions reported in the first row are uniformly distributed between 6 and 20 nucleotides). Columns indicate the output labels of Dissect: *All events* column represents the total number of transcripts Dissect has identified as a structural alteration *A. D/I* column represents the alignments that contain a short ambiguous interval that cannot be verified with certainty as an insertion or a duplication, and *N.A.* column indicates the number of transcript sequences for which Dissect did not return a valid high similarity alignment.

Insertion Length	Total	WT	All events	A. D/I	N.A.
6-20 bases	8365	8335	12	16	2
21-35 bases	8365	8284	52	23	6
36-50 bases	8365	8223	106	24	13
51-65 bases	8365	8117	204	20	24

Simulated transcriptional events. In order to estimate the sensitivity of Dissect, we initially prepared wild-type transcriptome datasets without any structural alterations using the Known Genes mouse gene structure annotation database [51] and modified extracted wild-type transcript sequences according to various structural alteration scenarios. In this step, any transcript sequence shorter than 50bp is removed, since structural modifications in such short transcript sequences often prevent reliable mapping of the anchor sequences used by Dissect.

The thirteen simulations described below aim to emulate the aberrant formations that can occur in transcripts due to structural alterations. These simulations involve: (1) *tandem duplications of the full transcript*, (2) *tandem duplication of the longest exon*, (3) *tandem duplication of the shortest exon*, (4) *internal-inversion of the longest exon*, (5) *internal-inversion of the shortest exon*, (6) *suffix-inversion with a breakpoint close to middle (prefix to suffix ratio: 36%-65%)*, (7) *suffix-inversion with a breakpoint close to the beginning/end, (prefix to suffix ratio: 16%-35% or 66%-85%)*, (8) *rearrangement of the full transcript sequence from a particular split position*, (9) *rearrangement of adjacent exons*, (10) *rearrangement of non-adjacent exons*, (11) *well-balanced fusions* (shorter fused sequence is $\geq$ %60 of the longer one), (12) *moderately-balanced fusions (*30% $\leq$ *short to long ratio* $<$ 60%*)*, (13) *imbalanced fusion* (short to long ratio: $<$ %30). The distribution of transcript alignments according to these event type labels for various event simulations are given in Table 6.2.

Table 6.2: The number of structural alterations detected by Dissect for the simulation datasets.

	Tot.	Tot-E.	Fusion	Inv.	F. Dup.	F. Rea.
Exp. 1	5234	5099	1	5	5092	1
Exp. 2	5234	5172	0	1	5171	0
Exp. 3	5234	5093	0	0	5093	0
Exp. 4	4788	4762	0	4762	0	0
Exp. 5	4788	4331	0	4331	0	0
Exp. 6	3188	3125	0	3125	0	0
Exp. 7	4654	4501	0	4501	0	0
Exp. 8	4788	4512	2	8	3	4499
Exp. 9	4788	4623	0	8	2	4613
Exp. 10	4316	4255	0	14	4	4237
Exp. 11	1312	1237	1232	5	0	0
Exp. 12	1558	1433	1433	0	0	0
Exp. 13	2363	562	562	0	0	0

Tot. = Total number of transcript sequences, *Tot-E.* = Total number of discovered structural event containing transcripts, *Fusion* = Total number of fusions, *Inv.* = Inversion events including inverted duplications, inverted rearrangements, in-place inversions, and suffix inversions, *F. Dup.* = Forward duplications, *F. Rea.* = Forward rearrangement events.

Transcriptomic Structural Alterations in Prostate Cancer Cell Line C4-2. We applied Dissect to high coverage 50bp RNA-Seq read data from human prostate cancer cell line C4-2. The reads were assembled using short-read transcriptome assembler Trans-Abyss [11] version 1.2.0, using k-mer sizes of 26 and 49 - the minimum overlap length between two reads to be combined in a contig. For two contigs to be merged we required 10 pair-end mappings between the contigs.

Among a total of 576,381 contigs assembled, Dissect did not return a high quality alignment for 167,187 of them. Among the remaining contigs, 391,293 of them were aligned with no structural alterations. In 4,309 contigs, Dissect detected an ambiguous short insertion/duplication region. In 13,583 contigs, Dissect discovered a structural alteration: 1,044 fusions, 1,331 duplications, 555 rearrangements, and 10,653 inversion events. 10,992 of 12,539 non-fusion event contigs displayed $\geq 90\%$ overlap with a single gene annotated in HG18 Known Genes dataset. Among 10,653 inversions, 69 are multiple breakpoint inversions and another 79 contain combined duplication/rearrangement events. Within the

remaining $10,505$ single-breakpoint suffix-inversion events, $2,600$ contain overlapping regions within the two strands and $7,905$ have non-overlapping suffix/prefix formation.

We compared Dissect alignment results on contigs from C4-2 with long range PCR validated fusions reported by the fusion discovery tool Comrad [85] on the same data set. Among 8 validated gene fusion pairs, RERE-PIK3CD, HPR-MRPS10, CCDC43-YBX2, TFDP1-GRK1, BMPR2-FAM117B, GPS2-MPP2, MIPOL1-DGKB, ITPKC-PPFIA3, Dissect correctly identified 6 of them: RERE-PIK3CD, HPR-MRPS10, CCDC43-YBX2, BMPR2-FAM117B, MIPOL1-DGKB, ITPKC-PPFIA3. Note that because the two genes involved in the fusion BMPR2-FAM117B are in close genomic proximity, Dissect returned a rearrangement, rather than a fusion as per it is set to do. Among the two fusions Dissect could not identify, there was no contig returned by the assembler that spanned the TFDP1-GRK1 fusion breakpoint and the assembled contig spanning the GPS2-MPP2 fusion breakpoint was highly imbalanced (10% : 90%). Note that for three out of these four genes, GPS2, MPP2 and TFDP1, Dissect also reported wild-type alignments, without any evidence for a fusion event.

In order to better understand and differentiate the limitations of Dissect from that of the assembly process, we extracted breakpoint sequences of length 200bp for each of the eight gene fusion events given above. Dissect produced alignments that correctly capture the fusion breakpoint for each of the eight fusions. Six of these breakpoint sequences were reported as straightforward fusions. Among the remaining two breakpoints transitions, BMPR2-FAM117B was identified as a rearrangement event and TFDP1-GRK was identified as a wild-type alignment due to close proximity of the fused genes: a comparison with gene annotation uncovered the inter-genic structure of the breakpoints discovered.

6.3 Conclusion

We introduce novel algorithmic formulations for the problem of aligning transcripts to a genome under structural alterations such as duplications, inversions, rearrangements and fusions.

Our first formulation involves nucleotide-level alignment that can detect structural alterations by a single unified dynamic programming approach. The fastest algorithms we developed for this formulation require $O(NMlog(M))$ time for convex genomic gap penalties and $O(MN)$ time for simple convex (including logarithmic) gap penalties (M and N

correspond to the lengths of transcript and genome sequences respectively).

Our second formulation allows a faster but lower-sensitivity solution for a whole genome alignment setting. Given a set of shared fragments between the transcript and the genome, we show how to obtain an optimal chain of fragments in $O(K^2)$ time for disjoint or overlapping fragments (K being the total number of fragments).

We also present a novel computational tool, Dissect, which implements the fragment chaining formulation described above. Dissect achieves high sensitivity and specificity in identifying structural alterations in simulated data sets, as well as in uncovering gene fusions in a prostate cancer cell line.

Chapter 7

Boosting Sequence Compression Algorithms

As mentioned in Chapter 1, the high throughput sequencing (HTS) platforms generate unprecedented amounts of data that introduce challenges for the computational infrastructure. Data management, storage, and analysis have become major logistical obstacles for those adopting the new platforms. Fast and efficient compression algorithms designed specifically for HTS data should be able to address some of the issues in data management, storage, and communication. Such algorithms would also help with analysis provided they offer additional capabilities such as random access to any read and indexing for efficient sequence similarity search.

Available compression techniques for HTS data either exploit (i) the similarity between the reads and a reference genome [50, 61] or (ii) the similarity between the reads themselves [124, 125, 111, 19, 122, 18, 54, 12]. In particular, the Lempel-Ziv methods [124, 125] (e.g gzip and derivatives) iteratively go over the concatenated sequence and encode a prefix of the uncompressed portion by a "pointer" to an identical substring in the compressed portion. This general methodology has three major benefits: (i) Lempel-Ziv based methods (e.g. gzip and derivatives) have been optimized through many years and are typically very fast; in fact, the more "compressible" the input sequence is, the faster they work, both in compression and decompression; (ii) these methods do not need a reference genome; and (iii) because these techniques are almost universally available, there is no need to distribute a newly developed compression algorithm.

Interestingly, the availability of a reference genome can improve the compression rate achieved by standard Lempel-Ziv techniques. If the reads are first mapped to a reference genome and then reordered with respect to the genomic coordinates they map to before they are concatenated, they are not only compressed more due to increased locality, but also in less time. This, mapping first compressing later approach, combines some of the advantages of the two distinct sets of methods above: (i) it does not necessitate the availability of a reference genome during decompression (compression is typically applied once to a data set, but decompression can be applied many times), and (ii) it only uses the re-ordering idea as a front end *booster* (Burrows Wheeler transform –BWT– is a classical example for a compression booster. It rearranges input symbols to improve the compression achieved by Run Length Encoding and Arithmetic Coding [99, 118, 103] . Further boosting for BWT is also possible: see [30, 28, 27]). Any well-known, well-distributed compression software can be applied to the re-ordered reads. Unfortunately, this strategy still suffers from the need for a reference genome during compression.

In this chapter, we introduce a novel HTS genome (or transcriptome, exome, etc.) sequence compression approach that will combine the advantages of the two types of algorithms above. It is based on reorganization of the reads so as to "boost" the locality of reference. The reorganization is achieved by observing sufficiently long "core" substrings that are shared between the reads, and clustering such reads to be compressed together. This reorganization acts as a fast substitute for mapping based reordering (see above); in fact the first step of all standard seed and extend type mapping methods identify blocks of identity between the reads and the references genome.

The core substrings of our boosting method are derived from the Locally Consistent Parsing (LCP) method devised by Sahinalp and colleagues [102, 17, 9]. For any user-specified integer c and with any alphabet (in our case, the DNA alphabet), the LCP identifies "core" substrings of length between c and $2c$ such that (i) any string from the alphabet of length $3c$ or more include at least one such core string, (ii) there are no more than three such core strings in any string of length $4c$ or less, and (iii) if two long substrings of a string are identical, then their core substrings must be identical.

LCP is a combinatorial pattern matching technique that aims to identify "building blocks" of strings. It has been devised for pattern matching, and provides faster solutions in comparison to the quadratic running time offered by the classical dynamic programming schemes. As a novel application, we introduce LCP to genome compression, where it aims to

act as a front end (i.e. booster) to commonly available data compression programs. For each read, LCP simply identifies the longest core substring (there could be one or more cores in each read). The reads are "bucketed" based on such representative core strings and within the bucket, ordered lexicographically with respect to the position of the representative core. We compress reads in each bucket using Lempel-Ziv variants or any other related method without the need for a reference genome.

As can be seen, LCP mimics the mapping step of the mapping-based strategy described above in an intelligent manner: on any pair of reads with significant (suffix-prefix) overlaps, LCP identifies the same core substring and subsequently buckets the two reads together. For a given read, the recognition of the core strings and bucketing can be done in time linear with the read length. Note that the "dictionary" of core substrings is devised once for a given read length as a pre-processing step. Thus, the LCP-based booster we are proposing is very efficient. LCP provides mathematical guarantees that enable highly efficient and reliable bucketing that captures substring similarities.

Our tests indicate that SCALCE can improve the compression rate achieved through gzip by a factor of 4.19 - when the goal is to compress the reads alone. In fact on SCALCE reordered reads, gzip running time can improve by a factor of 15.06 on a standard PC with a single core and 6GB memory. Interestingly even the running time of SCALCE + gzip improves that of gzip alone by a factor of 2.09. When compared to the recently published BEETL - which aims to sort the (inverted) reads in lexicographic order for improving bzip2, SCALCE+gzip provides up to 2.01 times better compression while improving the running time by a factor of 5.17. SCALCE also provides the option to compress the quality scores as well as the read names, in addition to the reads themselves. This is achieved by compressing the quality scores through order-3 Arithmetic Coding and the read names through gzip through the reordering SCALCE provides on the reads. This way, in comparison to gzip compression of the unordered FASTQ files (including reads, read names and quality scores), SCALCE (together with gzip and arithmetic encoding) can provide up to 3.34 improvement in the compression rate and 1.26 improvement in running time.

SCALCE (Sequence Compression Algorithm using Locally Consistent Encoding) is implemented with both gzip and bzip2 compression options. It also supports multi-threading when gzip option is selected, and the pigz binary is available. The source code for SCALCE is available at `http://scalce.sourceforge.net/`

7.1 Methods

7.1.1 A theoretical exposition to the LCP technique

The simplest form of the LCP technique works only on reads that involve no tandemly repeated blocks (i.e. the reads can not include a substring of the form XX where X is a string of any length ≥ 1; note that a more general version of LCP that does not require this restriction is described in [101, 102, 9] so that LCP works on any string of any length). Under this restriction, given the alphabet $\{0, 1, 2, \ldots, k-1\}$, LCP partitions a given string S into non-overlapping blocks of size at least 2 and at most k such that two identical substrings R_1 and R_2 of S are partitioned identically – except for a constant number of symbols on the margins. LCP achieves this by simply marking all local maxima (i.e. symbols whose value is greater than its both neighbours) and all local minima which do not have a neighbour already marked as a local maxima - note that beginning of S and the ending of S are considered to be special symbols lexicographically smaller than any other symbol. LCP puts a block divider after each marked symbol and the implied blocks will be of desirable length and will satisfy the identical partitioning property mentioned above. Then, LCP extends each block residing between two neighbouring block dividers by one symbol to the right and one symbol to the left to obtain *core blocks* of S. Note that two neighbouring core blocks overlap by two symbols.

7.1.2 Example

Let $S = 213120321020213120321 02$; in other words $S = X\mathbf{0}X$, where $X = 21312032102$. The string S satisfies the above condition; i.e. it contains no identically and tandemly repeated substrings. When the above simple version of LCP is applied to S, it will be partitioned as |213|12|03|2102|**0**2|13|12|03|2102|. Clearly, with the exception of the leftmost blocks, the two occurrences of X are partitioned identically. Now LCP identifies the core blocks as **213**1, 3**12**0, 2**03**2, 3**2102**0, 2**02**1, 2**13**1, 3**12**0, 2**03**2, 3**2102**.

Observe that the (i) two occurrences of string X are partitioned by LCP the same way except in the margins. Further observe that (ii) if a string is identified as a core block in a particular location, it must be identified as a core block elsewhere due to the fact that all symbols that lead LCP to identify that block as a core block are included in the core block. As a result (iii) all core blocks that entirely reside in one occurrence of X should

be identical to those that reside in another occurrence of X. Finally observe that (iv) the number of cores that reside in any substring X is at most $1/2$ of its length and at least $1/k$ of its length.

The above version of LCP can return core blocks with length as small as 4; a length 4 substring is clearly not specific enough for clustering an HTS read; we have to ensure that the minimum core block length c is a substantial fraction of the read length. LCP as described in [101, 102, 9] enables to partition S into non-overlapping blocks of size at least c and at most $2c - 1$ for any user defined c. These blocks can be extended by a constant number of symbols to the right and to the left to obtain the "core" blocks of S. In the context of compressing HTS reads, if c is picked to be a significantly long fraction of the read size, LCP, applied on the HTS reads will guarantee that each read will include at least one and at most three of these core blocks.

Unfortunately this general version of LCP is too complex to be of practical interest. As a result we have developed a practical variant of LCP described below to obtain core blocks of each HTS read with minimum length 8 and maximum length 20. Interestingly we observed that in practice more than 99% of all HTS reads of length 50 or more include at least one core of length 14 or less. As a result, we are interested in identifying only those core blocks of lengths in the range $[8, 14]$. Still there could be multiple such core blocks in each HTS read; SCALCE will pick the longest one as the *representative core block* of the read (if there are more than one such block, SCALCE may break the tie in any consistent way). SCALCE will then cluster this read with other reads that have the same representative core block.

7.1.3 A practical implementation of LCP for reordering reads

The purpose of reordering reads is to group highly related reads, in fact those reads that ideally come from the same region and have large overlaps together so as to boost gzip and other Lempel-Ziv-77 based compression methods. If one concatenates reads from a donor genome in an arbitrary order, highly similar reads will be scattered over the resulting string. Because Lempel-Ziv-77 based techniques compress the input string iteratively, from left to right, replacing the longest possible prefix of the uncompressed portion of the input string with a pointer to its earlier (already compressed) occurrence, as the distance between the two occurrences of this substring to be compressed increases, the binary representation of the pointer also increases. As a result gzip and other variants only search for occurrences of strings within a relatively small window. Thus reordering reads so as to bring together those

with large (suffix-prefix) overlaps is highly beneficial to gzip and other similar compression methods. For this purpose, it is possible to reorder the reads by sorting them based on their mapping loci on the reference genome. Alternatively it may be possible to find similarities between the reads through pairwise comparisons [122]. However each one of these approaches are time-wise costly.

In contrast our goal here is to obtain a few core blocks for each read so that two highly overlapping reads will have common core blocks. The reads will be reordered based on their common core blocks, which satisfy the following properties: (i) Each HTS read includes at least one core block. (ii) Each HTS read includes at most a small number of core blocks. This would be achieved if any sufficiently "long" prefix of a core block can not be a suffix of another core block (this assures that two subsequent core blocks can not be too close to each other).

We first extend the simple variant of LCP described above so as to handle strings from the alphabet $\Sigma = \{0, 1, 2, 3\}$ (0=A, 1=C, 2=G, 3=T) that *can* include tandemly repeated blocks. In this variant we define a core block as any 4-mer that satisfies one of the following rules:

- (Local Maxima) $xyzw$ where $x < y$ and $z < y$;
- (Low Periodicity) $xyyz$ where $x \neq y$ and $z \neq y$;
- (Lack of Maxima) $xyzw$ where $x \neq y$ and $y < z < w$;
- (Periodic Substrings) $yyyx$ where $x \neq y$.

We computed all possible 4-mers (there are 256 of them) from the 4 letter alphabet Σ and obtained 116 core blocks that satisfy the rules above. The reader can observe that the minimum distance between any two neighbouring cores will be 2 and the maximum possible distance will be 6 (note that this implementation of LCP is not aimed to satisfy any theoretical guarantee; rather, it is developed to work well in practice). This ensures that any read of length at least 9 includes one such core block.

To capture longer regions of similarity between reads, we need to increase the lengths of core blocks. For that purpose we first identify the so called marker symbols in the read processed as follows. Let x, y, z, w, x, $v \in \Sigma$, then:

- y is a "marker" for xyz, when $x < y$ and $z < y$;

- y is a "marker" for $xyyz$, when $x < y$ and $z < y$;
- y is a "marker" for $xyyyz$, when $x \neq y$ and $z \neq y$;
- yy is a "marker" for $xyyyyz$, when $x \neq y$ and $z \neq y$;
- y is a "marker" for $xwyzv$, when $y < w \leq x$ and $y < z \leq v$.

Now on a given read, we first identify all marker symbols. We apply LCP to the sequence obtained by concatenating these marker symbols to obtain the core blocks of the marker symbols. We then map these core blocks of the marker symbols to the original symbols to obtain the core blocks of the original read. Given read $R = 0230000300$, we identify its marker symbols as follows: 3 is the marker for 230, 00 is the marker for 300003, and 3 is the marker for 030 as per the marker identification rules above. The sequence obtained by concatenating these markers is 3003, which is itself (4-mer) core block according to the LCP description above. The projection of this core block on R is 23000030, which is thus identified as a core block (actually the only core block) of the read.

For the 4 letter alphabet Σ, we computed all ($\approx$ 5 million) possible core blocks of length $\{8, \ldots, 14\}$ according to the above rules (this is about 1% of all blocks in this length range). These rules assure that the minimum distance between two subsequent core blocks is 4 and thus the maximum number of core blocks per read is at most 11 per each HTS read of length 50. Furthermore we observed that more than 99.5% of all reads have at least one core block (the other reads have all cores of length 15 to 20). Although this guarantee is weaker than the theoretical guarantee provided by the most general version of LCP, it serves our purposes.

7.1.4 A data structure for identifying core substrings of reads

We build a trie [32] data structure representing each possible core substring by a path to efficiently place reads into "buckets". We find "all" core substrings of each read and place the read in the bucket associated with the core substring that is the longest (if there are two or more such buckets, we pick the one that contains the maximum number of reads). If one simply uses the trie data structure, finding all core substrings within a read would require $O(cr)$ time where r is the read length, and c is the length of all core substrings in that read. To improve the running time we build an automaton implementing the Aho-Corasick dictionary matching algorithm [3]. This improves the running time to $O(r + k)$, where k

is the number of core substring occurrences in each read. Because the size of the alphabet Σ is very small (4 symbols), and the number of the core substrings is fixed, we can further improve the running time by pre-processing the automaton such that, for a given state of the automaton we calculate the associated bucket in $O(1)$ time, reducing the total search time to $O(r)$.

This is done by analysing the original Aho-Corasick trie and redirecting the backward (also called failure) edges to the ultimate source edges (and thus reducing the number of steps for failure look-up), as shown in Figure 7.1.

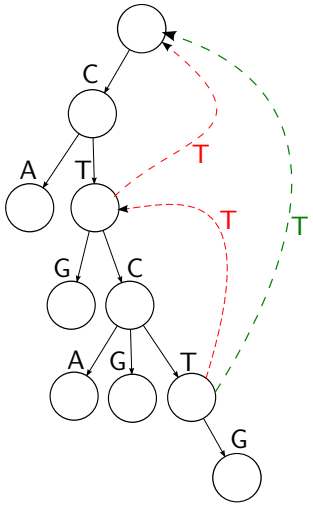

Figure 7.1: Aho-Corasick trie preprocessing. Red edges indicate original failure edges, while green edges indicate preprocessed failure edges. Note that we need to perform only one jump by using green edges to reach the destination, contrary to the two jumps needed by red edges.

In order to improve the compression of the reads even more, we apply cyclic shifting based on the core position in each read, and then sort each bucket alphabetically, in order to boost the similarity between the neighbouring sequences even more.

7.1.5 Compressing the quality scores

Note that the HTS platforms generate additional information for each read that is not confined to the 4 letter alphabet Σ. Each read is associated with a secondary string that contains the base calling *Phred* [24] quality score. Quality score of a base defines the probability that the base call is incorrect, and it is formulated as $Q = -10 \times \log_{10}(P(error))$ [24]. The size of the alphabet for the quality scores is typically $|\Sigma| = 40$ for the Illumina platform, thus the compression rate for quality scores is lower than the actual reads. As mentioned in previous studies [114], lossy compression can improve the quality scores compression rate. We provide an optional controlled lossy transformation approach based on the following observation. In most cases, for any base pair b, the quality scores of its "neighbouring" base pairs would be either the same or within some small range of b's score (see Figure 7.2). Based on this observation, we provide a lossy transformation scheme to reduce the alphabet size. We calculate the frequency table for the alphabet of quality scores from a reasonable subset of the qualities (1 million quality scores). We first use a simple greedy algorithm to find the local maxima within this table. We then reduce the variability among the quality scores in the vicinity of local maxima up to some error threshold e.

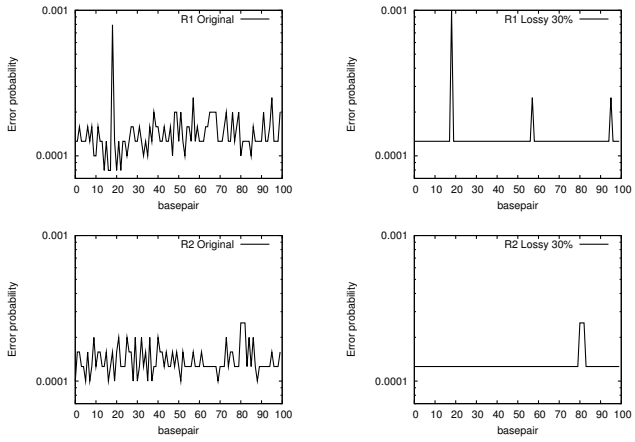

Figure 7.2: Original (left) and transformed (right) quality scores for two random reads that are chosen from NA18507 individual. The original scores show much variance, where the transformed quality scores are smoothened except for the peaks at local maxima, that help to improve the compression ratio.

Table 7.1: Input data statistics and compression rates achieved by gzip only and SCALCE + gzip on reads from the P. aeruginosa RNA-Seq library (dataset 1).

Dataset Info	**Size**	**4,327**
	Number of Reads	89 million
gzip	**Size**	**1,071**
	Compression Rate	4.04
	Time	13m 18s
SCALCE+gzip	**Size**	**256**
	Compression Rate	16.92
	SCALCE+gzip time	6m 21s
	Boosting Factor	4.19x
	gzip time after reordering	53s

File sizes are reported in megabytes.

7.2 Results

We evaluated the performance of the SCALCE algorithm for boosting gzip on a single core 2.4GHz Intel Xeon X5690 PC (with network storage and 6GB of memory).

We used four different data sets in our tests:
(i) P. aeruginosa RNA-Seq library (51 bp, single lane), (ii) P. aeruginosa genomic sequence library (51 bp, single lane). (iii) whole genome shotgun sequencing (WGS) library generated from the genome of the HapMap individual NA18507 (100 bp reads at 40X genome coverage), and (iv) a single lane from the same human WGS data set corresponding to approximately 1.22X genome coverage (Sequence Read Archive ID: SRR034940). We removed any comments from name section (any string that appears after the first space). Also the third row should contain a single character (+/-) separator character.

The reads from each data set were reordered through SCALCE and three separate files were obtained for (i) the reads themselves, (ii) the quality scores and (iii) the read names (each maintaining the same order). Note that LCP reordering is useful primarily for compressing the reads themselves through gzip. The quality scores were compressed via the scheme described above. Finally the read names were compressed through gzip as well.

The compression rate and run time achieved by gzip software alone, only on the reads from the P. aeruginosa RNA-Seq library (dataset 1) is compared against those achieved by

Table 7.2: Input data statistics and compression rates achieved by gzip only and SCALCE + gzip + AC on complete FASTQ files.

	P. aeruginosa RNAseq	P. aeruginosa Genomic	NA18507 WGS	NA18507 Single Lane
Original size	**10,076**	**9,163**	**300,337**	**7,708**
Number of reads	89 million	81 million	1.4 billion	36 million
gzip Performance				
Size	**3,183**	**3,211**	**113,132**	**3,058**
Rate	3.17	2.85	2.65	2.52
SCALCE Performance with Lossless Qualities				
Size	**1,496**	**1,655**	**76,890**	**2,146**
Rate	6.74	5.54	3.91	3.59
Boosting factor	2.13x	1.94x	1.47x	1.42x
SCALCE Performance with 30% Lossy Setting Applied				
Size	**953**	**1,126**	**58,031**	**1,639**
Rate	10.58	8.14	5.18	4.70
Boosting factor	3.34x	2.85x	1.95x	1.86x

File sizes are reported in megabytes.

SCALCE followed by gzip in Table 7.1. The compression rates achieved by the gzip software alone in comparison with gzip following SCALCE on the combination of reads, quality scores and read names are presented in Table 7.2. The run times for the two schemes (again on reads, quality scores and read names all together) are presented in Table 7.3.

When SCALCE is used with arithmetical coding of order 3 with lossless qualities, it boosts the compression rate of gzip between $1.42-2.13$-fold (when applied to reads, quality scores and read names), significantly reducing the storage requirements for HTS data. When arithmetical coding of order 3 is used with 30% loss – without reducing the mapping accuracy – improvements in compression rate are between $1.86-3.34$. In fact, the boosting factor can go up to 4.19 when compressing the reads only. Moreover, the speed of the gzip compression step can be improved by a factor of 15.06. Interestingly the total run time for SCALCE + gzip is less than the run time of gzip by a factor of 2.09. Furthermore, users can tune the

Table 7.3: Run time for running gzip alone and SCALCE+gzip+AC on complete FASTQ files.

	P. aeruginosa RNAseq	P. aeruginosa Genomic	NA18507 WGS	NA18507 Single Lane
gzip Timing				
Total	**20m**	**20m**	**10h 52m**	**18m**
SCALCE Timing with single thread				
SCALCE reordering	7m	6m	3h	5m
gzip+AC	6m	5m	3h 1m	5m
Total	**13m**	**11m**	**6h 1m**	**10m**
SCALCE Timing with 3 threads				
Total	**9m**	**9m**	**4h 28m**	**7m 32s**

memory available to SCALCE through a parameter to improve the run time when a large main memory is available. In our tests, we limited the memory usage to 6GB.

Note that our goal here is to devise a very fast boosting method, SCALCE, which, in combination with gzip gives compression rates much better than gzip alone. It is possible to get better compression rates through mapping based strategies but these methods are several orders of magnitude slower than SCALCE+gzip. We tested the effects of the lossy compression schemes for the quality scores, used by SCALCE as well as CRAM tools, to single nucleotide polymorphism (SNP) discovery. For that, we first mapped the NA18507 WGS data set with the original quality values to the human reference genome (GRCh37) using the BWA aligner [68], and called SNPs using the GATK software [20]. We repeated the same exercise with the reads after 30% lossy transformation of the base pair qualities with SCALCE. Note that the parameters for BWA and GATK we used in these experiments were exactly the same. We observed almost perfect correspondence between two experiments. In fact, $> 99.95\%$ of the discovered SNPs were the same (Table 7.6); not surprisingly most of the difference was due to SNPs in mapping to common repeats or segmental duplications. We then compared the differences of both SNP call sets with dbSNP Release 132 [106] in Table 7.6.

Table 7.4: Comparison of single-threaded SCALCE with DSRC.

Name	DSRC		SCALCE	
	Time	Size	Time	Size
P. aeruginosa RNAseq	12m	1,767	13m	1,496
P. aeruginosa Genomic	6m	1,846	11m	1,655
NA18507 WGS[a]	3h 16m	94,707	6h 1m	76,890
NA18507 Single Lane	4m	2,341	10m	2,146

DSRC was tested using the -l option. This option provides better compression ratio but it is slower. [a] DSRC crashed while using -l option. Instead we used a faster but less powerful setting for this data set.

Table 7.5: Comparison of single-threaded SCALCE with BEETL.

Name	BEETL		SCALCE	
	Time	Size	Time	Size
P. aeruginosa RNAseq	29m	197	8m	95
P. aeruginosa Genomic	31m	257	6m	137
NA18507 Single Lane	51m	448	10m	412

Here, the data sets contained only reads from the FASTQ file, as BEETL supports only FASTA file format.

In addition, we carried out the same experiment with compressing/decompressing of the alignments with CRAM tools. As shown in Table 7.6, quality transformation of the CRAM tools introduced about 2.5% errors in SNP calling (97.5% accuracy) with respect to the calls made for the original data (set as the gold standard).

One interesting observation is that 70.7% of the new calls after SCALCE processing matched to entries in dbSNP where this ratio was only 62.75% for the new calls after CRAM tools quality transformation. Moreover, 57.95% of the SNPs that SCALCE "lost" are found in dbSNP, and CRAM tools processing caused removal of 18.4 times more potentially real SNPs than SCALCE.

As a final benchmark, we compared the performance of SCALCE with mapping based reordering before gzip compression. We first mapped one lane of sequence data from the genome of NA18507 (same as above) to human reference genome (GRCh37) using BWA [68],

Table 7.6: Number of SNPs found in the NA18507 genome using original qualities and transformed qualities with 30% noise reduction. Also reported are the number and percentage of novel SNPs in regions of segmental duplication or common repeats (SD+CR).

		# SNP	dbSNP v132 (%)	Novel	
Qualities		Count		Total	in SD+CR (%)
Original	Total	4,296,152	4,092,923 (95.26)	203,229	192,114 (94.53)
SCALCE	Total	4,303,140	4,098,875 (95.25)	204,265	192,976 (94.47)
	Lost	7,931	4,596 (57.95)	3,335	2,963 (88.84)
	New	14,919	10,548 (70.70)	4,371	3,825 (87.51)
CRAM	Total	4,202,298	4,013,401 (95.50)	188,897	179,875 (95.22)
	Lost	101,957	84,607 (82.98)	17,350	15,036 (86.66)
	New	8,103	5,085 (62.75)	3,018	2,797 (92.67)

and sorted the mapped reads using samtools [70], and reconverted the map-sorted BAM file back to FASTQ using Picard (`http://picard.sourceforge.net`). We then used the gzip tool to compress the map-sorted file to $3,091.5$ MB, achieving 2.49-fold compression rate. The preprocessing step for mapping and sorting required 18.2 CPU hours, and FASTQ conversion required 30 minutes, whereas compression was completed in 28 minutes. Moreover, the mapping based sorting did not improve the compression run time even if we do not factor in the preprocessing. In contrast, SCALCE+gzip+AC generated a much smaller file in less amount of time, with no mapping based preprocessing. We then repeated this experiment on the entire WGS data set (NA18507). The mapping based preprocessing took 700 CPU hours for BWA+samtools, and 10 CPU hours for Picard, whereas gzip step was completed in 11 CPU hours, resulting in a compression rate of 3.1x. On the other hand, gzip needed only 3 CPU hours to compress the same data set (3.67x faster) after the preprocessing by SCALCE, which took 3 CPU hours, and achieved a better compression rate (Tables 7.2 and 7.3). The run time of mapping based preprocessing step can be improved slightly through the use of BAM-file-based compressors such as CRAM tools [50], but this would reduce the time only by 10 CPU hours for the Picard step. Thus, in total, SCALCE+gzip is about 120 times faster than any potential mapping based scheme (including CRAM tools) on this data set.

Our tests showed that SCALCE (when considering only reads) outperforms BEETL [18]

combined with bzip2 by a factor between $1.09 - 2.07$, where running time is improved by a factor between $3.60 - 5.17$ (see Table 7.5). SCALCE (on full FASTQ files) also outperforms DSRC [19] compression ratio on complete FASTQ files by a factor between $1.09 - 1.18$ (see Table 7.4).

7.3 Conclusion

The rate of increase in the amount of data produced by the HTS technologies is now faster than the Moore's Law [4]. This causes problems related to both data storage and transfer of data over a network. Traditional compression tools such as gzip and bzip2 are not optimized for efficiently reducing the files to manageable sizes in short amount of time. To address this issue several compression techniques have been developed with different strengths and limitations. For example pairwise comparison of sequences can be used to increase similarity within "chunks" of data, thus increasing compression ratio [122], but this approach is also very time consuming. Alternatively, reference-based methods can be used such as SlimGene [61] and CRAM tools [50]. Although these algorithms achieve very high compression rates, they have three major shortcomings. First, they require pre-mapped (and sorted) reads along with a reference genome, and this mapping stage can take very long time depending on the size of the reference genome. Second, speed and compression ratio are highly dependent on the mapping ratio since the unmapped reads are handled in a more costly manner (or completely discarded), which reduces the efficiency for genomes with high novel sequence insertions and organisms with incomplete reference genomes. Finally, the requirement of a reference sequence makes them unusable for *de novo* sequencing projects of the genomes of organisms where no such reference is available, for example, the Genome 10K Project [42].

The SCALCE algorithm provides a new and efficient way of reordering reads generated by the HTS platform to improve not only compression rate but also compression run time. We note that the names associated with each read do not have any specific information and they can be discarded during compression. The only consideration here is that during decompression, new read names will need to be generated. These names need to be unique identifiers within a sequencing experiment, and the paired-end information must be easy to track. In fact, the Sequence Read Archive (SRA) developed by the International Nucleotide Sequence Database Collaboration adopts this approach to minimize the stored

metadata, together with a lossy transformation of the base pair quality values similar to our approach [59]. However, in this chapter we demonstrated that lossy compression of quality affects the analysis result, and although the difference is very small for SCALCE, this is an optional parameter in our implementation, and we leave the decision to the user. Additional improvements in compression efficiency and speed may help ameliorate the data storage and management problems associated with high throughput sequencing [105].

Chapter 8

Conclusion

High Throughput Sequencing Technologies have revolutionized the way genomic research is being conducted. These platforms generate unprecedented amounts of data that introduce many challenges for processing, downstream analysis and infrastructure. In addition to single nucleotide variations and small insertions-deletions (indels), larger size structural variations (for example, insertions, deletions, inversions, segmental duplications and copy-number polymorphism) contribute significantly to human genetic diversity. In almost all recent structural variation discovery studies, short reads from a donor genome have been mapped to a reference genome as a first step. The accuracy of such an structural variation discovery study is directly correlated to the accuracy of this mapping step, which also provides the main computational bottleneck of the structural variation detection study.

This thesis presents fast and efficient algorithms to map reads that are generated by HTS platforms with the consideration that the mappings will be used in structural variation discovery studies. We also propose an approach on how to compress and deal with the unprecedented amount of data that is being generated by these platforms.

We first presented an overview of available methods for short read mapping and compression of HTS data. Then, we introduced mrsFAST, our cache oblivious read mapping tool that is specifically designed to deal with Illumina generated data. mrsFAST is mathematically guaranteed to find and report "all" mapping locations for a given collection of HTS reads. We showed that the number of cache misses by mrsFAST, at any level of the cache hierarchy, is optimal among all families of seed-and-extend algorithms for mapping, within a factor of 2 in the worst case. We also showed that mrsFAST is highly sensitive and fast in comparison to many popular tools.

Later in mrsFAST-ultra, we further investigated the search step of the mapping algorithm and introduced two filters in order to decrease the false positive candidates during the mapping process. We showed that the introduction of these two filters and also use of compact data structures can improve the performance of mrsFAST by **five**-fold.

We also introduced drFAST, a mapping algorithm for di-base encoded reads produced by AB SOLiD platforms. We showed that simple conversion of the color space reads to base space reads cannot provide a correct way to do the mapping. We provided two algorithms to do the mapping of color space reads to the reference genome namely "dynamic programming" and "color transformation" methods.

We also consider the case that the length of HTS reads will increase significantly covering the whole transcriptome. We introduce two algorithms to do transcriptome to genome mapping while considering the structural alterations such as duplications, inversions rearrangements and fusions. The first algorithm is based on nucleotide level alignment which is slow and more sensitive. It can detect structural alterations by using a single unified dynamic programming. The second algorithm was faster and less sensitive based on chaining fragments shared between transcriptome and genome.

Finally, we proposed a booster algorithm, SCALCE, for HTS data compression. General purpose tools (i.e. gzip) are not optimized for HTS data. We introduce an algorithm to pre-process and re-order the HTS reads based on Locally Consistent Parsing scheme in order to increase the locality of the data. This re-ordering boosted HTS data compression significantly both in time and space. This scheme coupled with arithmetic coding for quality scores provided a significant improvement over all the existing tools.

8.1 Future Directions

Continuous development and validation of novel algorithms for discovering genetic variation will be essential for the analysis of the growing volume of data generated by the High Throughput Sequencing platforms. As HTS reads get longer, the sequencing error related to these data increases as well. This requires fast and efficient algorithms that can tolerate higher errors as well as more complex events in addition to indels and substitutions.

As HTS platforms advances, the throughput of these platforms increase rapidly. It is essential to replace the current general purpose tools with specialized HTS compression tools. These tools should be faster than the general purpose tools, and they should provide better

compression factors. One of the important piece of information that should be compressed is quality scores. Finding methods that can compress this data effectively will reduce the storage requirements for HTS platforms significantly.

Bibliography

[1] 1000 Genomes Project Consortium. A map of human genome variation from population-scale sequencing. *Nature 467*, 7319 (Oct 2010), 1061–1073.

[2] 1000 Genomes Project Consortium. An integrated map of genetic variation from 1,092 human genomes. *Nature 491*, 7422 (Nov 2012), 56–65.

[3] Aho, A. V., and Corasick, M. J. Efficient String Matching: an Aid to Bibliographic Search. *Commun. ACM 18*, 6 (1975), 333–340.

[4] Alkan, C., Coe, B. P., and Eichler, E. E. Genome structural variation discovery and genotyping. *Nature Reviews. Genetics 12*, 5 (May 2011), 363–376.

[5] Alkan, C., Kidd, J. M., Marques-Bonet, T., Aksay, G., Antonacci, F., Hormozdiari, F., Kitzman, J. O., Baker, C., Malig, M., Mutlu, O., Sahinalp, S. C., Gibbs, R. A., and Eichler, E. E. Personalized copy number and segmental duplication maps using next-generation sequencing. *Nature Genetics 41*, 10 (2009), 1061–1067.

[6] Altschul, S. F., Madden, T. L., Schffer, A. A., Zhang, J., Zhang, Z., Miller, W., and Lipman, D. J. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research 25*, 17 (Sep 1997), 3389–3402.

[7] Armstrong, M. Groups and Symmetry. In *Springer Verlag* (1988), p. 53.

[8] Asmann, Y. W., Hossain, A., Necela, B. M., Middha, S., Kalari, K. R., Sun, Z., Chai, H.-S., Williamson, D. W., Radisky, D., Schroth, G. P., Kocher, J.-P. A., Perez, E. A., and Thompson, E. A. A novel bioinformatics pipeline for identification and characterization of fusion transcripts in breast cancer and normal cell lines. *Nucleic Acids Research 39*, 15 (Aug 2011), e100.

[9] Batu, T., Ergün, F., and Sahinalp, S. C. Oblivious string embeddings and edit distance approximations. In *SODA 2006* (2006), pp. 792–801.

[10] Bentley, D. R., Balasubramanian, S., Swerdlow, H. P., Smith, G. P., Milton, J., Brown, C. G., Hall, K. P., Evers, D. J., Barnes, C. L., Bignell,

H. R., Boutell, J. M., Bryant, J., Carter, R. J., Cheetham, R. K., Cox, A. J., Ellis, D. J., Flatbush, M. R., Gormley, N. A., Humphray, S. J., Irving, L. J., Karbelashvili, M. S., Kirk, S. M., Li, H., Liu, X., Maisinger, K. S., Murray, L. J., Obradovic, B., Ost, T., Parkinson, M. L., Pratt, M. R., Rasolonjatovo, I. M. J., Reed, M. T., Rigatti, R., Rodighiero, C., Ross, M. T., Sabot, A., Sankar, S. V., Scally, A., Schroth, G. P., Smith, M. E., Smith, V. P., Spiridou, A., Torrance, P. E., Tzonev, S. S., Vermaas, E. H., Walter, K., Wu, X., Zhang, L., Alam, M. D., Anastasi, C., Aniebo, I. C., Bailey, D. M. D., Bancarz, I. R., Banerjee, S., Barbour, S. G., Baybayan, P. A., Benoit, V. A., Benson, K. F., Bevis, C., Black, P. J., Boodhun, A., Brennan, J. S., Bridgham, J. A., Brown, R. C., Brown, A. A., Buermann, D. H., Bundu, A. A., Burrows, J. C., Carter, N. P., Castillo, N., Catenazzi, M. C. E., Chang, S., Cooley, R. N., Crake, N. R., Dada, O. O., Diakoumakos, K. D., Dominguez-Fernandez, B., Earnshaw, D. J., Egbujor, U. C., Elmore, D. W., Etchin, S. S., Ewan, M. R., Fedurco, M., Fraser, L. J., Fajardo, K. V. F., Furey, W. S., George, D., Gietzen, K. J., Goddard, C. P., Golda, G. S., Granieri, P. A., Green, D. E., Gustafson, D. L., Hansen, N. F., Harnish, K., Haudenschild, C. D., Heyer, N. I., Hims, M. M., Ho, J. T., Horgan, A. M., Hoschler, K., Hurwitz, S., Ivanov, D. V., Johnson, M. Q., James, T., Jones, T. A. H., Kang, G.-D., Kerelska, T. H., Kersey, A. D., Khrebtukova, I., Kindwall, A. P., Kingsbury, Z., Kokko-Gonzales, P. I., Kumar, A., Laurent, M. A., Lawley, C. T., Lee, S. E., Lee, X., Liao, A. K., Loch, J. A., Lok, M., Luo, S., Mammen, R. M., Martin, J. W., McCauley, P. G., McNitt, P., Mehta, P., Moon, K. W., Mullens, J. W., Newington, T., Ning, Z., Ng, B. L., Novo, S. M., O'Neill, M. J., Osborne, M. A., Osnowski, A., Ostadan, O., Paraschos, L. L., Pickering, L., Pike, A. C., Pike, A. C., Pinkard, D. C., Pliskin, D. P., Podhasky, J., Quijano, V. J., Raczy, C., Rae, V. H., Rawlings, S. R., Rodriguez, A. C., Roe, P. M., Rogers, J., Bacigalupo, M. C. R., Romanov, N., Romieu, A., Roth, R. K., Rourke, N. J., Ruediger, S. T., Rusman, E., Sanches-Kuiper, R. M., Schenker, M. R., Seoane, J. M., Shaw, R. J., Shiver, M. K., Short, S. W., Sizto, N. L., Sluis, J. P., Smith, M. A., Sohna, J. E. S., Spence, E. J., Stevens, K., Sutton, N., Szajkowski, L., Tregidgo, C. L., Turcatti, G., Vandevondele, S., Verhovsky, Y., Virk, S. M., Wakelin, S., Walcott, G. C., Wang, J., Worsley, G. J., Yan, J., Yau, L., Zuerlein, M., Rogers, J., Mullikin, J. C., Hurles, M. E., McCooke, N. J., West, J. S., Oaks, F. L., Lundberg, P. L., Klenerman, D., Durbin, R., and Smith, A. J. Accurate whole human genome sequencing using reversible terminator chemistry. *Nature 456*, 7218 (Nov 2008), 53–59.

[11] Birol, I., Jackman, S. D., Nielsen, C. B., Qian, J. Q., Varhol, R., Stazyk, G., Morin, R. D., Zhao, Y., Hirst, M., Schein, J. E., Horsman, D. E., Connors, J. M., Gascoyne, R. D., Marra, M. A., and Jones, S. J. M. De

novo transcriptome assembly with ABySS. *Bioinformatics 25*, 21 (Nov 2009), 2872–2877.

[12] Bonfield, J. K., and Mahoney, M. V. Compression of FASTQ and SAM Format Sequencing Data. *PLoS ONE 8*, 3 (2013), e59190.

[13] Brudno, M., Malde, S., Poliakov, A., Do, C. B., Couronne, O., Dubchak, I., and Batzoglou, S. Glocal alignment: finding rearrangements during alignment. *Bioinformatics 19 Suppl 1* (2003), i54–i62.

[14] Burge, C., and Karlin, S. Prediction of complete gene structures in human genomic DNA. *Journal of Molecular Biology 268*, 1 (Apr 1997), 78–94.

[15] Burrows, M., and Wheeler, D. A block sorting lossless data compression algorithm. Tech. rep., Hewlett Packard, 1994.

[16] Chen, K., Wallis, J. W., McLellan, M. D., Larson, D. E., Kalicki, J. M., Pohl, C. S., McGrath, S. D., Wendl, M. C., Zhang, Q., Locke, D. P., Shi, X., Fulton, R. S., Ley, T. J., Wilson, R. K., Ding, L., and Mardis, E. R. BreakDancer: an algorithm for high-resolution mapping of genomic structural variation. *Nature Methods 6*, 9 (Aug 2009), 677–681.

[17] Cormode, G., Paterson, M., Sahinalp, S. C., and Vishkin, U. Communication complexity of document exchange. In *SODA 2000* (2000), pp. 197–206.

[18] Cox, A. J., Bauer, M. J., Jakobi, T., and Rosone, G. Large-scale compression of genomic sequence databases with the Burrows-Wheeler transform. *Bioinformatics 28*, 11 (Jun 2012), 1415–1419.

[19] Deorowicz, S., and Grabowski, S. Compression of DNA sequence reads in FASTQ format. *Bioinformatics 27*, 6 (Mar 2011), 860–862.

[20] DePristo, M. A., Banks, E., Poplin, R., Garimella, K. V., Maguire, J. R., Hartl, C., Philippakis, A. A., del Angel, G., Rivas, M. A., Hanna, M., McKenna, A., Fennell, T. J., Kernytsky, A. M., Sivachenko, A. Y., Cibulskis, K., Gabriel, S. B., Altshuler, D., and Daly, M. J. A framework for variation discovery and genotyping using next-generation DNA sequencing data. *Nature Genetics 43* (May 2011), 491–498.

[21] Eid, J., Fehr, A., Gray, J., Luong, K., Lyle, J., Otto, G., Peluso, P., Rank, D., Baybayan, P., Bettman, B., Bibillo, A., Bjornson, K., Chaudhuri, B., Christians, F., Cicero, R., Clark, S., Dalal, R., Dewinter, A., Dixon, J., Foquet, M., Gaertner, A., Hardenbol, P., Heiner, C., Hester, K., Holden, D., Kearns, G., Kong, X., Kuse, R., Lacroix, Y., Lin, S., Lundquist, P., Ma, C., Marks, P., Maxham, M., Murphy, D., Park, I., Pham, T., Phillips, M., Roy, J., Sebra, R., Shen, G., Sorenson, J., Tomaney, A., Travers, K.,

Trulson, M., Vieceli, J., Wegener, J., Wu, D., Yang, A., Zaccarin, D., Zhao, P., Zhong, F., Korlach, J., and Turner, S. Real-time DNA sequencing from single polymerase molecules. *Science 323*, 5910 (Jan 2009), 133–138.

[22] Elias, P. Universal codeword sets and representations of the integers. *Information Theory, IEEE Transactions on 21*, 2 (1975), 194–203.

[23] Ergün, F., Muthukrishnan, S., and Sahinalp, S. C. Comparing Sequences with Segment Rearrangements. In *FSTTCS* (2003), pp. 183–194.

[24] Ewing, B., and Green, P. Base-calling of automated sequencer traces using phred. II. Error probabilities. *Genome Research 8* (Mar 1998), 186–194.

[25] Farrar, M. Striped Smith-Waterman speeds database searches six times over other SIMD implementations. *Bioinformatics 23*, 2 (Jan 2007), 156–161.

[26] Faust, G. G., and Hall, I. M. YAHA: fast and flexible long-read alignment with optimal breakpoint detection. *Bioinformatics 28*, 19 (Oct 2012), 2417–2424.

[27] Ferragina, P., Giancarlo, R., and Manzini, G. The Engineering of a Compression Boosting Library: Theory vs Practice in BWT Compression. In *ESA 2006* (2006), pp. 756–767.

[28] Ferragina, P., Giancarlo, R., Manzini, G., and Sciortino, M. Boosting textual compression in optimal linear time. *J. ACM 52*, 4 (2005), 688–713.

[29] Ferragina, P., and Manzini, G. Opportunistic Data Structures with Applications. In *FOCS* (2000), pp. 390–398.

[30] Ferragina, P., and Manzini, G. Compression boosting in optimal linear time using the Burrows-Wheeler Transform. In *SODA 2004* (2004), pp. 655–663.

[31] Fonseca, N. A., Rung, J., Brazma, A., and Marioni, J. C. Tools for mapping high-throughput sequencing data. *Bioinformatics 28*, 24 (Dec 2012), 3169–3177.

[32] Fredkin, E. Trie memory. *Commun. ACM 3*, 9 (Sept. 1960), 490–499.

[33] Frigo, M., Leiserson, C. E., Prokop, H., and Ramachandran, S. Cache-Oblivious Algorithms. In *40th Annual Symposium on Foundations of Computer Science* (New York, New York, Oct. 17–19 1999), pp. 285–297.

[34] Galil, Z., and Giancarlo, R. Speeding up dynamic programming with applications to molecular biology. *Theor. Comput. Sci. 64* (April 1989), 107–118.

[35] Ge, H., Liu, K., Juan, T., Fang, F., Newman, M., and Hoeck, W. FusionMap: detecting fusion genes from next-generation sequencing data at base-pair resolution. *Bioinformatics 27*, 14 (Jul 2011), 1922–1928.

[36] Gnerre, S., Maccallum, I., Przybylski, D., Ribeiro, F. J., Burton, J. N., Walker, B. J., Sharpe, T., Hall, G., Shea, T. P., Sykes, S., Berlin, A. M., Aird, D., Costello, M., Daza, R., Williams, L., Nicol, R., Gnirke, A., Nusbaum, C., Lander, E. S., and Jaffe, D. B. High-quality draft assemblies of mammalian genomes from massively parallel sequence data. *Proc. Natl. Acad. Sci. U.S.A. 108*, 4 (Jan 2011), 1513–1518.

[37] Golomb, S. W. Run-Length Encodings. *IEEE Transactions on Information Theory 12* (September 1966), 399–401.

[38] Gontarz, P. M., Berger, J., and Wong, C. F. SRmapper: a fast and sensitive genome-hashing alignment tool. *Bioinformatics 29*, 3 (Feb 2013), 316–321.

[39] Hach, F., Hormozdiari, F., Alkan, C., Hormozdiari, F., Birol, I., Eichler, E. E., and Sahinalp, S. C. mrsFAST: a cache-oblivious algorithm for short-read mapping. *Nature Methods 7*, 8 (2010), 576–577.

[40] Hach, F., Numanagic, I., Alkan, C., and Sahinalp, S. C. SCALCE: boosting sequence compression algorithms using locally consistent encoding. *Bioinformatics 28*, 23 (2012), 3051–3057.

[41] Hajirasouliha, I., Hormozdiari, F., Alkan, C., Kidd, J. M., Birol, I., Eichler, E. E., and Sahinalp, S. C. Detection and characterization of novel sequence insertions using paired-end next-generation sequencing. *Bioinformatics 26*, 10 (2010), 1277–1283.

[42] Haussler, D., O'Brien, S. J., Ryder, O. A., Barker, F. K., Clamp, M., Crawford, A. J., Hanner, R., Hanotte, O., Johnson, W. E., McGuire, J. A., Miller, W., Murphy, R. W., Murphy, W. J., Sheldon, F. H., Sinervo, B., Venkatesh, B., Wiley, E. O., Allendorf, F. W., Amato, G., Baker, C. S., Bauer, A., Beja-Pereira, A., Bermingham, E., Bernardi, G., Bonvicino, C. R., Brenner, S., Burke, T., Cracraft, J., Diekhans, M., Edwards, S., Ericson, P. G., Estes, J., Fjelsda, J., Flesness, N., Gamble, T., Gaubert, P., Graphodatsky, A. S., Marshall Graves, J. A., Green, E. D., Green, R. E., Hackett, S., Hebert, P., Helgen, K. M., Joseph, L., Kessing, B., Kingsley, D. M., Lewin, H. A., Luikart, G., Martelli, P., Moreira, M. A., Nguyen, N., Orti, G., Pike, B. L., Rawson, D. M., Schuster, S. C., Seuanez, H. N., Shaffer, H. B., Springer, M. S., Stuart, J. M., Sumner, J., Teeling, E., Vrijenhoek, R. C., Ward, R. D., Warren, W. C., Wayne, R., Williams, T. M., Wolfe, N. D., and Zhang, Y. P. Genome 10K: a proposal to obtain whole-genome sequence for 10,000 vertebrate species. *Journal of Heredity 100* (2009), 659–674.

[43] Hillier, L. W., Marth, G. T., Quinlan, A. R., Dooling, D., Fewell, G., Barnett, D., Fox, P., Glasscock, J. I., Hickenbotham, M., Huang, W.,

Magrini, V. J., Richt, R. J., Sander, S. N., Stewart, D. A., Stromberg, M., Tsung, E. F., Wylie, T., Schedl, T., Wilson, R. K., and Mardis, E. R. Whole-genome sequencing and variant discovery in C. elegans. *Nature Methods 5*, 2 (Feb 2008), 183–188.

[44] Homer, N., Merriman, B., and Nelson, S. F. BFAST: An Alignment Tool for Large Scale Genome Resequencing. *PLoS ONE 4*, 11 (2009), 12.

[45] Hon, K., Lam, T., Sadakane, K., Sung, W., and Yiu, S. A Space and Time Efficient Algorithm for Constructing Compressed Suffix Arrays. *Algorithmica 48*, 1 (2007), 23–36.

[46] Hormozdiari, F., Alkan, C., Eichler, E., and Sahinalp, S. Combinatorial Algorithms for Structural Variation Detection in High Throughput Sequenced Genomes. *Genome Research 19*, 7 (2009), 1270–1278.

[47] Hormozdiari, F., Alkan, C., Ventura, M., Hajirasouliha, I., Malig, M., Hach, F., Yorukoglu, D., Dao, P., Bakhshi, M., Sahinalp, S. C., and Eichler, E. E. Alu repeat discovery and characterization within human genomes. *Genome Res. 21*, 6 (Jun 2011), 840–849.

[48] Hormozdiari, F., Hach, F., Sahinalp, S. C., and Alkan, C. Sensitive and fast mapping of di-base encoded reads. *Bioinformatics 27*, 14 (2011), 1915–1921.

[49] Hormozdiari, F., Hajirasouliha, I., Dao, P., Hach, F., Yörükoglu, D., Alkan, C., Eichler, E. E., and Sahinalp, S. C. Next-generation VariationHunter: combinatorial algorithms for transposon insertion discovery. *Bioinformatics [ISMB] 26*, 12 (2010), 350–357.

[50] Hsi-Yang Fritz, M., Leinonen, R., Cochrane, G., and Birney, E. Efficient storage of high throughput DNA sequencing data using reference-based compression. *Genome Research 21*, 5 (May 2011), 734–740.

[51] Hsu, F., Kent, W. J., Clawson, H., Kuhn, R. M., Diekhans, M., and Haussler, D. The UCSC Known Genes. *Bioinformatics 22*, 9 (May 2006), 1036–1046.

[52] Huffman, D. A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the IRE 40*, 9 (Sept. 1952), 1098–1101.

[53] Inaki, K., Hillmer, A. M., Ukil, L., Yao, F., Woo, X. Y., Vardy, L. A., Zawack, K. F. B., Lee, C. W. H., Ariyaratne, P. N., Chan, Y. S., Desai, K. V., Bergh, J., Hall, P., Putti, T. C., Ong, W. L., Shahab, A., Cacheux-Rataboul, V., Karuturi, R. K. M., Sung, W.-K., Ruan, X., Bourque, G., Ruan, Y., and Liu, E. T. Transcriptional consequences of genomic structural aberrations in breast cancer. *Genome Research 21*, 5 (May 2011), 676–687.

[54] Jones, D. C., Ruzzo, W. L., Peng, X., and Katze, M. G. Compression of next-generation sequencing reads aided by highly efficient de novo assembly. *Nucleic Acids Research 40*, 22 (Dec 2012), e171.

[55] Kent, W. J. BLAT–the BLAST-like alignment tool. *Genome Research 12*, 4 (Apr 2002), 656–664.

[56] Kidd, J. M., Cooper, G. M., Donahue, W. F., Hayden, H. S., Sampas, N., Graves, T., Hansen, N., Teague, B., Alkan, C., Antonacci, F., Haugen, E., Zerr, T., Yamada, N. A., Tsang, P., Newman, T. L., Tzn, E., Cheng, Z., Ebling, H. M., Tusneem, N., David, R., Gillett, W., Phelps, K. A., Weaver, M., Saranga, D., Brand, A., Tao, W., Gustafson, E., McKernan, K., Chen, L., Malig, M., Smith, J. D., Korn, J. M., McCarroll, S. A., Altshuler, D. A., Peiffer, D. A., Dorschner, M., Stamatoyannopoulos, J., Schwartz, D., Nickerson, D. A., Mullikin, J. C., Wilson, R. K., Bruhn, L., Olson, M. V., Kaul, R., Smith, D. R., and Eichler, E. E. Mapping and sequencing of structural variation from eight human genomes. *Nature 453*, 7191 (May 2008), 56–64.

[57] Kidd, J. M., Sampas, N., Antonacci, F., Graves, T., Fulton, R., Hayden, H. S., Alkan, C., Malig, M., Ventura, M., Giannuzzi, G., Kallicki, J., Anderson, P., Tsalenko, A., Yamada, N. A., Tsang, P., Kaul, R., Wilson, R. K., Bruhn, L., and Eichler, E. E. Characterization of missing human genome sequences and copy-number polymorphic insertions. *Nature Methods 7*, 5 (May 2010), 365–371.

[58] Kim, D., Pertea, G., Trapnell, C., Pimentel, H., Kelley, R., and Salzberg, S. L. TopHat2: accurate alignment of transcriptomes in the presence of insertions, deletions and gene fusions. *Genome Biology 14*, 4 (Apr 2013), R36.

[59] Kodama, Y., Shumway, M., and Leinonen, R. The sequence read archive: explosive growth of sequencing data. *Nucleic Acids Research 40* (Oct 2011), 56–57.

[60] Korbel, J. O., Urban, A. E., Affourtit, J. P., Godwin, B., Grubert, F., Simons, J. F., Kim, P. M., Palejev, D., Carriero, N. J., Du, L., Taillon, B. E., Chen, Z., Tanzer, A., Saunders, A. C., Chi, J., Yang, F., Carter, N. P., Hurles, M. E., Weissman, S. M., Harkins, T. T., Gerstein, M. B., Egholm, M., and Snyder, M. Paired-End Mapping Reveals Extensive Structural Variation in the Human Genome. *Science 318*, 5849 (2007), 420–426.

[61] Kozanitis, C., Saunders, C., Kruglyak, S., Bafna, V., and Varghese, G. Compressing Genomic Sequence Fragments Using SlimGene. *Journal of Computational Biology 18*, 3 (2011), 401–413.

[62] Langmead, B., and Salzberg, S. L. Fast gapped-read alignment with Bowtie 2. *Nature Methods 9*, 4 (Apr 2012), 357–359.

[63] Langmead, B., Trapnell, C., Pop, M., and Salzberg, S. L. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biology 10*, 3 (2009), R25.

[64] Lapuk, A. V., Wu, C., Wyatt, A. W., McPherson, A., McConeghy, B. J., Brahmbhatt, S., Mo, F., Zoubeidi, A., Anderson, S., Bell, R. H., Haegert, A., Shukin, R., Wang, Y., Fazli, L., Hurtado-Coll, A., Jones, E. C., Hach, F., Hormozdiari, F., Hajirasouliha, I., Boutros, P. C., Bristow, R. G., Zhao, Y., Marra, M. A., Fanjul, A., Maher, C. A., Chinnaiyan, A. M., Rubin, M. A., Beltran, H., Sahinalp, S. C., Gleave, M. E., Volik, S. V., and Collins, C. C. From sequence to molecular pathology, and a mechanism driving the neuroendocrine phenotype in prostate cancer. *J. Pathol. 227*, 3 (Jul 2012), 286–297.

[65] Lee, S., Cheran, E., and Brudno, M. A Robust Framework for Detecting Structural Variations in a Genome. *Bioinformatics 24*, 13 (2008), 59–67.

[66] Lee, S., Hormozdiari, F., Alkan, C., and Brudno, M. MoDIL: Detecting INDEL Variation with Clone-end Sequencing. *Nature Methods 6*, 7 (2009), 473–474.

[67] Levin, J. Z., Berger, M. F., Adiconis, X., Rogov, P., Melnikov, A., Fennell, T., Nusbaum, C., Garraway, L. A., and Gnirke, A. Targeted next-generation sequencing of a cancer transcriptome enhances detection of sequence variants and novel fusion transcripts. *Genome Biology 10*, 10 (2009), R115.

[68] Li, H., and Durbin, R. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics 25*, 14 (Jul 2009), 1754–1760.

[69] Li, H., and Durbin, R. Fast and accurate long-read alignment with Burrows-Wheeler transform. *Bioinformatics 26*, 5 (Mar 2010), 589–595.

[70] Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G. T., Abecasis, G. R., and Durbin, R. The Sequence Alignment/Map format and SAMtools. *Bioinformatics 25*, 16 (2009), 2078–2079.

[71] Li, H., Ruan, J., and Durbin, R. Mapping short DNA sequencing reads and calling variants using mapping quality scores. *Genome Research 18*, 11 (2008), 1851–1858.

[72] Li, R., Li, Y., Kristiansen, K., and Wang, J. SOAP: short oligonucleotide alignment program. *Bioinformatics 24*, 5 (Mar 2008), 713–714.

[73] Li, R., Yu, C., Li, Y., Lam, T. W., Yiu, S. M., Kristiansen, K., and Wang, J. SOAP2: an improved ultrafast tool for short read alignment. *Bioinformatics 25*, 15 (Aug 2009), 1966–1967.

[74] Li, Y., Chien, J., Smith, D. I., and Ma, J. FusionHunter: identifying fusion transcripts in cancer using paired-end RNA-seq. *Bioinformatics 27*, 12 (Jun 2011), 1708–1710.

[75] Lin, H., Zhang., Z., Zhang., M., Ma., B., and Li, M. ZOOM! Zillions of oligos mapped. *Bioinformatics 24*, 21 (2008), 2431–2437.

[76] Lin, Y.-Y., Dao, P., Hach, F., Bakhshi, M., Mo, F., Lapuk, A., Collins, C., and Sahinalp, S. C. CLIIQ: Accurate Comparative Detection and Quantification of Expressed Isoforms in a Population. In *WABI* (2012), pp. 178–189.

[77] Loh, P. R., Baym, M., and Berger, B. Compressive genomics. *Nat. Biotechnol. 30*, 7 (Jul 2012), 627–630.

[78] Lupski, J. R., Reid, J. G., Gonzaga-Jauregui, C., Rio Deiros, D., Chen, D. C., Nazareth, L., Bainbridge, M., Dinh, H., Jing, C., Wheeler, D. A., McGuire, A. L., Zhang, F., Stankiewicz, P., Halperin, J. J., Yang, C., Gehman, C., Guo, D., Irikat, R. K., Tom, W., Fantin, N. J., Muzny, D. M., and Gibbs, R. A. Whole-genome sequencing in a patient with Charcot-Marie-Tooth neuropathy. *New England Journal of Medicine 362*, 13 (Apr 2010), 1181–1191.

[79] Manber, U., and Myers, G. Suffix Arrays: A New Method for On-Line String Searches. In *SODA* (1990), pp. 319–327.

[80] Marco-Sola, S., Sammeth, M., Guigo, R., and Ribeca, P. The GEM mapper: fast, accurate and versatile alignment by filtration. *Nature Methods 9*, 12 (Dec 2012), 1185–1188.

[81] Mardis, E. The impact of next-generation technology on genetics. *Trends Genetics 24*, 3 (2008), 133–141.

[82] Margulies, M., Egholm, M., Altman, W. E., Attiya, S., Bader, J. S., Bemben, L. A., Berka, J., Braverman, M. S., Chen, Y.-J., Chen, Z., Dewell, S. B., Du, L., Fierro, J. M., Gomes, X. V., Godwin, B. C., He, W., Helgesen, S., Ho, C. H., Ho, C. H., Irzyk, G. P., Jando, S. C., Alenquer, M. L. I., Jarvie, T. P., Jirage, K. B., Kim, J.-B., Knight, J. R., Lanza, J. R., Leamon, J. H., Lefkowitz, S. M., Lei, M., Li, J., Lohman, K. L., Lu, H., Makhijani, V. B., McDade, K. E., McKenna, M. P., Myers, E. W., Nickerson, E., Nobile, J. R., Plant, R., Puc, B. P., Ronan, M. T., Roth, G. T., Sarkis, G. J., Simons, J. F., Simpson, J. W., Srinivasan, M., Tartaro, K. R., Tomasz, A., Vogt, K. A., Volkmer, G. A., Wang, S. H., Wang, Y., Weiner, M. P., Yu, P., Begley, R. F., and Rothberg, J. M. Genome sequencing in microfabricated high-density picolitre reactors. *Nature 437*, 7057 (Sep 2005), 376–380.

[83] McKernan, K. J., Peckham, H. E., Costa, G. L., McLaughlin, S. F., Fu, Y., Tsung, E. F., Clouser, C. R., Duncan, C., Ichikawa, J. K., Lee, C. C.,

Zhang, Z., Ranade, S. S., Dimalanta, E. T., Hyland, F. C., Sokolsky, T. D., Zhang, L., Sheridan, A., Fu, H., Hendrickson, C. L., Li, B., Kotler, L., Stuart, J. R., Malek, J. A., Manning, J. M., Antipova, A. A., Perez, D. S., Moore, M. P., Hayashibara, K. C., Lyons, M. R., Beaudoin, R. E., Coleman, B. E., Laptewicz, M. W., Sannicandro, A. E., Rhodes, M. D., Gottimukkala, R. K., Yang, S., Bafna, V., Bashir, A., MacBride, A., Alkan, C., Kidd, J. M., Eichler, E. E., Reese, M. G., De La Vega, F. M., and Blanchard, A. P. Sequence and structural variation in a human genome uncovered by short-read, massively parallel ligation sequencing using two-base encoding. *Genome Research 19*, 9 (2009), 1527–41.

[84] McPherson, A., Hormozdiari, F., Zayed, A., Giuliany, R., Ha, G., Sun, M. G. F., Griffith, M., Heravi Moussavi, A., Senz, J., Melnyk, N., Pacheco, M., Marra, M. A., Hirst, M., Nielsen, T. O., Sahinalp, S. C., Huntsman, D., and Shah, S. P. deFuse: an algorithm for gene fusion discovery in tumor RNA-Seq data. *PLoS Computational Biology 7*, 5 (May 2011), e1001138.

[85] McPherson, A., Wu, C., Hajirasouliha, I., Hormozdiari, F., Hach, F., Lapuk, A., Volik, S., Shah, S., Collins, C., and Sahinalp, S. C. Comrad: detection of expressed rearrangements by integrated analysis of RNA-Seq and low coverage genome sequence data. *Bioinformatics 27*, 11 (Jun 2011), 1481–1488.

[86] McPherson, A., Wu, C., Wyatt, A., Shah, S., Collins, C., and Sahinalp, C. nFuse: discovery of complex genomic rearrangements in cancer using high-throughput sequencing. *Genome Research 22*, 11 (Nov 2012), 2250–2261.

[87] Medvedev, P., Stanciu, M., and Brudno, M. Computational methods for discovering structural variation with next-generation sequencing. *Nature Methods 6*, 11 Suppl (Nov 2009), 13–20.

[88] Miller, W., and Myers, E. W. Sequence comparison with concave weighting functions. *Bulletin of Mathematical Biology 50*, 2 (1988), 97–120.

[89] Mills, R. E., Pittard, W. S., Mullaney, J. M., Farooq, U., Creasy, T. H., Mahurkar, A. A., Kemeza, D. M., Strassler, D. S., Ponting, C. P., Webber, C., and Devine, S. E. Natural genetic variation caused by small insertions and deletions in the human genome. *Genome Research 21*, 6 (Jun 2011), 830–839.

[90] Minoche, A. E., Dohm, J. C., and Himmelbauer, H. Evaluation of genomic high-throughput sequencing data generated on Illumina HiSeq and Genome Analyzer systems. *Genome Biology 12*, 11 (Nov 2011), R112.

[91] Mott, R. EST_GENOME: a program to align spliced DNA sequences to unspliced genomic DNA. *Computer Applications in the Biosciences 13*, 4 (Aug 1997), 477–478.

[92] Myers, E. W. A Sublinear Algorithm for Approximate Keyword Searching. *Algorithmica 12*, 4/5 (1994), 345–374.

[93] Nacu, S., Yuan, W., Kan, Z., Bhatt, D., Rivers, C. S., Stinson, J., Peters, B. A., Modrusan, Z., Jung, K., Seshagiri, S., and Wu, T. D. Deep RNA sequencing analysis of readthrough gene fusions in human prostate adenocarcinoma and reference samples. *BMC Med Genomics 4* (2011), 11.

[94] Ng, S. B., Bigham, A. W., Buckingham, K. J., Hannibal, M. C., McMillin, M. J., Gildersleeve, H. I., Beck, A. E., Tabor, H. K., Cooper, G. M., Mefford, H. C., Lee, C., Turner, E. H., Smith, J. D., Rieder, M. J., Yoshiura, K.-I., Matsumoto, N., Ohta, T., Niikawa, N., Nickerson, D. A., Bamshad, M. J., and Shendure, J. Exome sequencing identifies MLL2 mutations as a cause of Kabuki syndrome. *Nature Genetics 42*, 9 (Sep 2010), 790–793.

[95] Nothnagel, M., Herrmann, A., Wolf, A., Schreiber, S., Platzer, M., Siebert, R., Krawczak, M., and Hampe, J. Technology-specific error signatures in the 1000 Genomes Project data. *Human Genetics 130*, 4 (Feb 2011), 505–516.

[96] Ondov, B. D., Varadarajan, A., Passalacqua, K. D., and Bergman, N. H. Efficient mapping of Applied Biosystems SOLiD sequence data to a reference genome for functional genomic applications. *Bioinformatics 24*, 23 (Dec 2008), 2776–2777.

[97] O'Roak, B. J., Deriziotis, P., Lee, C., Vives, L., Schwartz, J. J., Girirajan, S., Karakoc, E., Mackenzie, A. P., Ng, S. B., Baker, C., Rieder, M. J., Nickerson, D. A., Bernier, R., Fisher, S. E., Shendure, J., and Eichler, E. E. Exome sequencing in sporadic autism spectrum disorders identifies severe de novo mutations. *Nature Genetics 43*, 6 (Jun 2011), 585–589.

[98] Pushkarev, D., Neff, N. F., and Quake, S. R. Single-molecule sequencing of an individual human genome. *Nature Biotechnology 27*, 9 (Sep 2009), 847–850.

[99] Rissanen, J., and Langdon, G. G. Arithmetic Coding. *IBM Journal of Research and Development 23*, 2 (march 1979), 149 –162.

[100] Rumble, S. M., Lacroute, P., Dalca, A. V., Fiume, M., Sidow, A., and Brudno, M. SHRiMP: Accurate Mapping of Short Color-space Reads. *PLoS Computational Biology 5*, 5 (2009), 11.

[101] Sahinalp, S. C., and Vishkin, U. Symmetry breaking for suffix tree construction. In *STOC 1994* (1994), pp. 300–309.

[102] Sahinalp, S. C., and Vishkin, U. Efficient Approximate and Dynamic Matching of Patterns Using a Labeling Paradigm. In *FOCS 1996* (1996), pp. 320–328.

[103] Said, A. *Introducing to Arithmetic Coding - Theory and Practice*, published as a chapter in lossless compression handbook by khalid sayood ed. HPL-2004-76. Imaging Systems Laboratory, HP Laboratories Palo Alto, Apr. 2004.

[104] Sboner, A., Habegger, L., Pflueger, D., Terry, S., Chen, D. Z., Rozowsky, J. S., Tewari, A. K., Kitabayashi, N., Moss, B. J., Chee, M. S., Demichelis, F., Rubin, M. A., and Gerstein, M. B. FusionSeq: a modular framework for finding gene fusions by analyzing paired-end RNA-sequencing data. *Genome Biology 11*, 10 (2010), R104.

[105] Schadt, E. E., Linderman, M. D., Sorenson, J., Lee, L., and Nolan, G. P. Computational solutions to large-scale data management and analysis. *Nature Reviews. Genetics 11* (Sep 2010), 647–657.

[106] Sherry, S. T., Ward, M. H., Kholodov, M., Baker, J., Phan, L., Smigielski, E. M., and Sirotkin, K. dbSNP: the NCBI database of genetic variation. *Nucleic Acids Research 29* (Jan 2001), 308–311.

[107] Slater, G. S. C., and Birney, E. Automated generation of heuristics for biological sequence comparison. *BMC Bioinformatics 6* (2005), 31.

[108] Smith, D. R., Quinlan, A. R., Peckham, H. E., Makowsky, K., Tao, W., Woolf, B., Shen, L., Donahue, W. F., Tusneem, N., Stromberg, M. P., Stewart, D. A., Zhang, L., Ranade, S. S., Warner, J. B., Lee, C. C., Coleman, B. E., Zhang, Z., McLaughlin, S. F., Malek, J. A., Sorenson, J. M., Blanchard, A. P., Chapman, J., Hillman, D., Chen, F., Rokhsar, D. S., McKernan, K. J., Jeffries, T. W., Marth, G. T., and Richardson, P. M. Rapid whole-genome mutational profiling using next-generation sequencing technologies. *Genome Research 18*, 10 (Oct 2008), 1638–1642.

[109] Smith, T. F., and Waterman, M. S. Identification of common molecular subsequences. *Journal of Molecular Biology 147*, 1 (Mar 1981), 195–197.

[110] Sudmant, P. H., Kitzman, J. O., Antonacci, F., Alkan, C., Malig, M., Tsalenko, A., Sampas, N., Bruhn, L., Shendure, J., Project, . G., and Eichler, E. E. Diversity of human copy number variation and multicopy genes. *Science 330*, 6004 (Oct 2010), 641–646.

[111] Tembe, W., Lowey, J., and Suh, E. G-SQZ: compact encoding of genomic sequence and quality data. *Bioinformatics 26*, 17 (Sep 2010), 2192–2194.

[112] Trapnell, C., Pachter, L., and Salzberg, S. L. TopHat: discovering splice junctions with RNA-Seq. *Bioinformatics 25*, 9 (May 2009), 1105–1111.

[113] Vissers, L. E. L. M., de Ligt, J., Gilissen, C., Janssen, I., Steehouwer, M., de Vries, P., van Lier, B., Arts, P., Wieskamp, N., del Rosario, M.,

van Bon, B. W. M., Hoischen, A., de Vries, B. B. A., Brunner, H. G., and Veltman, J. A. A de novo paradigm for mental retardation. *Nature Genetics 42*, 12 (Dec 2010), 1109–1112.

[114] Wan, R., Anh, V. N., and Asai, K. Transformations for the compression of FASTQ quality scores of next-generation sequencing data. *Bioinformatics 28*, 5 (2012), 628–635.

[115] Weese, D., Emde, A. K., Rausch, T., Doring, A., and Reinert, K. RazerS–fast read mapping with sensitivity control. *Genome Research 19*, 9 (Sep 2009), 1646–1654.

[116] Weese, D., Holtgrewe, M., and Reinert, K. RazerS 3: faster, fully sensitive read mapping. *Bioinformatics 28*, 20 (Oct 2012), 2592–2599.

[117] Wheeler, D. A., Srinivasan, M., Egholm, M., Shen, Y., Chen, L., McGuire, A., He, W., Chen, Y.-J., Makhijani, V., Roth, G. T., Gomes, X., Tartaro, K., Niazi, F., Turcotte, C. L., Irzyk, G. P., Lupski, J. R., Chinault, C., zhi Song, X., Liu, Y., Yuan, Y., Nazareth, L., Qin, X., Muzny, D. M., Margulies, M., Weinstock, G. M., Gibbs, R. A., and Rothberg, J. M. The complete genome of an individual by massively parallel DNA sequencing. *Nature 452*, 7189 (Apr 2008), 872–876.

[118] Witten, I. H., Neal, R. M., and Cleary, J. G. Arithmetic coding for data compression. *Commun. ACM 30*, 6 (June 1987), 520–540.

[119] Wu, C., Wyatt, A. W., Lapuk, A. V., McPherson, A., McConeghy, B. J., Bell, R. H., Anderson, S., Haegert, A., Brahmbhatt, S., Shukin, R., Mo, F., Li, E., Fazli, L., Hurtado-Coll, A., Jones, E. C., Butterfield, Y. S., Hach, F., Hormozdiari, F., Hajirasouliha, I., Boutros, P. C., Bristow, R. G., Jones, S. J., Hirst, M., Marra, M. A., Maher, C. A., Chinnaiyan, A. M., Sahinalp, S. C., Gleave, M. E., Volik, S. V., and Collins, C. C. Integrated genome and transcriptome sequencing identifies a novel form of hybrid and aggressive prostate cancer. *J. Pathol. 227*, 1 (May 2012), 53–61.

[120] Wu, T. D., and Nacu, S. Fast and SNP-tolerant detection of complex variants and splicing in short reads. *Bioinformatics 26*, 7 (Apr 2010), 873–881.

[121] Wu, T. D., and Watanabe, C. K. GMAP: a genomic mapping and alignment program for mRNA and EST sequences. *Bioinformatics 21*, 9 (May 2005), 1859–1875.

[122] Yanovsky, V. ReCoil - an algorithm for compression of extremely large datasets of DNA data. *Algorithms Mol Biol 6* (2011), 23.

[123] Yorukoglu, D., Hach, F., Swanson, L., Collins, C. C., Birol, I., and Sahinalp, S. C. Dissect: detection and characterization of novel structural alterations in transcribed sequences. *Bioinformatics 28*, 12 (Jun 2012), i179–187.

[124] Ziv, J., and Lempel, A. A Universal Algorithm for Sequential Data Compression. *IEEE Transactions on Information Theory 23*, 3 (1977), 337–343.

[125] Ziv, J., and Lempel, A. Compression of Individual Sequences via Variable-Rate Coding. *IEEE Transactions on Information Theory 24*, 5 (1978), 530–536.