Detecting and Analyzing Genomic Structural Variation using Distributed Computing

Christopher Whelan
A.B. Computer Science, Harvard University, 1997

Presented to the
Center for Spoken Language Understanding
within the Oregon Health & Science University
School of Medicine
in partial fulfillment of the
requirements for the degree
Doctor of Philosophy
in
Computer Science & Engineering

February 2014

Center for Spoken Language Understanding
School of Medicine
Oregon Health & Science University

CERTIFICATE OF APPROVAL

This is to certify that the Ph.D. dissertation of
Christopher Whelan
has been approved.

Dr. Kemal Sönmez, Thesis Advisor
Associate Professor

Dr. Izhak Shafran
Associate Professor

Dr. Brian Roark
Research Scientist, Google, Inc.

Dr. Lucia Carbone
Assistant Professor, Dept. of Behavioral Neuroscience

Dr. Steven Bedrick
Assistant Professor

Acknowledgments

This dissertation would not be possible without my co-advisors, Kemal Sönmez and Lucia Carbone. Kemal has been an excellent mentor and a great friend. He helped me to develop many ideas that went into this dissertation. Of all of his valuable contributions, I am most thankful for his ability to point me in new directions and re-energize me when I was discouraged or had painted myself into a corner.

Lucia has also been a terrific role model and an inspiring scientific leader. The day I was given the chance to collaborate with her and her lab was my luckiest moment at OHSU. Lucia has taught me most of the biology I know, given me the opportunity to work on tremendously exciting projects, and gone out of her way many times to introduce me to members of the genomics community; in particular, I owe a great debt of gratitude to her for bringing me annually to the Biology of Genomes meeting at Cold Spring Harbor. Computational work cannot have real value without actual biological data and real scientific questions to work on and learn from, both of which Lucia provided. Of course, Lucia also introduced me to the problem of structural variation detection; this dissertation could not exist without her. I would also like to thank the other members of the Carbone Lab, who have been a pleasure to work with and learn from: Larry Wilhelm, Josh Meyer, Nathan Lazar, Liz Terhune, and Kim Nevonen.

I would also like to thank Brian Roark and Zak Shafran for all of the great help they've given me. Brian is a fabulous teacher and always provided excellent feedback and made himself available for questions or just to talk. Cloudbreak started as a project in a course taught by Zak and Richard Sproat on problem solving with large clusters, which provided me with the core idea that eventually turned into this dissertation. Zak has done a great deal of other things for me as well: providing me with ideas to use in my research; inviting me to explore new topics in machine learning with him and his group; and last but not least acquiring and helping to configure and administer the Hadoop compute clusters at CSLU, without which this work would not be possible. On that last note, I would also like to thank cluster administrators who have put up with my sometimes taxing usage of our compute resources, including Rob Stites and Jason Brooks.

I would also like to acknowledge members of the broader genomics community who have provided feedback and encouragement for my projects, including Bob Handsaker, Steve McCarroll, and Ben Raphael.

Finally, my greatest thanks go to my wife Sarah and my son Colin, who displayed tremendous patience in putting up with their husband and father's graduate-school lifestyle. Sarah's support, encouragement, and love have been an integral component of this work, without which it could never have happened.

Contents

List of Tables

List of Figures

Abstract

Genomic structural variations are an important class of genetic variants with a wide variety of functional impacts. The detection of structural variations using high-throughput short-read sequencing data is a difficult problem, and published algorithms do not provide the sensitivity and specificity required in research and clinical settings. Meanwhile, high-throughput sequencing is rapidly generating ever-larger data sets, necessitating the development of algorithms that can provide results rapidly and scale to use cloud and cluster infrastructures. MapReduce and Hadoop are becoming a standard for managing the distributed processing of large data sets, but existing structural variation detection approaches are difficult to translate into the MapReduce framework. We have formulated a general framework for structural variation detection in MapReduce, and implemented a software package called Cloudbreak, which detects genomic deletions and insertions with very high accuracy compared to existing popular tools. Through the use of MapReduce and Hadoop, Cloudbreak can scale to harness large compute clusters and big data sets, leading to much faster runtimes than existing methods. In addition, we show that Cloudbreak's formulation of the structural variation detection problem in terms of local feature generation allows it to simultaneously integrate many informative signals in statistical learning frameworks. We demonstrate this using conditional random fields, which enable learning conditional probability distributions over labels on sequences of observations, and show that it improves Cloudbreak's results, in particular increasing breakpoint resolution.

In addition to the development of Cloudbreak and its extensions, we describe a data analysis project in which we examined the genomic features that occur near evolutionary breakpoints in the genome of the gibbon, whose karyotype is heavily rearranged compared to other primate species. Using a distributed pipeline to conduct Monte Carlo permutation tests, we find a statistical enrichment of segmental duplications, certain families of transposable elements, and evolutionarily shared binding sites of the protein CTCF near the locations of gibbon rearrangements. These findings may help us understand the process by which structural variations formed and were preserved in the gibbon lineage.

Chapter 1

Introduction

The aim of the field of *genomics* is to characterize the structure and function of the DNA of an organism or population of organisms, with the ultimate goal of understanding how the sequences of nucleotides that make up the genome affect phenotypes and reveal evolutionary history. To do so, it is necessary to identify and understand the differences between the DNA from two samples, whether the samples come from two different individuals or two different tissues from the same individual. Given that each sample can contain DNA from multiple cells, and each cell in a human sample contains approximately 3 billion DNA bases packaged into two sets of 23 chromosomes, this is difficult and complex undertaking.

If, as is the case for humans, the species has been widely studied, a *reference genome* is often used in place of one of the samples. This allows variations between individuals to be described as variations between the sample and the reference. Experiments of this type are known as *resequencing* experiments. Assuming that a reference is used, or that the samples come from an individual or individuals from the same or closely related species, the majority of the DNA sequence will be the same between the two samples. In that case, variants can be categorized into one of several forms. The first is a difference of a single base of DNA at a particular location within the genome, or a *single nucleotide variant* (SNV). If the variation is shared between many individuals in a population, these are referred to as *single nucleotide polymorphisms* (SNPs). Another type of variation is the insertion or deletion of a small number of base pairs at a particular location, commonly referred to as *indels*. A final category of variants are genomic *structural variations* (SVs). This term describes variations that affect a large number of bases of DNA (in common usage at least 40 base pairs, ranging up to hundreds of megabases or entire chromosomes). SVs can take a variety of forms: strings of DNA can be deleted, inserted, duplicated, inverted, or translocated to a different chromosome. Because they can be large and frequent, SVs account for the majority of the bases that differ among normal human genomes [128, 42].

All of these types of variants can alter the function of a genome in different ways. SNVs can change the sequence of the proteins coded for by genes, sometimes altering their function and sometimes rendering them inactive, or they can alter regulatory elements and cause genes to be expressed at differing levels. Indels typically disrupt the function of a gene or regulatory element. SV's can cause a variety of functional changes, ranging from the deletion of exons to the formation of fusion genes such as the famous Brc-Abl Philadelphia chromosome in chronic myelogenous leukemia [87].

In fact, SVs are very common in some types of cancer, producing extremely rearranged genomes. Because of this, they have particular importance in cancer research, and it is therefore essential to be able to identify them in biological samples, as well as to characterize their functional impact and the mechanisms of their formation. The SVs that arise in cancer genomes have similarities to those that have arisen between different species in evolution (Figure 1.1). For example, the genomes of gibbon species, while still closely related to humans and other apes, have undergone a variety of chromosomal rearrangements and other structural variations when compared to these other species; many more, in fact, than are typical in species with similar levels of divergence. Understanding the evolutionary changes in gibbons, therefore, may shed light on the much faster processes that take place in progenitor cancer cells.

The current technology to detect genomic variations is massively parallel, high-throughput sequencing. This procedure involves amplifying the DNA from a sample and then shearing it into small fragments. The ends of those fragments are then sequenced, producing hundreds of millions or billions of *read pairs* for a sample using current, widely used instruments. The challenge of genomics is then to discover the variants present in the sample using only these short reads. By generating and sequencing enough fragments from a sample (quantified in terms of *coverage*, the average number of reads that cover any one locus in the reference genome), the hope is that it should be possible to capture almost all of the important variants in a given sample. Current practices suggest that 30X average coverage

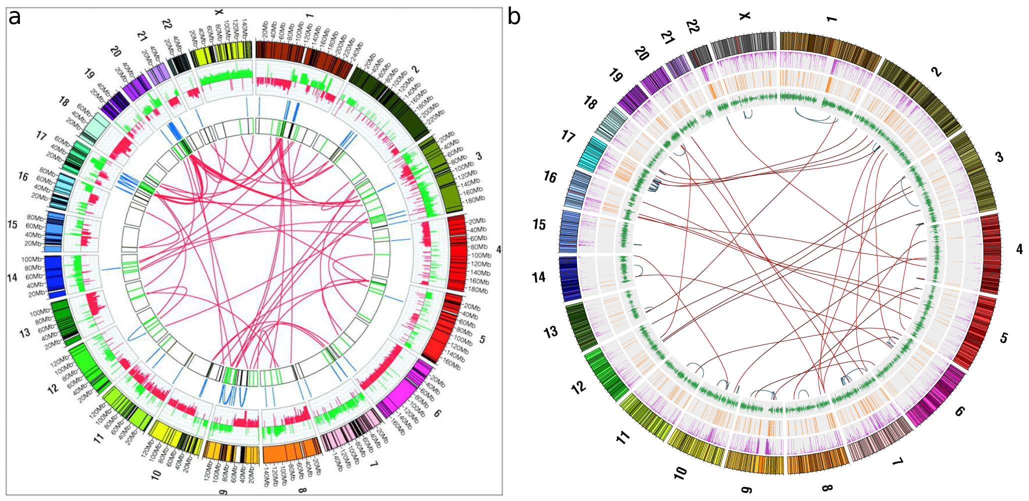

Figure 1.1: Similarity of genomic breakpoints that occur in cancer and evolution. The regions around the circle represent the human chromosomes. Lines in the centers of the circles show rearrangements between and within chromosomes. A) Somatic rearrangements detected within a breast cancer cell line, adapted from [64]. B) Rearrangements with respect to the human genome sequence present in the gibbon genome, adapted from [29].

depth is required to achieve accuracy in detecting SNVs. Even with this level of coverage, however, the task is made difficult by errors in the sequencing process, the fact that mammalian genomes are filled with repetitive sequences that make it difficult to ascertain the location in the genome that generated a particular read pair, and the computational challenges of analyzing the volume of data generated.

While SNVs and indels can be characterized relatively well from sequencing data using current algorithmic approaches, the identification of SVs remains challenging. SVs often arise within repetitive regions of the genome, making it difficult to align the reads surrounding them unambiguously to the reference genome. In addition, when the boundaries of an SV (the *breakpoints*) fall within a read sequence, it can be difficult to map that read back to its point of origin in the reference sequence. Although some approaches are based on searching for reads that contain breakpoints, it is often necessary to fall back on other signals present in the read set: the distance between successfully mapped read pairs, which should match the size of the fragments generated in the sequencing protocol, and the depth of reads that cover individual loci along the genome. Detection of SVs with current high-throughput sequencing technology remains a difficult problem, with limited concordance between available algorithms and high false discovery rates [128].

While current approaches struggle with accuracy, they also often fail to consider speed and scalability. A 30X coverage data set for an individual sample, in compressed format with associated quality scores, contains over 100GB of data to analyze. A typical bioinformatic pipeline includes steps to run quality control checks on the raw data; align the reads to the reference genome; perform filtering and recalibration steps after alignment; call SNVs and indels; and finally search for SVs. While a great deal of effort has been put into developing, optimizing, and parallelizing fast methods for alignment and variant calling, SV detection algorithms have not received the same attention, primarily because research in SV detection algorithms has focused on improving accuracy. As large scale sequencing projects like the 1000 Genomes Project and The Cancer Genome Atlas grow, the need for fast and accurate algorithms is becoming more apparent. DNA sequencing is already moving into the clinic, which will only exacerbate this requirement by requiring rapid analysis turnarounds for patients.

One approach to scaling data analysis pipelines is to harness the power of distributed computing using frameworks that tie together clusters of servers. Google's MapReduce [45] framework was designed to manage the storage and efficient processing of very large scale data sets across clusters of commodity servers. Hadoop is an open source project of the Apache Foundation which provides an implementation of the MapReduce programming framework as well as a distributed file system (HDFS) for distributing the redundant storage of large data sets across a cluster. Hadoop and MapReduce are rapidly becoming a standard in industrial data mining applications. However, they require the use of a specific programming model, which can make it difficult to design general-purpose algorithms for arbitrary sequencing analysis problems like SV detection.

This dissertation presents several novel techniques for detecting SVs using distributed computing and machine learning, which derive from the development of an algorithmic framework for SV detection methods in MapReduce. In particular, the main contributions are:

- The description of an algorithmic framework for solving SV detection problems in

Hadoop and MapReduce based on the computation of local features along the genome from paired end mappings (Chapter 4).

- The development in this framework of a software package, Cloudbreak, for discovering genomic deletions up to 25,000bp long, and short insertions, which improves accuracy over existing approaches and uses distributed computing to achieve dramatically faster runtimes (Chapter 5 and Chapter 6).
- An evaluation of the strengths and weaknesses of Cloudbreak, as compared to several other popular SV detection tools, when tested on several real and simulated data sets (Chapter 6).
- An exploration of the use of local features as described in Chapter 4 to reformulate SV detection as a sequence labeling problem, and the corresponding implementation and evaluation of a conditional random field model to create a novel method for integrating different signals of structural variations (Chapter 7).

In separate work not related to the algorithmic developments listed above, we also present the results of data analysis projects which examined the SVs that have massively rearranged the genome of the gibbon in an evolutionary time frame. This has led to the additional contribution of:

- Identification of sets of genomic features that are enriched near the breakpoints of the structural variations are present between gibbons and humans. These include segmental duplications and some families of transposable elements, as well as evolutionarily shared transcription factor binding sites. This analysis enhances our understanding of gibbon genome rearrangements. (Chapter 8).

A preliminary version of parts of this work was peer-reviewed and accepted for oral presentation at the Third Annual RECOMB Satellite Workshop On Massively Parallel Sequencing (RECOMB-seq). A preliminary breakpoint analysis was published in [28]. Throughout this document, I have tried to explicitly identify any work that was carried out by my collaborators.

Chapter 2

Biological Background

In this chapter we will explore the nature of structural variations, discuss the mechanisms that may be behind their formation, and their effect on phenotypes and disease. We will then describe high-throughput sequencing and discuss the typical informatics pipelines that researchers create to analyze sequencing data, placing SV detection within the context of the other types of computation that must be done to support end-to-end analysis. Finally, we will discus the challenges that the increasing number and size of sequencing data sets represent.

2.1 Structural Variations

As we described in the introduction, structural variations (SVs) are rearrangements, relative to some reference genome sequence, of sequences of DNA with a length longer than 40 or 50bp. These can take the form of deletions of reference sequence, insertions of novel sequence, duplications of sequence, inversions of orientation of a stretch of sequence with respect to the rest of its chromosome, or translocations of sequence from one chromosome to another. Deletions, insertions, and duplications are also frequently referred to as *copy number variations* (CNVs). In addition, many complex SVs exist in which these simple SV types are combined.

SVs can occur and become part of a gene pool on several different biological and time scales. On the smallest level, SVs can rearrange the genomes of individual cells within an organism. If the SV contributes to the cell's ability to proliferate, or occurs in conjunction with another mutation that does so, it can give rise to a tumor, or a new subpopulation or clone within an existing tumor. On a population scale, the genomes of individuals within a species harbor SVs with respect to one another, either because they were transmitted through the germline or because they arose *de novo* in certain individuals. Finally, on a larger scale, SVs within a species may become fixed, contributing to differences between the genomes of varying species on an evolutionary scale.

2.1.1 SV Effects on Phenotype and Disease

On all of these levels, SVs are associated with differences in phenotypes and have been shown to associate with many types of disease, making them an important area of study. Of course, our knowledge of these effects has been shaped by the types of SVs that researchers have been able to detect in the past several decades given the assays at their disposal and the scale at which samples could be analyzed. These limitations have biased our understanding towards the large variants with large effect sizes that were discoverable before the advent of modern array and sequencing technologies.

Cancer genomes contain many SVs that have been linked to tumor progression. The famous Brc-Abl mutation [87] mentioned previously is the most famous example of a fusion gene, in which portions of the genome coding for two different genes are made adjacent due to an SV. This results in the creation of new gene which can take on novel functions, which in turn drive cancer growth. Fusion genes can be caused through any type of SV that causes a novel adjacency between regions of the genome, including deletions, duplications, translocations, and inversions, and there have been dozens of recurring fusion genes identified across varying cancers to this point [10]. In another example, the genomic locus that contains the *MYC* gene, which among other functions regulates cell proliferation [50], is frequently duplicated multiple times in breast cancer cells [55]. These duplications change the amount of protein made by the cell, causing a dosage-related defect in cell regulation. In other cancer cells, a dramatic process called *chromothripsis* has been observed, in which entire chromosomes appear to "shatter" and then be pieced back together. This gives rise to hundreds of SVs, some of which can promote cancer development [179]. On the other end of the spectrum in terms of variant size, small deletions of less than 150bp in untranslated coding regions of genes have been observed to contribute to aberrant gene expression in leukemia [73].

On the scale of the human population, SVs and CNVs have been implicated in a variety of diseases (for a comprehensive recent review see Weischenfeldt et al. [191]). The best studied of these are large CNVs, including deletions or duplications of genes. These can can include variants that arise *de novo* in individuals, either giving rise to rare disorders [115] or contributing to the risk of more common conditions such as Crohn's disease [122], autism [169], and schizophrenia [188]. However, most CNVs in the human genome are polymorphic within the population and contribute to human genetic diversity [123]. The rise of microarrays and high-throughput sequencing has sped the discovery of smaller variants that affect coding sequence by deleting exons. To give one recent example, an 8kb deletion of an exon in the BAG3 gene has been tied to risk for cardiomyopathy [141].

There are also numerous examples of deletions that affect human phenotypes even though they do not directly overlap with the coding regions of genes, instead interacting with regulatory elements that control gene expression levels. For example, deletions ranging from 36kb to 319kb in size that are over one megabase away from the SOX9 gene locus can alter its expression by removing an enhancer element, giving rise to a rare cleft palate syndrome called Pierre Robin sequence [21]. Expanding on this idea, a survey of CNVs between inbred mouse strains showed that 28% of gene expression differences between strains could be statistically explained by CNVs, but that only 7% of the identified CNVs directly overlapped one of the differentially expressed genes, suggesting that SVs may shape many aspects of organism phenotypes in complex ways [26]. In another dramatic example from other species, the well-studied stickleback fish, which has repeatedly evolved adaptations to fresh water in several locations around the world as it moved from the sea into lakes, owes one of those adaptations, the loss of spines on its pelvis, to recurrent sub-2kb deletions of an enhancer that regulates the Pitx1 gene [31].

2.1.2 Mechanisms and Signatures of SV Formation

Given that many of the phenotypic effects of SVs described above are deleterious to the organism, it is natural to ask how they arise in genomes. Biologists have identified several different mechanisms that can cause SVs. By studying many of the genomic disorders mentioned in the previous section, geneticists found that often CNVs recur in different individuals at the same locations in the genome. Examining these locations revealed that they are homologous to sequences repeated elsewhere in the genome in the form of segmental duplications, stretches of DNA that are over 1kb in length with greater than 90% sequence identity [170]. This helped to identify the process of non-allelic homologous recombination (NAHR) [112]. In the best studied form of NAHR, the cell attempts to repair DNA in the cell that has suffered double-stranded breaks (DSBs). To do so, it attempts to find homologous DNA in the nucleus that can be used as a template to repair the broken strand. However, repeats in the genome provide the opportunity for DNA from the wrong genomic location to be used in this repair process, creating genomic breakpoints in the repaired DNA. In mammals, as little as 295bp of non-allelic homologous sequence can be responsible for NAHR events [111]. The human genome contains many copies of transposable elements, including 500,000 of the LINE family and over one million of the *Alu* family, with average lengths of 6kb and 300bp, respectively. Both *Alu* [104] and LINE [158] are frequently involved in NAHR events. As these and other transposons copy themselves into new locations in the genome, they also give rise to their own class of SV, mobile element insertions (MEIs), which leave their own signatures of small tandem site duplications (TSDs) at either end of the insertions.

In contrast to NAHR, other mechanisms can create SVs without extensive sequence similarity at the breakpoint. Non-homologous end joining (NHEJ) and microhomology mediated end joining (MMEJ) are two such processes [67]. Both of these mechanisms involve the repair of DSBs in the cell and typically result in SV breakpoints that have very small (less than 20bp) homologies on either side. Other processes of SV formation that leave a signature of microhomologies at the breakpoint are Fork Stalling and Template Switching (FoSTeS) [101] and microhomology-mediated break-induced replication (MMBIR) [67]. In these models, as the cell replicates all of its DNA two replication forks can switch DNA templates, causing complex structural variations to form. Finally, the chromothripsis process mentioned above also gives rise to microhomologies [113].

Given that different SV formation mechanisms have different signatures that can be determined by the presence of varying amounts of duplication or homology at or near the breakpoints, it is possible to analyze SVs to gain insight into their formation in different contexts. For example, Conrad et al. [41] surveyed CNVs from 40 unrelated individuals using a targeted sequencing method that enabled them to determine exact sequences around the breakpoints of some events. They found that although the best studied human genomic disorders are due to recurrent SVs caused by NAHR, NHEJ and MMEJ were responsible for the great majority of CNVs they were able to detect, a result later confirmed by the 1000 Genomes Project [128], although studies in mice have found that MEI may be more frequent albeit harder to detect [197]. In Chapter 8, we will discuss efforts to analyze the sequence context of genomic rearrangements between species.

2.2 High-Throughput Short-Read Sequencing

The pace of analysis of genome rearrangements and SVs has risen dramatically with the widespread adoption of high-throughput short-read sequencing (for a review see Shendure and Ji [172]). In early projects to interrogate SVs in DNA samples researchers had to painstakingly map individual variants using time-consuming microscopy (for very large rearrangements), fluorescence in situ hybridization (FISH), southern blotting, or polymerase chain reaction (PCR) assays [14]. The development of microarray technology allowed simultaneous testing of many genomic probes on a sample, although the probes had to be developed using prior knowledge of what variants to expect. Short-read sequencing, however, allows direct DNA sequence data to be collected in a relatively unbiased way from the entire genome, overcoming these limitations.

High-throughput short-read sequencing technology is currently dominated by Illumina, Inc. (San Diego, CA), and unless otherwise noted we will be referring to data generated by Illumina instruments when we refer to short-read or next-generation sequencing data in this document. The Illumina sequencing process begins with creation of genomic DNA library, which has billions of small fragments of DNA, sheared randomly across the genome, with adapter sequences attached to the ends. These adapters attach randomly to locations on the surface of the sequencing instrument's *flow cell*, and then are amplified in place to form clusters on the flow cell, each consisting of thousands of copies of a DNA fragment from the library. The sequencer then initiates a process in which the complements of each strand of DNA in the clusters are synthesized one base at a time. Each complementary base added to the new molecule contains a fluorescent tag, and the instrument takes a picture of the flow cell at each step. Pooling the color signal from each strand in a cluster, the sequencer optically analyzes the image from each time step to call the base at that position in each cluster. Because all clusters can be analyzed simultaneously in a single flow cell image, Illumina's sequencing instruments can now process billions of fragments at the same time. Illumina's HiSeq 2500 instrument, the current throughput leader, can produce 1.2 billion paired-end reads, or 120Gb of data, in 27 hours.

While this technique produces unprecedented amounts of sequencing data, the downside is that read lengths are limited by the extent to which the synthesis process can be regulated and kept in sync across all of the clusters on the flow cell. In Illumina's current high-throughput instruments, this limits read lengths to 100bp or 150bp. The downsides of this can be mitigated somewhat through techniques that sequence both ends of a DNA fragments, producing the paired-end reads referred to earlier. If the size of the fragment is approximately known, the distance between the two sequenced ends can be estimated, as we will discuss in detail in Section 3.2. These paired-end techniques typically use fragments of between 250 and 500bp, although in some cases *mate pair* fragment sequencing can produce reads from the ends of much larger fragments (6-8kb).

Because nearly half of the human genome is made up of repetitive elements, the analysis of the short reads produced by these sequencing technologies present a unique set of challenges [182]. As we mentioned earlier, repetitive elements such as *Alu* and LINE appear many times in the genome. If a read, or pair of reads, falls entirely within a single repetitive element, it will be impossible to determine the location in the genome it came from if there has not been enough sequence divergence of the repeat over evolutionary time to distinguish it from the others. Because many SV formation mechanisms depend on sequence homologies at the breakpoints, they are particularly likely to fall into repeats in the genome, complicating the task of SV detection from short-read data.

In addition to making SV detection difficult, short-read sequencing – by the unprecedented volume of data it produces – also presents challenges to developing scalable infrastructure and algorithms for processing sequencing data in general. We will return to the problem of SV detection in the next chapter, and in the remainder of this chapter will give an overview of the applications of high-throughput sequencing data, and will discuss the computational challenges they represent.

2.2.1 Sequencing Analysis Pipelines

To transform raw sequencing reads into usable information typically requires a series of discrete steps. As genomic bioinformatics has matured, these have been increasingly modularized and made into components that can be connected together to create analytics pipelines designed for particular applications. In this section, we will describe pipelines for several of the most popular applications of high-throughput sequencing.

DNA Resequencing

As we mentioned in Chapter 1, resequencing projects attempt to identify variants present in the genomic DNA of samples from a population for which there exists a reference sequence. Variants can then be specified in standard formats (typically VCF, the Variant Call Format [44]) that describes them in terms of the sequence and coordinates of the reference. These can then be used in study designs such as case/control or genome wide association studies (GWAS), or, in clinical contexts, can be searched directly using prior knowledge for variants likely to be associated with diseases or conditions.

Figure 2.1 shows a high-level representation of the steps involved in a large-scale resequencing project using the Genome Analysis Toolkit [46]. The main components are:

1. **Quality Control and Data Preparation.** Although not shown in Figure 2.1, most pipelines typically begin by running a set of reports to assess the quality of the raw data, often with a standardized QC suite such as FastQC [8] and data preparation steps that include trimming the portions of the input sequences that have base calls with low confidence scores or that contain sequence that matches known artifacts of

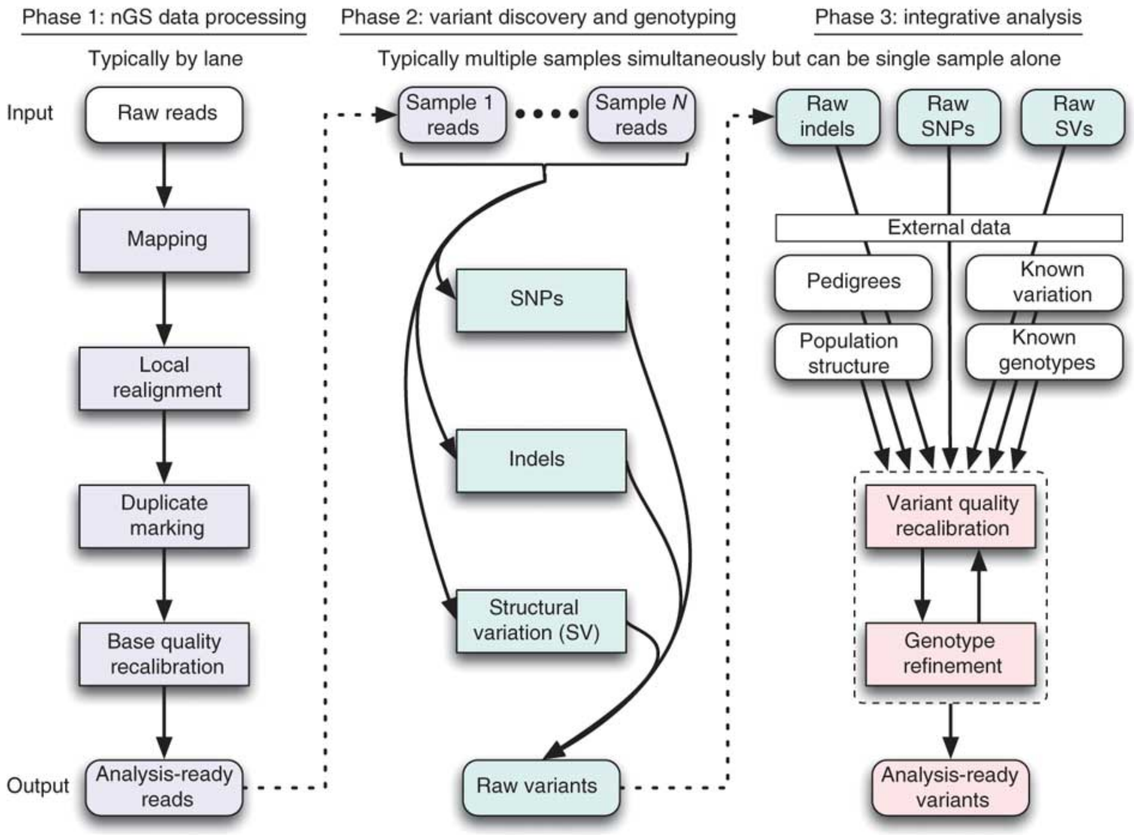

Figure 2.1: An example of a full computational pipeline for high-throughput short-read DNA resequencing projects. This figure describes the Genome Analysis Toolkit (GATK) best practices pipeline, and is representative of the computational steps involved in a full resequencing pipeline designed to process many samples simultaneously. Reprinted by permission from Macmillan Publishers Ltd: Nature Genetics 43:5pp491-498 [46], copyright 2011

the sequencing process (adapter trimming). These components operate on raw data in the FASTQ format [40].

2. **Read Mapping.** In this phase, reads are mapped to the reference genome to find their most likely coordinates of origin. There exist a wide variety of short-read mappers that are heavily optimized for the task of aligning short DNA sequences to reference genomes (for a survey from several years ago see Li and Homer [108]). These typically use some form of a seed-and-extend strategy, in which exact matches of portions of the input read to the reference genome are found quickly using pre-computed indices. These indices can take the form of hash tables or some form of

enhanced suffix array, or more popularly, the suffix trie equivalent FM-index [58], which uses the Burrows-Wheeler Transform (BWT) [25] to convert a suffix trie into a highly compressed structure that is particularly useful for indexing highly repetitive genome sequences while still giving fast (under most conditions linear in the length of the query pattern) lookup times. These exact matches are then extended using direct comparisons or through dynamic programming techniques such as the Smith-Waterman alignment algorithm [176]. Examples of hash-based aligners include Novoalign [142] and MOSAIK [103], while BWA [106] and Bowtie 2 [94] are popular BWT-based aligners. The output of this step is typically in the Sequence Alignment/Map (SAM) format [107], which in addition to the reads stores their alignment coordinates on the reference genome, mapping quality scores, and information about mismatches to the reference.

3. **Local Realignment, Duplicate Marking, and Recalibration.** In some pipelines there is often an additional quality control step after read mapping. PCR amplification in the preparation of sequencing libraries from DNA samples can give rise to duplicate fragments, which are typically identified and removed at this point. In addition, this step can include a more expensive realignment phase for groups of reads that appear to span short insertions and deletions (indels, typically under 40bp in size), and a recalibration of the base call quality scores reported by the sequencer based on the empirically observed distribution of mismatches between the reads and the reference sequence [46].

4. **Variant Calling.** The next phase includes calling of single nucleotide variants (SNVs), indels, and SVs. To call SNVs and indels, algorithms examine each possible variant indicated by mismatches of reads to the reference. They typically evaluate the number of reads supporting the variant against the total number of reads at that location, as well as prior beliefs, in a Bayesian framework. Popular SNP and INDEL callers include the GATK [124], SAMtools [107], and SOAPsnp [109]. In some cases, multiple samples from a population are examined simultaneously, giving additional power to detect variants in that population. We will discuss SV callers in more detail

in Chapter 3.

5. **Integrative Analysis/Filtering.** After raw variants have been called, additional sources of outside information can be used to adjust their confidence scores or filter out false positives. This can include data from existing catalogs of genomic variants or knowledge about the population structure of the samples. If multiple samples from the same pedigree are being sequenced, data can be reconciled based on the principles of inheritance. DePristo et al. [46] describe one such set of filters appropriate for a large scale study involving many individuals from a population. In Section 3.7 we will describe a simple filtering scheme we implemented for a cancer sequencing project.

Other Sequencing Applications

In addition to DNA resequencing to detect genomic variants, there are many other applications of short-read sequencing. We will not go into detail on how each works, but will briefly list the goals and major informatic components of several of the most common so that we can address a full picture of the computational scaling challenges that short-read sequencing represents.

- **RNA-seq.** In RNA-seq (see Oshlack et al. [146] for a review), the goal is to interrogate the expression of genes across samples, for example in different disease conditions. Rather than sequencing DNA, libraries are constructed from RNA taken from the sample. Similarly to DNA resequencing, the reads are QC'd and aligned to the reference genome, although there are read mappers especially designed for RNA-seq that include functionality to map reads over exon junctions. The reads that map to annotated genes on the reference are then analyzed and quantified to determine the genes that are being expressed in the sample, along with their relative expression levels and isoforms.

- **ChIP-seq** The intent of ChIP-seq (reviewed by Park [147]) is to determine the locations in which DNA, or the chromatin that contains it, is being bound by protein transcription factors or modified, with the ultimate goal of determining the modifications that are driving gene expression. In this case fragments of DNA that are bound to the protein of interest are preferentially separated and sequenced. After read trimming and mapping, the locations of binding events can be identified by searching for "peaks" of coverage on the reference genome.

- ***de novo* Assembly** When there is no reference genome present for the organism being sequenced, the goal is to try to reconstruct the full sequence of the input genome based only on the short reads. A full discussion of *de novo* assembly algorithms is beyond the scope of this section; for a review see Nagarajan and Pop [135]. The general approach is to build a graph representing overlaps between the reads in the form of either a string graph [133] or a *de Bruijn* graph, and then walk the graph to determine the underlying sequence. This is complicated by the fact that mammalian genomes are highly repetitive. The need to construct a graph data structure linking all of the input reads means that assembly is typically an extremely memory-intensive process, making it difficult to distribute computation.

There are of course additional sequencing applications, including workflows specifically designed for cancer-related projects and metagenomics, which aims to discover all of the microorganisms present in a diverse sample.

2.2.2 Big Data from Sequencing

All of these sequencing applications are creating an enormous amount of data that is pushing the limits of the computational infrastructure and algorithms used by biologists. For example, the European Bioinformatics Institute currently stores multiple petabytes of genomic data, and that amount has been taking less than one year to double since the advent of high-throughput sequencing in 2008 [120]. This has led to increasing calls for new bioinformatics strategies for storing, managing, sharing, and processing that data. An early recognition of the fact that computational requirements would soon outstrip the capacity of the prevailing model of processing data on local servers in individual laboratories created a movement towards centralized databases and processing on remote servers enabled by standardized data formats and annotations [177]. After the creation of large scale cloud compute services, bioinformatics leaders quickly realized their potential to ease the burden of processing sequencing data [178, 164, 162]. In particular, they noted that cloud computing would allow the scaling of compute infrastructure to meet demand, rather than having to plan for, acquire, and maintain compute clusters capable of handling the predicted peak load. In addition, cloud computing offers a potential solution to the problem of sharing large data sets in the highly collaborative field of genomics; by storing large data sets in the cloud that are accessible to multiple groups, collaborators could become less dependent on the prevalent strategy of sharing data by shipping hard drives to one another. Several of these arguments also noted that cloud computing is a natural fit for creating distributed algorithms that can be scaled to larger data sets by adding more compute resources, in particular through the use of the MapReduce programming model [164, 162]. We will return to the topic of cloud computing in Chapter 4, where we will develop a framework for detecting SVs in MapReduce. Before we do so, however, we will review existing algorithms for SV detection in Chapter 3.

Chapter 3

Algorithms for Structural Variation Detection

In Chapter 2, we described the biological impact of SVs, and the development of short-read sequencing technology, along with the computational challenges they represent. In this chapter we review existing algorithms for SV detection based on short-read sequencing data. We will discuss the four major computational strategies for SV detection from sequencing data, and review specific approaches, with an emphasis on algorithms that use paired sequencing reads as a primary source of information. Finally, as a demonstration of how SV detection can fit into the pipeline models we described in Chapter 2, we will describe a small pipeline for SV detection that we developed that used an existing tool, BreakDancer, to characterize SVs in a low-coverage cancer sample.

3.1 The Four Signals of SVs in Sequencing Data

Most popular SV detection algorithms can be placed into one of four categories based on the type of signal they extract and analyze from sequencing data sets [4, 83]. The first three categories depend upon first aligning short reads to the reference genome. In Section 2.2, we described paired-end sequencing, in which both ends of a single DNA fragments are sequenced. Read pair (RP) based methods use the distance between and orientation of the mappings of these sequenced ends to identify the signatures of SVs. Out of all SV detection techniques, RP approaches are the most commonly used based on their ability to detect many types of SVs and their computational tractability [4]. Read depth (RD) approaches identify regions of the genome with anomalous coverage by read mappings, which may indicate the presence of deletions or duplications, a subcategory of SV known as copy number variations (CNVs). Split Read (SR) approaches attempt to find local mappings of portions of individual reads that span SV breakpoints. Finally, assembly-based methods attempt to construct as much of the genome sequence as possible directly from the reads, without first mapping them to the reference genome. The constructed sequence is then compared to the reference to identify SVs. Beyond these four categories, several recent approaches have attempted to integrate more than one type of signal to increase accuracy. In this review of published algorithms, we will focus on those that are built to handle whole-genome DNA resequencing from a single sample at a time; there are of course other approaches for different applications, including SV detection from targeted resequencing data such as exome sequencing, and approaches which are designed only to operate on many samples at once.

3.2 Read Pair Approaches

Given correct mappings of paired reads to the reference genome and a reliable estimate of the length of the fragments from which the reads came, it is relatively straightforward to detect SVs. For example, Figure 3.1 shows the signatures of insertions and deletions based on the insert size of paired-end mappings: deletions lead to a longer distance between read mappings than expected, insertions to a shorter distance. Similarly, because protocols determine the orientation in which the read will be sequenced, inversions can be detected if they diverge from expectations. Finally, inversions can be detected if the reads map to different chromosomes in the reference genome. These ideas were first applied to high-throughput short-read sequencing data by Korbel et al. [85] and Campbell et al. [27], who built upon strategies developed for analyzing SVs through the mapping of sequenced ends of larger DNA fragments (bacterial artificial chromosomes (BACs) and fosmids) from non-high-throughput sequencing data [187, 156, 184].

Most read pair approaches use a basic strategy outlined in Figure 3.2. They begin by separating paired end mappings onto the reference genome into those that are *concordant* and those that are *discordant*, i.e. those which deviate from the expected insert size or

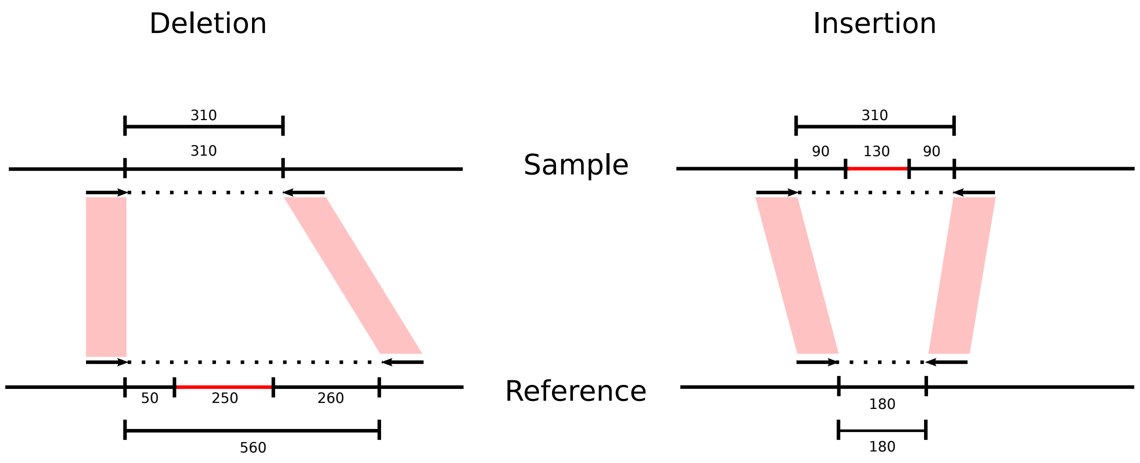

Figure 3.1: Detecting deletions and insertions from the distance between mappings of paired reads. For the case of a deletion, a fragment with a 310bp internal insert size is created from the sample DNA, in a region in which 250bp has been deleted relative to the reference. The two reads when mapped to the reference will be 460bp apart, providing evidence for a deletion if we expect the internal insert size to be approximately 300bp. A 310bp internal insert size fragment that overlaps a 150bp insertion in the sample, conversely, will give a distance between reads of 180bp when mapped to the reference.

orientation of the library. With only a few exceptions, discussed below, RP approaches discard concordant pairs and consider only discordant pairs for the remainder of their analysis. The next step is then to cluster the discordant mappings such that each cluster coherently supports a single candidate SV call. Finally, they filter or rank the candidate SV calls, for example only keeping those with support from multiple discordantly mapped read pairs.

The BreakDancerMax component of BreakDancer [37] is probably the most widely used of these algorithms[1]. As described above, BreakDancer first searches for discordant read pairs based on a hard threshold on the distance between mapped paired reads. It then looks for regions of the genome that anchor more discordant read pairs than expected according to its model; if two of these regions are connected by a minimum number of discordant read pairs, it calls an SV that links them. It calls breakpoints based on the inner boundaries of the two connected regions, as illustrated in Figure 3.2 (d), and then

[1]Based on a sampling of Google Scholar citation counts as of 12/28/2013: BreakDancer, 270; VariationHunter, 153; PEMer, 111; GASV, 65; SVDetect, 46.

assigns a confidence score by computing the likelihood of the SV based on a null model in which discordant read pairs are distributed across the genome according to a mixture Poisson model. Other seminal RP-based tools include GASV [173], PEMer [84], VariationHunter [72], HYDRA [154], and SVDetect [202]. These all operate on similar principles, differing primarily in the method used to cluster discordant read pairs that support the same potential SV call, and in the filtering steps used to select candidate SV calls. For example, GASV formulates the clustering step in terms of a geometric model in which the goal is to find the intersection of the coordinates of breakpoints that each discordant read pair could possibly support. Marschall et al. [118] insightfully observed that most of the popular clustering procedures are computationally equivalent to solving some variant of the problem of finding maximum cliques in a graph structure in which nodes are read pairs, and edges exist if the pairs could possibly support the same variant. VariationHunter and HYDRA were also the first approaches to also integrate ambiguously mapped discordant read pairs, a topic we will consider in the next section.

Read pair approaches have the advantage of being theoretically able to detect any type of SV except for multiple copy number duplications. Their disadvantages stem from the fact that they depend on comparing mapping distances between reads to the unknown size of the fragments from which they came. This means that they cannot capture the breakpoints of SVs with single nucleotide resolution, and that they depend on having a sequencing library with a tight distribution of fragment sizes in order to have power. In addition, the fact that many SV breakpoints occur in regions with high sequence similarity to other parts of the genome, as discussed in Section 2.1.2, means that it may be difficult to unambiguously map reads to those locations.

3.2.1 Ambiguously Mapped Read Pairs

Many of these approaches use only reads that are unambiguously mapped to the reference genome; this has the advantage of using the same set of alignments that are used for calling SNVs and indels in most sequencing pipelines. A second group of RP methods attempt to include discordant read pairs which cannot be unambiguously mapped to the reference genome in their analysis, in an effort to improve sensitivity in repetitive regions

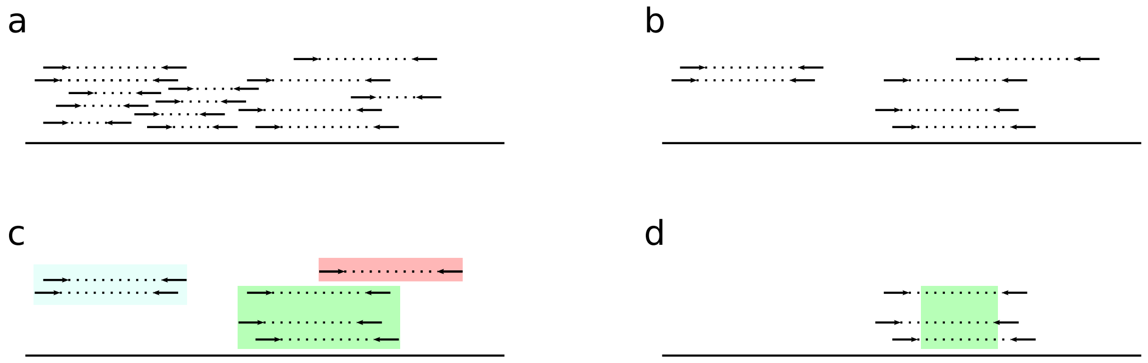

Figure 3.2: The general algorithmic steps of a classic read-pair algorithm. a) The algorithm accepts a set of alignments of paired reads to the reference as input. b) The algorithm identifies discordant read pairs. c) Discordant pairs are clustered to find groups that could support the same algorithm. d) Clusters of discordant read pairs are filtered (in this case, by the number of supporting read pairs), and bounds on the potential breakpoints are identified.

of the genome. One approach to incorporating this type of information can be found in *soft clustering* algorithms, which assign each ambiguously mapped read pair to one of its mappings such that it clusters with other discordant read pairs. These approaches include VariationHunter [72], which allocates ambiguously mapped reads by optimizing a maximum parsimony explanation of all discordant reads; HYDRA [154], which takes a similar approach based on heuristics; and GASVPro [174], which uses a Markov-Chain Monte Carlo sampling strategy to assign a read to its correct mapping. Even though they do consider more information than methods solely based on concordant reads, most methods in this category (with the exception of VariationHunter) use only a limited number of ambiguous discordant mappings per read pair, in part because of the storage and computational requirements necessary to process all or most ambiguous mappings of each read pair in a high-coverage data set. In Chapters 5 and 6, we will show that the MapReduce distributed computing framework has the potential to help address these challenges.

3.2.2 Concordant Read Pairs

The RP approaches listed above only consider the mapped insert sizes of discordant read pairs. (GASVPro does consider some concordant mappings in its final output, which we will mention in Section 3.6, but only in the sense of computing an RD signal rather than actually considering the insert sizes of the concordant pairs themselves.) Since discordant read pairs make up only a very small fraction of the mapped reads in a typical sequencing data set, these approaches are not considering much of the data available. Several algorithms have tried to use the information present in concordant read pairs to generate SV predictions. One simple example can be found in ChopSticks [199], which refines the breakpoint locations of deletion calls based on discordant pairs by looking for concordant pairs that overlap the ends of the predicted variant. This approach only works for homozygous deletions, however, and is therefore limited in scope.

More comprehensive attempts to include concordant RP information were described in MoDIL [102] and the BreakDancerMini component of BreakDancer [37]. MoDIL models the distribution of insert sizes at candidate locations in the genome using a Gaussian mixture model (GMM) describing the insert sizes observed; deletions and insertions are represented as additional components in the mixture. This has two advantages: because reads are not categorized as concordant or discordant based on a hard threshold, it is possible to detect smaller insertions and deletions; and these approaches can explicitly model the zygosity (presence of the variant on one or both of the pairs of chromosomes in the cell) of the variant in the sample, and potentially classify the variant as homozygous or heterozygous. The disadvantage of this approach in these implementations has been the computational requirements. BreakDancerMini considered *only* concordant read pairs, after processing discordant pairs in the BreakDancerMax. For each location in the genome they executed a sliding window Kolmogorov-Smirnov test comparing the distribution of concordant insert sizes to that of the entire library. This limited the approach to detecting only very small variants; the authors of BreakDancer no longer recommend running BreakDancerMini, presumably because of its computational requirements, instead suggesting SR-based approaches such as Pindel [200] to find smaller insertion and deletion variants. In Chapters 5 and 6, we will show that a strategy that considers the RP signal present in concordant read pairs can be made to give highly accurate result with very fast runtimes through the use of an algorithm developed in the MapReduce distributed computing framework.

Finally, CLEVER [118] took an alternative approach based on constructing a graph linking all paired reads, both concordant and discordant, that could support the same allele at a position, and then clustering reads based on finding maximum cliques in the graph. As mentioned above, they showed that this is conceptually similar to the clustering approaches used by other algorithms for discordant reads. They demonstrated that their approach was superior at detecting smaller variants because of its lack of hard thresholds for discordancy, and provided a relatively efficient implementation that can process a full 30X genome in approximately 8 hours.

3.3 Read Depth Approaches

Read-depth (RD) approaches consider the changing depth of coverage of concordantly mapped reads along the genome to infer the presence of SVs. For example, a homozygously deleted region will have zero coverage in the reference genome, while a region that has been duplicated many times, as can happen in some regions of the genome and in some cancers, will have a much higher coverage than average. These approaches differ mainly in the statistical and signal processing techniques used to identify anomalous regions. For example, CNVnator [2] uses a mean-shift approach to segment the genome into CNV regions. Other approaches in this category include MrFAST [5], Event-Wise Testing [201], and SegSeq [38].

RD approaches are good at finding large deletions and duplications. As previously noted, they are the only approach that can identify segments of the genome that have been duplicated multiple times. Their disadvantages are their lack of ability to reliably detect smaller events, and their breakpoint resolution, which is even lower than than of RP approaches.

3.4 Split Read Approaches

Split-read (SR) methods look for breakpoints within individual reads by mapping portions of the read to different genomic locations. Due to the computational challenge involved in aligning reads to the reference genome while allowing for very large gaps between portions of the read, they use different strategies to guide the search. Pindel [200] looks for paired reads in which one read in the pair aligned to the reference genome but the other did not. Supposing that the other read may contain a breakpoint, it searches the reference nearby for split read mappings. CREST [189] takes advantage of aligners that insert gaps at the ends of read alignments when there are many mismatches between the read and the reference, known as *soft clipping*. By looking for multiple alignments with soft clips at the same reference coordinate, it can identify breakpoints. SplazerS [51] did not use heuristics to guide the SR search, instead designing a unique mapping strategy based on mapping the prefixes and suffixes of reads independently. They showed that their unbiased search is more sensitive than other approaches, at the cost of greatly increased runtimes.

Split read approaches can identify SVs with high specificity and single base breakpoint accuracy. They are particularly good at detecting smaller variants. However, their sensitivity is limited by coverage and the length of the reads. As read lengths increase with advances in sequencing technology, they will play a larger role in SV detection.

3.5 Assembly-Based Approaches

An alternative approach to mapping reads to the species reference to discover variants is to first attempt to directly assemble the genomic sequence from which the reads were generated (AS approaches). This typically involves the construction of a *de Bruijn* graph to represent the overlapping k-mers in the entire read set, and then walking the graph to construct the longest possible unambiguous sequence of k-mers. Although most work in assembly is focused on *de novo* assembly, where there is no reference for the organism being sequenced, there have been attempts to find variants in a resequencing context by comparing the contigs that result from assembling reads to a reference genome. This strategy for detecting SVs was first demonstrated by Li et al. [110] using assemblies created at the Beijing Genomics Institute. One published assembly approach that is targeted at detecting variants including SVs is Cortex [75]. Cortex uses the reference to guide assembly with a colored de Bruijn graph structure, and can therefore identify SVs by walking colored paths in its graph. In general the task of creating *de novo* assemblies for human-sized genomes, however, is still an extremely involved and difficult process that requires a great deal of technical knowledge, as well as extremely high-memory compute nodes.

In practice, AS approaches are more likely to be used to verify candidate SV calls generated by other approaches. In these workflows, the reads mapped to the reference genome near the site of the possible variant are put into a *de novo* assembly process. The resulting sequencing generated by the assembly process is then aligned to the reference to see if it supports the existing call. TIGRA [36] is a recently published assembly tool explicitly for this process; other groups have reported building in-house custom pipelines to accomplish this [152, 116].

While AS approaches can theoretically identify any type of SV, in practice assembly requires extremely high coverage (typically 100X). In addition, the computational requirements necessitate high-memory servers, making the task difficult to run on widely available, non-specialized hardware. Finally, genome assembly using short reads tends to collapse identical repeats, leading to a loss of visibility in repetitive regions of the genome and in segmental duplications [6]. Because SVs are enriched in these repetitive regions, this could potentially lead to many false negative calls, and potentially even false positives if repetitive regions are assembled incorrectly, as a loss of a set of repeats could look like a deletion.

3.6 Hybrid Approaches

Recently, many approaches have been published with the goal of combining more than one of the signals of SVs (RP, RD, SR, and AS) in order to improve accuracy. Table 3.1 lists those that we are aware of. In general these can be divided into three types: those that generate candidate SV calls using one primary signal, and then use a second signal to refine or filter those predictions; those that incorporate existing tools as modular components into a pipeline that then merges the candidate calls from each component; and those that have an integrative model of all three signals, typically using a feature-based statistical approach. We will review several examples of each category in the next sections.

Algorithm	Primary Signal	Secondary Signals	Comments
GASVPro [174]	RP	RD	support RP predictions with RD signals
DELLY [157]	RP	SR	refine RP predictions with SR mappings
PRISM [77]	RP	SR	refine RP predictions with SR mappings
Meerkat [198]	RP	SR	refine RP predictions with SR mappings
SoftSearch [66]	RP	SR	support RP predictions with softclips
Bellerophon [68]	RP	SR	support RP predictions with softclips (translocations only)
SVSeq [203]	SR	RP	support SR predictions by at least one discordant RP
CNVer [125]	RD	RP	support RD predictions with discordant RP mappings
inGAP-sv [153]	RD	RP	refine RD predictions with discordant RP mappings
Nord et al. [139]	RD	SR	support RD predictions with softclips
PeSV-Fisher [54]	RP	RD	support RP predictions with RD signals
SVMerge [194]	Pipeline	SR,RP,RD,AS	combines BreakDancer, Pindel, RDXexplorer, others
HugeSeq [90]	Pipeline	SR,RP,AS	combines BreakDancer, Pindel, BreakSeq, others
LUMPY [98]	Pipeline	SR,RP	modular approach, merges breakpoint intervals
iSVP [129]	Pipeline	SR,AS,RP	deletions only; merges GATK, Pindel, BreakDancer
Zinfandel [171]	Integrative	RP,RD	HMM based on RD, RP distances
forestSV [126]	Integrative	RP,RD	Random forest classifiers on RD and RP features
SVM^2 [39]	Integrative	RP,RD	SVM classifiers based on RD and RP features
SVMiner [69]	Integrative	RP,RD	GMM clustering based on RD, RP features

Table 3.1: A summary of published SV detection algorithms that combine more than one sequencing signal. Most use one primary signal and then either discard those predictions not supported by the secondary signal or refine the breakpoints of the primary prediction using secondary data. Pipelines independently run several modules based on different signals and then merge the results. Integrative approaches generate primary predictions based on a statistical model that includes features from multiple signals.

3.6.1 Support from Secondary Signals

One example of a tools that uses one of the four basic signals outlined above but then incorporates other signals into its algorithms to refine the call set is GASVPro [174]. GASVPro is primarily an RP based method, but it uses RD signals to validate its predicted breakpoints, assuming that coverage directly around the breakpoint, and in predicted deleted regions, should be reduced. DELLY [157] and PRISM [77], meanwhile, use RP based approaches to identify candidate SV regions, and then guide an SR search for the exact breakpoints of those SVs. Typically, these approaches that incorporate secondary signals improve the specificity of the primary approach, at the cost of a decrease in sensitivity.

3.6.2 Pipelines

SVMerge [194] and HugeSeq [90] are examples of pipelines that independently execute multiple algorithms of different types and then attempt to merge the results together. The latter ranked calls by the number of components that predicted them, while the former filtered and validated the union of the independent call sets using local assembly. While the integration of these approaches could detect any type of variant detectable by any individual algorithm, it is difficult to combine results from different approaches in a principled manner, and the large number of dependencies and complex parameterization and configuration required has prevented adoption of these pipelines outside of the laboratories in which they were created.

3.6.3 Integrative Models

Finally, several algorithms have been developed that integrate more than one signal at once in a statistical model to generate SV calls. All of these have created features based on components of RD and RP signals. For example, Zinfandel [171] created an HMM where the observations included RD and RP features, and the hidden states included differently sized deletions and duplications with hand-tuned transition probabilities. forestSV [126] and SVM^2 [39] created feature vectors and then trained machine learning classifiers on example data sets; the former demonstrated that it can be useful to include additional features related to genomic context in feature vectors for predicting SVs. In Chapter 7, we will demonstrate a novel integrative approach that incorporates many more features using conditional random fields.

3.7 An Example of an SV Detection Pipeline for a Cancer Dataset

To illustrate how these tools are used in practice, in this section we describe our experiences in developing a pipeline for SV detection using existing tools to try to detect genomic rearrangements in a low-coverage cancer data set. Our pipeline roughly follows the steps we outlined in Section 2.2.1. We will return briefly to this data set in Section 6.4.

The data came from samples taken from a patient with acute myelomonocytic leukemia (AML). Our collaborator Dr. Jeffrey Tyner collected peripheral blood under a written and oral informed consent process reviewed and approved by the Institutional Review Board of Oregon Health & Science University. Known cytogenetic abnormalities associated with this specimen included trisomy 8 and internal tandem duplications within the FLT3 gene. He then isolated cells for sequencing by separating mononuclear cells on a Ficoll gradient, followed by red cell lysis. Mononuclear cells were immunostained using antibodies specific for CD3, CD14, CD34, and CD117 (all from BD Biosciences) and cell fractions were sorted using a BD FACSAria flow cytometer. This enabled us to isolate cell fractions including non-cancerous T-cells (CD3+), malignant monocytes (CD14+), and malignant blasts (CD34, CD117+), which represent an intermediate state between the normal CD3+ T-cells and the CD14+ tumor cells.

We sequenced all three cell isolates on an Illumina Genome Analyzer II, producing 128,819,200 76bp paired-end reads for the CD14+ cells, 194,945,868 reads for the CD3+ cells, and 172,182,321 reads for the CD117+ cells. We then developed a pipeline to detect somatic SVs in the CD14+ tumor cells. This pipeline roughly follows the form described in Section 2.2.1 for variant calling from DNA resequencing data, although we were not interested in SNV calls and therefore did not include them in our pipeline. Our pipeline began with a QC step, in which we gathered statistics on read quality and possible duplication rates using FastQC [8], and then trimmed adapter sequence (artifacts of the Illumina sequencing process) using cutadapt [119]. We then aligned reads to the hGRC37 human reference genome using Novoalign [142]. We chose Novoalign because it has higher accuracy than many other aligners, at a cost of significantly longer runtimes [161]; we were able to make Novoalign runtimes managable by splitting input reads into chunks and simultaneously aligning them across a compute cluster using custom scripts to coordinate the HTCondor grid engine [181]. We then used Picard Tools [24] to identify and remove duplicate reads. To call SVs, we used BreakDancer [37], as a preliminary version of the simulations we will describe in Section 6.1.2 showed that it had slightly better accuracy than other the other tools we evaluated. By subtracting SV calls from the CD3+ normal cells from those called for the CD14+ tumor cells, we were able to identify a set of candidate somatic SV calls which might be associated with the patient's AML.

The QC and deduplication steps of our pipeline revealed that due to the small amount of input DNA in our samples, we had very high duplication rates in our sequencing data, with 15.69% and 85.54% of the reads being duplicates in the CD14+ and CD3+ data sets, respectively. This led to very low physical depths of coverage of the genome: 8.3X for the CD14+ cells and 1.3X for the C3+ cells. Nevertheless, BreakDancer still made over six thousand SV calls, far more than we could verify. Examination of selected calls, together with the knowledge that the cells' karyotype did not include gross rearrangements, showed that many were false positives due to incorrectly aligned reads in repetitive regions of the genome, particularly in areas with many simple repeats such as the regions near chromosome centromeres and telomeres. Therefore, we developed a filtering pipeline that used BreakDancer SV scores and genome annotations, including interval files containing the genomic coordinates of transposable elements, peri-centromeric and telomeric regions, and segmental duplications. This annotation pipeline is open-source and available at `https://github.com/cwhelan/sv-annotate`.

Using this filtering pipeline we were able to reduce the set of candidate SVs to a small number of high-quality calls that we could verify. This list included 38 deletions, 13 inversions, and 14 translocations. With the help of our collaborators Alberto L'Abbate and Tiziana Storlazzi at the University of Bari, Italy, we were able to test a subset of predictions by PCR. They designed PCR primers using Primer3 [185], considering the chromosome localization and orientation of the interval involved in the candidate rearrangement, and checked them for specificity using the BLAT tool of the UCSC Human Genome Browser [80]. All the primer pairs were preliminarily tested on the patient genomic DNA and a normal genomic DNA as control. The PCR conditions were as follows: 2 min at 95°C followed by 35 cycles of 30 sec at 95°C, 20 sec at 60°C, and 2 min at 72°C. All the obtained PCR products were sequenced and analyzed by BLAT for sequence specificity.

Overall, PCR confirmed nine out of eleven tested deletions, two out of seven tested inversions, and only one out of seven tested translocations. Nevertheless, the confirmed SVs included several that might be relevant to the patient's AML, including a deletion in the gene *CTDSPL*/*RBPS3*, an AML tumor suppressor [206]; and another deletion in *NBEAL1*, a gene up-regulated in some cancers [35]. We are currently investigating these deletions to determine their functional impact on this patient.

Our experiences in developing this pipeline illustrate several of the themes of this dissertation. Even for a low-coverage data set, to compute high-quality alignments using

Novoalign we resorted to data parallelization over a compute cluster coordinated by custom scripting, but were unable to speed up the runtime of BreakDancer. In the next chapter, we will discuss methods to easy distribution of sequencing analysis pipelines over compute clusters and to the compute cloud, and develop a truly distributable algorithmic framework for SV detection. In addition, the high false positive rate of our candidate SV calls illustrates the need for more accurate algorithms; we will develop and evaluate a novel algorithm in Chapters 5 and 6, and discuss a possible method for increasing its accuracy in Chapter 7.

Chapter 4

A Framework for SV Detection in MapReduce

In the previous two chapters, we discussed the need for scalable approaches to processing high throughput sequencing data, and explored algorithms for detecting genomic structural variations while noting that efficient processing has not been a primary concern in their design. In this chapter we will explore the MapReduce computing paradigm and its open source Hadoop implementation. We will then discuss ways in which Hadoop has been applied to sequencing tasks. Finally, we will describe a general approach for implementing SV detection algorithms in MapReduce and Hadoop, which we will explore in more detail in later chapters.

4.1 MapReduce and Hadoop

MapReduce [45] is a parallel computing framework designed at Google to handle computation over very large data sets - in particular, the vast amounts of data produced by Google's web crawlers. These volumes of data are too large to store and access efficiently on single instances or storage arrays, and must therefore be stored in distributed file systems (DFS), in which portions of the data are stored on individual nodes distributed throughout the cluster, rather than on a central file server; MapReduce runs in conjunction with the Google File System (GFS) [60]. MapReduce was designed to simplify the development of applications that need to process large volumes of data that are distributed across clusters of many servers, while hiding the complexity of data and processing distribution and load balancing from the application developer. An overriding goal in its design was to develop a robust framework in scenarios where clusters are composed of hundreds or thousands of commodity machines, which may have high failure rates and heterogeneous performance characteristics. In particular, it is optimized to provide the following benefits automatically to applications built according to its programming model:

- **Fault tolerance.** Both data and processing tasks have redundancy built into the application framework due to the expectation of a certain failure rate among the nodes in the cluster. As mentioned earlier, data is distributed over the file system such that individual blocks of data reside on individual nodes in the cluster. Rather than storing single copies of each block, however, the DFS ensures that multiple copies of each block are stored in the file system on different nodes, and if a node fails, will re-replicate blocks to other nodes so that no data is lost. The same philosophy is also applied to the task scheduler/tracker, which breaks up jobs into many small tasks. If the tracker notices an unresponsive task, for example caused by a hardware error on an individual node, it will restart additional tasks to operate on the same input block of data. In this way jobs can continue processing and complete successfully even when worker nodes fail in the middle of their processing.

- **Data locality.** Taking note of the fact that the most expensive part of distributed computing is often transferring data over slower network connections, MapReduce makes every effort to schedule tasks on nodes that physically hold a copy of their input data. This philosophy of "bringing the code to the data" is key to processing large data sets quickly without saturating cluster networks.

- **Scalability.** The framework's goal is allow the size of both the data sets and the cluster to grow seamlessly. As more data is added to a data set, the DFS divides it into blocks and distributes it across available space on the cluster, avoiding bottlenecks in storage capacity, and increasing the number of nodes that have copies of portions of the data set locally. As more nodes are added to the cluster to increase its capacity, they automatically expand the capacity and have data replicated to them. Since tasks are also broken up into small independent pieces, new nodes can also be seamlessly incorporated into the cluster's processing jobs.

Clearly, the desirable properties of the DFS are dependent on being able to divide the data into small blocks that can be distributed easily around the cluster. Similarly, the scalability goals for processing depend on dividing up compute jobs into small tasks that can run independently on different cluster nodes. To achieve this, MapReduce requires that developers structure their application into functional components known, unsurprisingly, as *Map* and *Reduce* tasks. This structure is inspired by a functional technique from the Lisp programming language. In functional programming, `map` is a function that takes a single-argument function m and a list l as arguments and returns a new list which is the result of applying m to every element of l. Meanwhile, `reduce` is a function that takes a binary operator r and a list l, and returns the result $r(r(r(l_1, l_2), l_3), ...)$, applying r to every element of l and a running result. Many operations can be expressed as an application of `map` to a list followed by an application of `reduce`. For example, to find the sum of squares of a list of integers l, one would write `reduce(sum, map(square,` l`))`. Although the semantics of MapReduce's functions are slightly different than those of functional programming [91], its creators hoped to enable the same expressive power while ensuring that applications are decomposed into parallelizable pieces.

In MapReduce, Map tasks are responsible for examining every record in a block of the input set, and emitting information in the form of $\langle key, value\rangle$ pairs. Reduce tasks then take as input all of the values that were emitted by a mapper under a particular key and and produce one or more outputs that summarize or aggregate those values. In order to accomplish the necessary data handling to ensure that reducers receive all of the keys for a particular value, in between the map and reduce phase MapReduce executes a "shuffle and sort" procedure, in which each node sorts the output of all of its mappers by key, sends the results for each key to the machine on which the reducer for that key will run, and then on the reduce machine merges the incoming data from map nodes. This distributed sorting phase is often the key to the efficiency and scalability of MapReduce algorithms.

Figure 4.1 shows the canonical example MapReduce application. The task is to count the number of occurrences of every word that appears in a large text corpus. In this simple implementation, mappers take as input blocks of the text input. For every word w that they encounter, they emit the key/value pair $\langle w, 1\rangle$. Reducers then sum all of the values

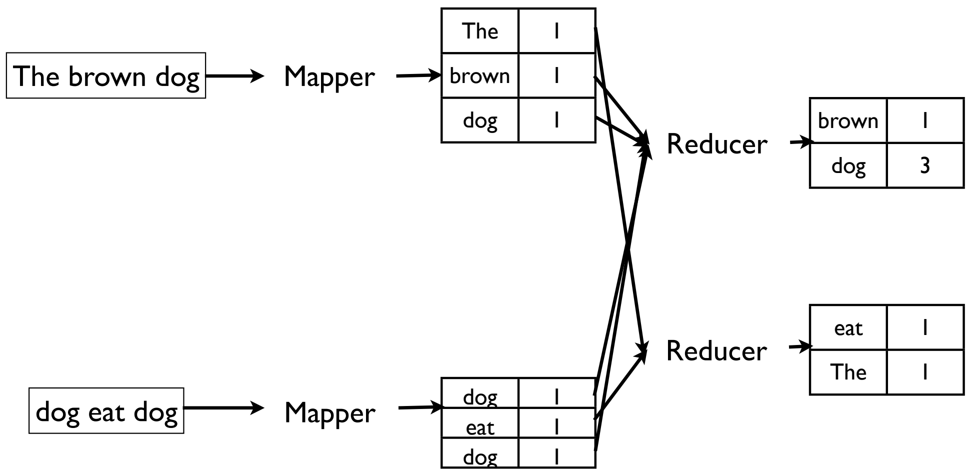

Figure 4.1: The word count example MapReduce application. Mappers examine blocks of text data and emit the value one under a key for each word that they encounter. Reducers sum the values they receive for each word, resulting in a count of the number of times each word occurs in the data set.

for each word key w, which gives the count of occurrences for that word in the input.

Hadoop is the Apache Software Foundation's open source implementation of the framework described by Google in Dean and Gehemewat's MapReduce paper. It includes an implementation of the MapReduce job scheduler, called the *TaskTracker*, as well as a distributed file system that mimics GFS called HDFS. The open source community has developed Hadoop extensively, with support from many corporations including Yahoo! and Facebook, making it a widely adopted and extended platform for distributed computing.

4.2 Uses of Hadoop and MapReduce in Sequencing Tasks

Now that we understand the components of Hadoop and MapReduce, we can explore its use in sequencing-related bioinformatic tasks. Figure 4.2 gives an overview of the published applications of Hadoop to three popular sequencing pipelines: DNA resequencing, RNA-seq, and *de novo* assembly. In some cases, researchers have developed native MapReduce algorithms to solve particular problems. In other instances, Hadoop has been used to run existing tools in parallel; of course, this is only possible when the tasks are embarrassingly parallel at some level. Finally, there are several lower-level APIs, toolkits, and frameworks that have been created in an attempt to ease the development of Hadoop pipelines. These toolkits provide libraries for reading and writing popular sequencing related file formats, wrappers for commonly used tools, and functions to manipulate short read sequences and alignment records. We will discuss each of these pipelines in more detail in the following sections.

4.2.1 DNA Resequencing

As we discussed in Section 2.2.1, there are multiple steps that are typically part of DNA resequencing pipelines. These steps included quality control analysis, alignment to the genome reference, a series of processes including realignment in problematic regions and removal of duplicate reads, steps to call SNVs and SVs, and finally annotation and analysis of the variants that were discovered.

A native algorithm for mapping short reads to a reference genome was first demonstrated in Cloudburst [163]. This algorithm creates keys for each k-mer that appears in both the reads and the reference. The reducers then bring together the reads and portions of the reference that match on the same k-mer, and determine whether the match on the seed can be extended to an alignment of the entire read to the reference. Because the read and reference portions are each transferred to the reducers for every single k-mer which appears in them, this algorithm, while accurate, generates large amounts of data that must be shuffled, sorted, and potentially cross the network. Therefore, it is difficult to scale it to very large read sets. This shows the difficulty of designing MapReduce algorithms that can scale and take advantages of Hadoop's infrastructure. CloudAligner [137] uses a similar seed-and-extend algorithm, although the exact details of their map and reduce steps are not specified.

Since short read mapping is an embarrassingly parallel task and highly optimized non-MapReduce alignment tools with full feature sets are readily available, it is more efficient in practice to use the map phase of a Hadoop job to align splits of input reads with existing tools, and then merge results in the reduce phase. SEAL [151] and the CRS4 Sequencing and Genotyping Platform's CSGP workflow [150] are two such pipelines which use the

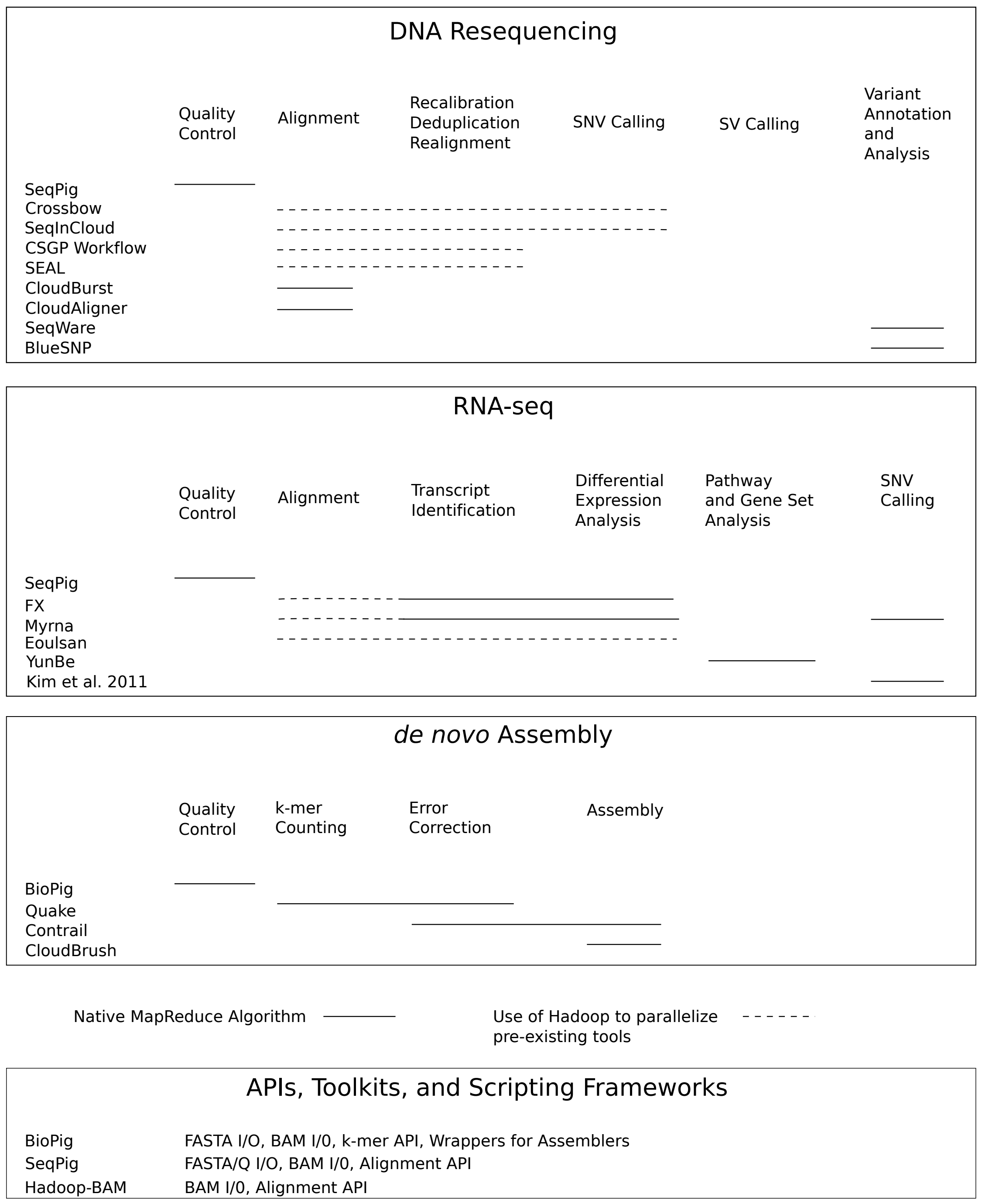

Figure 4.2: Hadoop-enabled tools for tasks of three popular sequencing analysis pipelines. Solid lines indicate native MapReduce applications developed specifically for Hadoop. Dotted lines indicate pipelines that parallelize the execution of existing tools on splits of the input data using Hadoop. In addition, several toolkits and frameworks are listed that are broadly applicable to many portions of these pipelines.

BWA [106] aligner in each map task. By emitting mappings under a key for the alignment location, they are then able to accomplish the duplicate removal step in the reduce function in a manner similar to that of the popular non-MapReduce tool Picard [24]. Similarly, Crossbow [95] uses Hadoop to parallelize Bowtie [96]. Crossbow then calls single nucleotide variants in the reduce phase using SOAPsnp [109], allowing it to function as a minimum viable pipeline for sequencing. Crossbow also includes infrastructure for running in the Amazon Elastic Compute Cloud, which has contributed to its popularity. SeqInCloud [131] similarly combines BWA with the GATK's [124] realignment and SNV calling steps, along with infrastructure to run in the Microsoft Azure cloud computing platform.

After variant discovery, the usual next step in a resequencing project is to analyze the variants found to detect functional impact (i.e. what genes might be disrupted by the variants found) or to find associations between variants and phenotypes, as in case/control sequencing projects. SeqWare [143] uses HBase [11], a Hadoop database based upon Google's BigTable [32], to provide functionality to annotate and query the variants. SeqWare uses the genomic coordinates of variants and coverage information as schema keys within HBase's MapReduce queries. BlueSNP [74], meanwhile, implements Hadoop algorithms for finding significant loci and estimating p-values through permutation analysis in genome wide association studies.

To our knowledge, there are no publicly available Hadoop/MapReduce software tools for SV detection. Several non-Hadoop pipelines such as HugeSeq [90] and SVMerge [194] use grid scheduling engines like SGE to distribute SV detection tasks across compute clusters. For example, the HugeSeq pipeline is an end-to-end pipeline for DNA resequencing that integrates BreakDancer [37], Pindel [200], CNVnator [2], and BreakSeq [89] for SV calling. However, these pipelines can only achieve parallelization by sample or at most by chromosome of the reference, limiting their scalability.

4.2.2 RNA-seq

The goals of RNA-seq are to determine the transcripts and isoforms being expressed, quantify their differential expression, and potentially call DNA variants based on the sequences of the RNA transcripts. Myrna [93] and FX [71] are Hadoop pipelines that execute alignment, transcript identification, and differential expression calculations in multiple Hadoop jobs. Myrna uses Bowtie in each mapper to distribute alignment work across the cluster, and then parallelizes each of the differential expression calculations by first assigning alignments to transcripts, gathering the data by sample to compute normalization factors, and then re-gathering the normalized counts under keys representing transcripts to compute statistics. FX has a less complex workflow, in which alignments are executed in a map-parallel fashion using the GSNAP alignment tool [195], and then has simple steps to count reads by gene for differential expression statistics, and to call SNPs in genes. Both Myrna and FX have infrastructure to automatically create Hadoop clusters in Amazon EC2. Eoulsan [78] is a very similar workflow, which allows a number of aligners to be used in the mapping step, and then computes read counts per gene in parallel, before computing differential expression statistics outside of the Hadoop environment using existing tools. For use after expression levels have been computed, YunBe [204] computes statistics to determine whether particular gene pathways have been perturbed in two sets of samples, using a MapReduce strategy where mappers produce expression values from the sample data under keys that represent specific pathways, and reducers aggregate all data for each pathway to determine a “perturbation statistic” between sample conditions. Finally, Kim et al. [81] described a pipeline for SNV detection from RNA-seq data based on a Hadoop job in which the mappers emit each base and quality score from the aligned reads under a key representing the genomic coordinate, and reducers call the absence or presence of a SNV at each location.

4.2.3 *de novo* Assembly

The third major pipeline for which there has been significant interest in applying Hadoop and MapReduce is *de novo* assembly. Most state-of-the-art algorithms for assembly depend on building large data structures that model the overlaps between reads (in the case of string graph based approaches) or between k-mers found in the reads (in the case of de Bruijn-graph algorithms) as edges in a graph. Given the high depth of coverage that is needed to produce a quality assembly, these graphs become very large, and the need to traverse them requires random rather than sequential access to the data. This would seem to make the problem a poor fit for Hadoop, which is optimized for sequential access of large data sets on commodity hardware. Nevertheless, several groups have attempted to create assembly algorithms using MapReduce. Contrail [165] builds a de Bruijn graph by emitting each pair of consecutive k-mers from the reads as a key-value pair to form a k-mer adjacency matrix. CloudBrush [33] uses a "prefix-and-extend" strategy [34], in which all k-mers that appear in the reads are used as keys, with the reads themselves as values. Extension procedures then test whether the k-mer is the prefix of one of the reads and the suffix of another. Both of these tools have then developed MapReduce procedures for graph pruning and traversal that are necessary to produce contigs from their respective graph data structures. Finally, some helpful read pre-processing steps for assembly have been implemented in Hadoop: Quake [79] is a widely used MapReduce k-mer counter, and BioPig [140] also contains facilities for k-mer analysis of large read sets.

4.2.4 Frameworks and Toolkits

In addition to the pipelines listed above, recently several sequencing-related frameworks and toolkits have been created to ease the development of Hadoop applications. These tools include APIs and library code to handle reading from and writing to files in popular sequencing data formats. For example, Hadoop-BAM [138] allows manipulation of data in the SAMtools [107] binary alignment format BAM. Two libraries, SeqPig [167] and BioPig [140], were also recently published that provide file formats and commonly used functions in the Apache Pig [12] environment, which provides a high-level programming language called Pig Latin that allows easy expression of simple data analysis tasks. These libraries allow for rapid development of scripts that gather statistics on read and alignment data sets, and Pig automatically optimizes their execution by translating them into one or more MapReduce jobs.

4.2.5 Other uses of Hadoop and MapReduce

Although not implemented in Hadoop, the very widely used Genome Analysis Toolkit [124] (GATK) implements many sequencing and variant calling functions using a MapReduce programming model. Although it currently can only distribute tasks across clusters using traditional grid engines, future versions of this package may be re-implemented to use Hadoop. Other notable sequencing applications that have been implemented in Hadoop include ChIP-seq peak calling [57], and computing genome mappability scores [100].

4.3 MapReduce Constraints on SV Algorithms

To this point, we know of no Hadoop-based SV detection algorithms. This may be because the need to separate logic into mappers and reducers makes it difficult to implement traditional RP-based SV detection approaches in MapReduce, particularly given the global clustering of paired end mappings at the heart of many RP approaches. MapReduce algorithms, by contrast, excel at conducting many independent calculations in parallel. The sequencing applications that have been implemented in MapReduce succeed by dividing processing into a series of local computations, for example calling a SNV at a particular location in the genome given the reads that cover it, to use the example of Crossbow [95], the most widely accepted sequencing MapReduce algorithm. As we noted in Chapter 3, SV approaches that are similarly based on local computations have been described: the RP-based SV callers MoDIL [102] and forestSV [126] try to solve the SV detection problem by computing scores or features along the genome and then producing SV predictions from those features in a post-processing step. In the remainder of this chapter, we formalize this idea and develop it as a potential framework for implementing SV detection algorithms in MapReduce.

4.4 A General MapReduce SV Detection Algorithm

Using the strategy of computing local features along the genome, we have developed a conceptual algorithmic framework for SV detection in MapReduce, which is outlined in Algorithm 1. This framework divides processing into three separate MapReduce jobs: an alignment job, a feature computation job, and an SV calling job. The overall workflow of the algorithm and its implementation on a compute cluster or cloud is summarized in Figure 4.3.

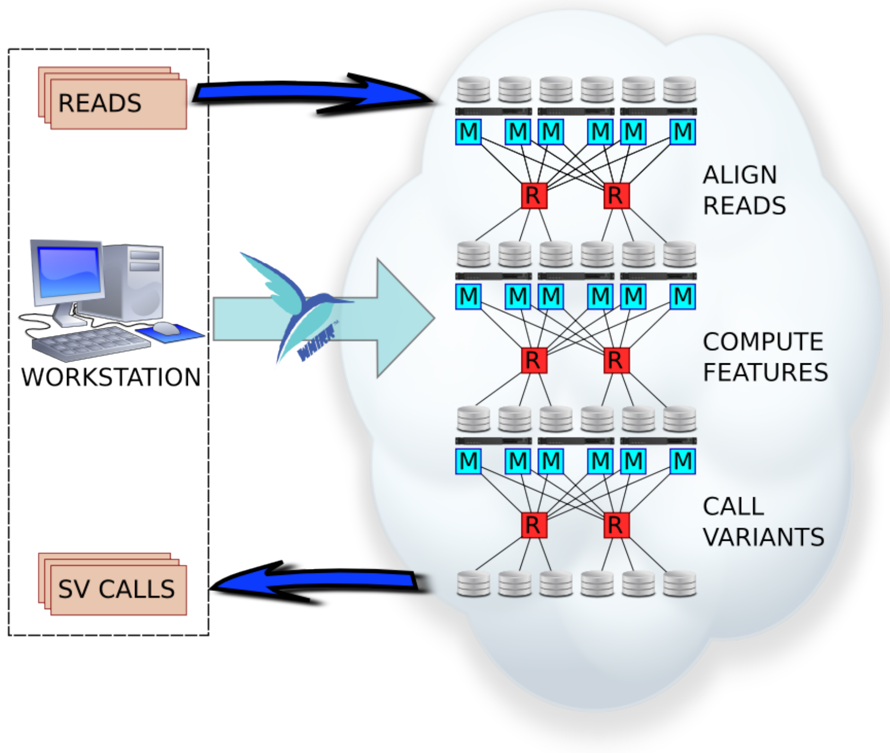

Figure 4.3: An overview of the steps of the MapReduce SV detection workflow. Reads are first uploaded to a Hadoop cluster from local storage. Processing is then then divided into three MapReduce jobs: 1) Mapping with sensitive settings. 2) Computation of features across the genome. 3) Calling structural variations based on the features computed in the previous step. Finally, SV predictions can be downloaded from the Hadoop cluster and examined and annotated. Cloudbreak can also use the Apache Whirr library to automatically provision Hadoop clusters on and deploy data to cloud providers such as Amazon Elastic Compute Cloud.

The Align Reads job uses existing alignment tools to discover mapping locations for each read pair. Aligners can be executed to report multiple possible mappings for each read, or only the best possible mapping. Given a set of read pairs, each of which consists of a read pair identifier *rpid* and two sets of sequence and quality scores $< s, q >$, each mapper aligns each pair end set $< s, q >$ in either single- or paired end mode and emits possible mapping locations under the *rpid* key. Reducers then collect the alignments for each paired end, making them available under one key for the next job.

In the Compute Features job, we compute a set of features for each location in the genome. To begin, we tile the genome with small fixed-width, non-overlapping intervals. For the experiments reported in Chapter 6 we use an interval size of 25bp (see Section 6.2.2 for an exploration of the effects of different window sizes on accuracy and runtime). Let $L = \{l_1, l_2, \ldots, l_N\}$ be the set of intervals covering the entire genome. Let $R^1 = \left\{r_1^1, r_2^1, \ldots, r_M^1\right\}$ and $R^2 = \left\{r_1^2, r_2^2, \ldots, r_M^2\right\}$ be the input set of paired reads. Let $A^1 = \left\{a_{m,1}^1, a_{m,2}^1, \ldots, a_{m,K}^1\right\}$ and $A^2 = \left\{a_{m,1}^2, a_{m,2}^2, \ldots, a_{m,L}^2\right\}$ be the set of alignments for the left and right reads from read pair m. For any given pair of alignments of the two reads in a read pair, $a_{m,i}^1$ and $a_{m,j}^2$, let the ReadPairInfo $rpi_{m,i,j}$ be information about the pair relevant to detecting SVs, e.g. the fragment size implied by the alignments and the likelihood that the alignments are correct. We then leave two functions to be implemented depending on the application:

$$\begin{aligned} \textsc{Loci} \ &: \langle a_{m,i}^1, a_{m,j}^2 \rangle \rightarrow L_m \subseteq L \\ \Phi &: \{\text{ReadPairInfo } rpi_{m,i,j}\} \rightarrow \mathbb{R}^N \end{aligned}$$

The first function, Loci, maps an alignment pair to a set of genomic locations to which it is relevant for SV detection; for example, the set of locations overlapped by the internal insert implied by the read alignments. We optimize this step by assuming that if there exist concordant mappings for a read pair, defined as those where the two alignments are in the proper orientation and with an insert size within three standard deviations of the expected library insert size, one of them is likely to be correct and therefore we do not consider any discordant alignments of the pair. The second function, Φ, maps a set of ReadPairInfos relevant to a given location to a set of real-valued vectors of features useful for SV detection.

Finally, the third MapReduce job, Call Variants, is responsible for making SV calls based on the features computed at each genomic location. It calls another application-specific function:

$$\textsc{PostProcess} : \{\phi_1, \phi_2, \ldots, \phi_N\} \rightarrow \{\langle \text{SVType } s, l_{start}, l_{end} \rangle\}$$

This function maps the sets of features for related loci into a set of SV calls characterized by their type s (i.e Deletion, Insertion, etc.) and their breakpoint locations l_{start} and l_{end}.

Algorithm 1 The algorithmic framework for SV calling in MapReduce.

```
1: job ALIGNMENT
2:     function MAP(ReadPairId $rpid$, ReadId $r$, ReadSequence $s$, ReadQuality $q$)
3:         for all Alignments $a \in$ ALIGN($< s, q >$) do
4:             EMIT(ReadPairId $rpid$, Alignment $a$)
5:     function REDUCE(ReadPairId $rpid$, Alignments $a_{1,2,\dots}$)
6:         AlignmentPairList $ap \leftarrow$ VALIDALIGNMENTPAIRS($a_{1,2,\dots}$)
7:         EMIT(ReadPairId $rp$, AlignmentPairList $ap$)
8: job COMPUTE SV FEATURES
9:     function MAP(ReadPairId $rp$, AlignmentPairList $ap$)
10:        for all AlignmentPairs $< a_1, a_2 > \in ap$ do
11:            for all GenomicLocations $l \in$ LOCI $(a_1, a_2)$ do
12:                ReadPairInfo $rpi \leftarrow <$ InsertSize($a_1, a_2$), AlignmentScore($a_1, a_2$) $>$
13:                EMIT(GenomicLocation $l$, ReadPairInfo $rpi$)
14:    function REDUCE(GenomicLocation $l$, ReadPairInfos $rpi_{1,2,\dots}$)
15:        SVFeatures $\phi_l \leftarrow \Phi$(InsertSizes $i_{1,2,\dots}$, AlignmentScores $q_{1,2,\dots}$)
16:        EMIT(GenomicLocation $l$, SVFeatures $\phi_l$)
17: job CALL SVS
18:    function MAP(GenomicLocation $l$, SVFeatures $\phi_l$)
19:        EMIT(Chromosome($l$), $< l, \phi_l >$)
20:    function REDUCE(Chromosome $c$, GenomicLocation $l_{1,2,\dots}$, $\phi_{1,2,\dots}$)
21:        StructuralVariationCalls $svs_c \leftarrow$ POSTPROCESS ($\phi_{1,2,\dots}$)
```

We parallelize this job in MapReduce by making calls for each chromosome in parallel, which we achieve by associating a location and its set of features to its chromosome in the map phase, and then making SV calls for one chromosome in each reduce task.

4.5 Discussion

The framework that we have just described is agnostic to the type of structural variations that the user wishes to detect. In the next chapter, we describe Cloudbreak, our implementation of this framework that identifies small deletions and insertions. To do so, it defines the relevant information from each pair (the ReadPairInfo) as information about the insert size implied by the mapping of the paired reads, sends them to every location which is spanned by that read pair in the reference genome, and then computes features from them by modeling the expected distribution of insert sizes at each location with a

Gaussian Mixture Model. We demonstrate some of the strengths of this particular implementation in Chapter 6. However, we believe that this general framework could be applied to many other SV detection problems. For example, to detect inversions, one might define a different ReadPairInfo for each pair that includes information about the orientation of the mappings. Translocation detection programs might emit ReadPairInfos that contain information about the chromosomes linked to by interchromosomal mappings, which the Φ function would then cluster to see if a preponderance of those mappings point to the same breakpoint location. In addition to paired mapping locations, it would also be possible to design ReadPairInfo and feature function definitions that allowed the computation of read depth or split read related features, enabling the integration of more of the signals available in the data set. We will explore one possible way to integrate disparate features such as these in Chapter 7. Use of the Hadoop/MapReduce computing framework would ensure that any of these applications, if carefully designed, could enjoy the benefits of redundancy, data locality, and scalability we described in Section 4.1, and therefore will be able to grow to meet the rising demands of genomics applications in the near future.

Chapter 5

Cloudbreak

In the previous chapter, we described and formalized a general strategy for building SV detection algorithms in MapReduce and Hadoop. In this chapter, we describe an software package, Cloudbreak, that we have developed using this algorithmic framework. To build Cloudbreak, we implemented the infrastructure necessary to support the algorithmic framework we described in Section 4.4, and also provided implementations of the three application-specific functions we described there. Here we will describe this implementation, as well as our choices for the three user-defined functions we specified in the framework definition, and additional functionality we developed to genotype calls and to facilitate the deployment of Cloudbreak on public cloud platforms including Amazon's EC2. In Chapter 6, we will provide an evaluation of the algorithm's accuracy and performance characteristics.

5.1 Variant types detected

Cloudbreak is our implementation of a detection algorithm for genomic deletions (40bp-25,000bp) and small insertions based on examining the insert sizes of paired end mappings. We chose this application because small deletions in the range of 50bp to 150bp are particularly difficult to detect using many existing SV algorithms [4, 127]. This is because most read-pair based algorithms use a hard cutoff based on the variance of the fragment size distribution to select discordant read pairs, as described in Section 3.2. By taking advantage of many compute cores using MapReduce, we can design an algorithm that considers all of available data (both concordant and discordant read pairs) in a generative statistical framework, as we will describe in Section 5.3.

5.2 Framework infrastructure

In addition to the specific algorithm for detecting deletions and insertions (which take the form of implementations of the user-defined functions we described in the previous chapter), the Cloudbreak package also contains the infrastructure necessary to implement the three MapReduce jobs defined in our MapReduce algorithmic framework. Providing a fully featured, multi-job Hadoop application requires several implementation decisions:

- **Programming language and method of interacting with Hadoop.** Hadoop applications can be developed in several ways. A native application is written in Java and directly uses the Hadoop application programming interface (API) to start jobs, implement Map and Reduce functions, and set advanced Hadoop configurations. Alternatively, applications can be developed in any language using the Hadoop *streaming* interface, as long as the input to and output from all map and reduce tasks is textual and certain conventions are followed with respect to data format. Finally, for C/C++ applications, the Hadoop *pipes* interface can be used to marshal input and output data from tasks. Each method has its own advantages and disadvantages. Native applications constrain the developer to Java but enjoy the best performance when the jobs are data intensive rather than CPU bound [47]. Streaming applications allow greater programming language flexibility but make it somewhat more difficult to organize complex applications and take advantage of advanced Hadoop features. Pipes, meanwhile, allow for maximum performance for CPU intensive applications. We opted to develop Cloudbreak as a native application to take advantage of the tight integration with the Hadoop API improved I/O performance on data intensive portions of the workflow. For the ALIGN READS job, which executes existing short-read alignment tools, a mapper class written in Java invokes the external tools using the system runtime environment.

- **File formats and compression.** Hadoop applications usually store their data in HDFS in text format or in sequence files, a binary format that allows numeric or complex data types to be stored in a key-value pair structure easily accessed by Hadoop. In addition, varying levels of compression can be used, although only certain compression types allow Hadoop to automatically split large files by HDFS blocks for processing by different map tasks, which is a key consideration for building fully parallelized applications. Cloudbreak uses sequence files for input and intermediate files, although for alignments the values are stored as text strings containing full records in the Sequence Alignment Map (SAM) format [107], to allow for easy exports of data. For compression we use the Snappy [63] compressor/decompressor (codec), a compression scheme developed at Google which aims for reasonable file size reduction with very fast compression and decompression speeds. This makes it ideal for data-intensive Hadoop applications. Given that the output data from the Align Reads job is alignment records, a future implementation goal is to switch to using Hadoop-BAM [138], a library for storing SAM/BAM records efficiently in HDFS; Hadoop-BAM did not posses necessary functionality at the time of Cloudbreak's initial implementation and so we proceeded with a text-based representation of alignment records.

- **Distribution of auxiliary files.** In some cases all tasks require access to large input files, such as genome reference indices for alignment, or genome annotation files. Hadoop offers a *distributed cache* service, which places copies of the files on each node that will host tasks for the job so that they will not all need to copy the files over the network. Cloudbreak makes use of the distributed cache to distribute index and annotation files, as well as the alignment executables for the Align Reads job.

Our implementation of the Align Reads job contains wrappers to execute the aligners BWA [106], GEM [117], Novoalign [142], RazerS 3 [190], mrFAST [5], and Bowtie 2 [94]. This job can also be skipped in favor of importing a pre-aligned BAM file directly into HDFS. The code is structured in such a way that to add a new aligner, developers would simply create a class that finds the necessary index files in the distributed cache and determines the proper command line parameters for aligner execution, and parses the aligner output if it is in a non-standard format.

Cloudbreak can be executed on any Hadoop cluster; Hadoop abstracts away the details of cluster configuration, making distributed applications portable. We deployed Cloudbreak on an internal 56-node cluster running the Cloudera CDH3 Hadoop distribution, version 0.20.2-cdh3u4. In addition, Cloudbreak can create and operate on Hadoop clusters located in commercial and private compute clouds (see Section 5.6).

The source code and user manual for Cloudbreak are publicly available at `https://github.com/cwhelan/cloudbreak`. We hope that by publishing under an open-source license, we will facilitate the adoption of our Cloudbreak implementation, as well as provide a base from which other computational researchers can develop their own SV detection algorithms for Hadoop.

5.3 Implementation of a MapReduce SV Algorithm

In Section 4.4, we described three user-defined functions that can be implemented to create an SV detection application in our MapReduce framework. These functions were named Loci, Φ, and PostProcess. These functions map aligned read pairs to locations on the genome to which they are relevant, compute a set of local features for each genomic location based on the relevant read pairs, and call variants based on the features computed for neighboring genomic locations. Cloudbreak contains implementations of these functions that combine to allow it to detect deletions (of size 40bp-25,000bp) and insertions. A detailed description of each of these implementations appears below, and an illustration of each phase of the Cloudbreak algorithm working on a simple example in MapReduce is shown in Figure 5.1.

Loci Because we are detecting deletions and short insertions, we map ReadPairInfos from each possible alignment to the genomic locations overlapped by the implied internal insert between the reads. For efficiency, we define a maximum detectable deletion size of 25,000bp, and therefore alignment pairs in which the ends are more than 25kb apart, or in the incorrect orientation, map to no genomic locations. In addition, if there are multiple possible mappings for each read in the input set, we optimize this

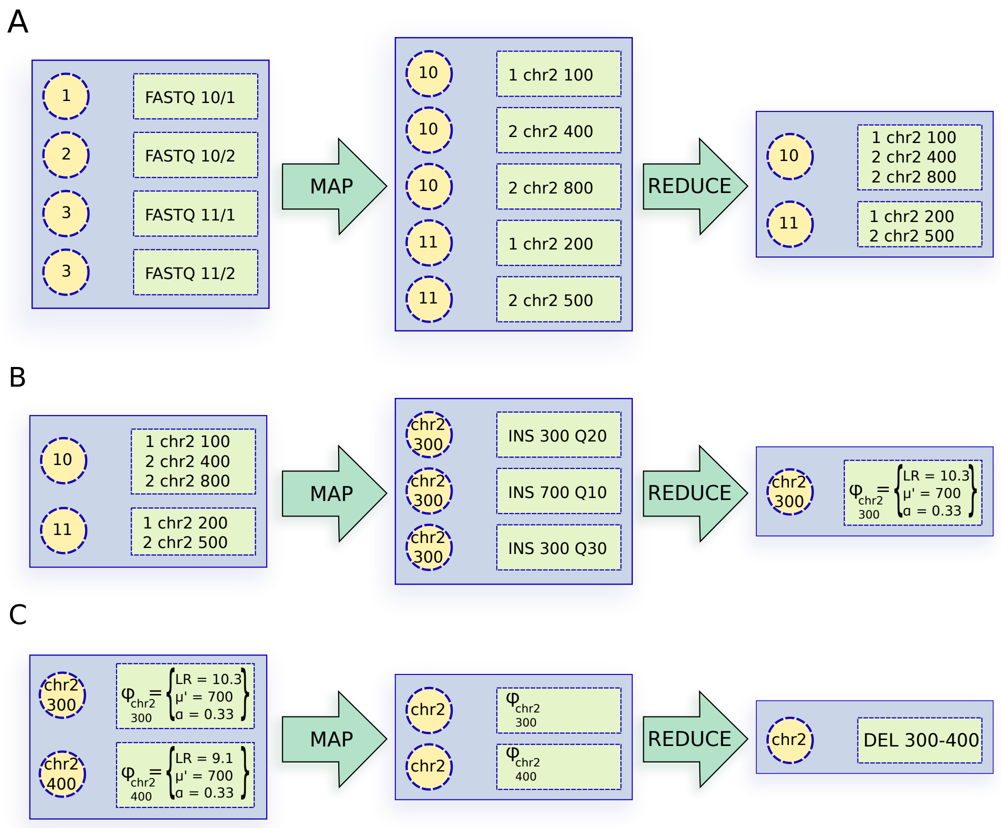

Figure 5.1: An Example of the Cloudbreak deletion and insertion detection algorithm running in MapReduce. A) In the first MapReduce job, mappers scan input reads in FASTQ format and execute an alignment program in either paired-end or single-ended mode to generate read mappings. Reducers gather all alignments for both reads in each pair. B) In the second MapReduce job, mappers first emit information about each read pair (in this case the insert size and quality) under keys indicating the genomic location spanned by that pair. Only one genomic location is diagrammed here for simplicity. Reducers then compute features for each location on the genome by fitting a GMM to the distribution of spanning insert sizes. C) Mappers group all emitted features by their chromosome, and reducers find contiguous blocks of features that indicate the presence of a variant.

step by assuming that if there exists a concordant mapping for a read pair, defined as a mapping pair in which the two alignments are in the proper orientation and with an insert size within three standard deviations of the expected library insert size, it is likely to be correct and therefore we do not consider any discordant alignments of the pair.

Φ To compute features for each genomic location, we follow the mixture of distributions approach (see Section 3.2.2) first described by Lee et al. [102], who observed that if all mappings are correct, the insert sizes implied by mappings which span a given genomic location should follow a Gaussian mixture model (GMM) whose parameters depend on whether a deletion or insertion is present at that locus. Figure 5.2 shows several examples of the mixtures observed for various types of variants. If there is no indel, the insert sizes implied by spanning alignment pairs should follow the distribution of actual fragment sizes in the sample, which is typically modeled as normally distributed with mean μ and standard deviation σ. If there is a homozygous deletion or insertion of length l at the location, μ should be shifted to $\mu + l$, while σ will remain constant. Finally, in the case of a heterozygous event, the distribution of insert sizes will follow a mixture of two normal distributions, one with mean μ, and the other with mean $\mu + l$, both with an unchanged standard deviation of σ, and mixing parameter α that describes the relative weights of the two components. The features generated for each location l include the log-likelihood ratio of the filtered observed data points under the fit GMM to their likelihood under the distribution $N(\mu, \sigma)$, the final value of the mixing parameter α, and μ', the estimated mean of the second GMM component.

The choice of a mixture of distributions model has several benefits. Firstly, it is a generative model for the entire data set, including concordant and discordant read pairs. This removes the need to set hard thresholds that define discordant read pairs, and allows the detection of smaller variants given a tight enough insert size distribution. Second, the parameters that are estimated can be used to refine and classify predictions. For example, the mixing parameter α can be used to genotype variants, as we will describe in Section 5.5. In addition, the estimated μ' parameter gives a prediction for how many genomic locations the variant might cover, which we leverage in the PostProcess function described below to integrate local features into variant calls.

To implement our model, at each genomic location we fit the parameters of the GMM using the Expectation-Maximization algorithm. Let $Y = y_{1,2,\ldots m}$ be the observed insert sizes at each location after filtering, and say the library has mean fragment size μ with standard deviation σ. Because the mean and standard deviation of the fragment sizes are selected by the experimenter and therefore known *a priori* (or at least easily estimated based on a sample of alignments), we only need to estimate the mean of the second component at each locus, and the mixing parameter α. Therefore, we initialize the two components to have means μ and $\bar{Y}$, set the standard deviation of both components to σ, and set $\alpha = .5$. In the E step, we compute for each y_i and GMM component j the value $\gamma_{i,j}$, which is the normalized likelihood that y_i was drawn from component j. We also compute $n_j = \sum_i \gamma_{i,j}$, the relative contributions of the data points to each of the two distributions. In the M step, we update α to be $n_2 - |Y|$, and set the mean of the second component to be $\frac{\sum_m \gamma_{m,2} y_m}{n_2}$. We treat the variance as fixed and do not update it, since under our assumptions the standard deviation of each component should always be σ. We repeat the E and M steps until convergence, or until a maximum number of steps has been taken. Prior to fitting the GMM at each location, we attempt to filter out incorrect mappings for that location using an outlier-detection based clustering scheme and an adaptive mapping quality cutoff; see Section 5.4 for details.

PostProcess The third MapReduce job is responsible for making SV calls based on the features defined above. We convert our features along each chromosome to insertion and deletion calls by first extracting contiguous genomic loci where the log-likelihood ratio of the two models is greater than a given threshold. To eliminate noise we apply a median filter with window size 5. We end regions when μ' changes by more than 60bp (2σ), and discard regions where the average value of μ' is less than μ or where the length of the region differs from μ' by more than μ.

5.4 Filtering Incorrect and Ambiguous Mappings

One of our goals in developing Cloudbreak was to see if the use of distributed computing in the MapReduce framework would allow us to develop an algorithm that could process multiple mappings for ambiguously mapped reads in a reasonable amount of time, especially

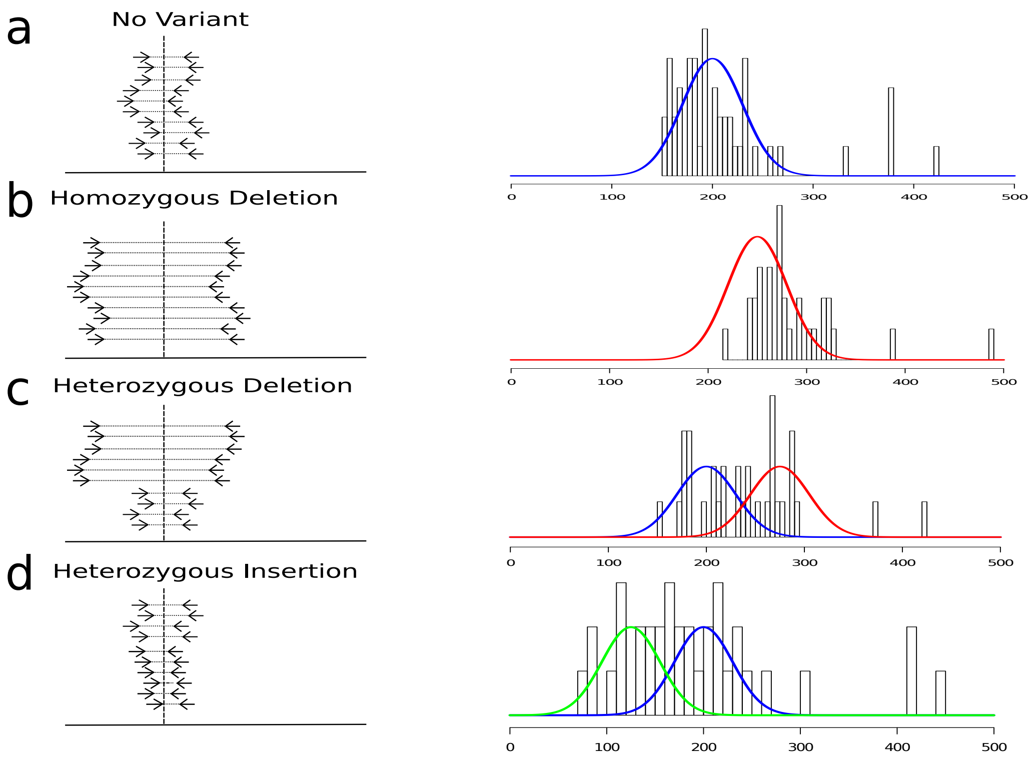

Figure 5.2: Illustration of insert size mixtures at individual genomic locations. A) there is no variant present at the location indicated by the vertical line (left), so the mix of insert sizes (right) follows the expected distribution of the library centered at 200bp, with a small amount of noise coming from low-quality mappings. B) a homozygous deletion of 50bp at the location has shifted the distribution of observed insert sizes. C) A heterozygous deletion at the location causes a mixture of normal and long insert sizes to be detected. D) A heterozygous small insertion shifts a portion of the mixture to have lower insert sizes.

because the size of data sets can grow very large if many possible mappings are kept for all ambiguous reads. Most SV detection tools that use multiple mappings attempt to identify the correct mapping for each ambiguously mapped pair; for example, GASVPro [174] uses an MCMC approach to sample from the distribution of possible mapping locations for each read, and VariationHunter [72] attempts to assign mappings to reads through a combinatorial optimization approach. Cloudbreak, in contrast, attempts to solve this problem at the genomic location level by filtering mappings during the feature computation step.

To handle incorrect and ambiguous mappings, we assume that in general they will not form normally distributed clusters in the same way that correct mappings will, and therefore use an outlier detection technique to filter the observed insert sizes for each location. We sort the observed insert sizes and define as an outlier an observation whose kth nearest neighbor is more than $n\sigma$ distant, where $k = 3$ and $n = 5$. In addition, we rank all observations by the estimated probability that the mapping is correct and use an *adaptive quality cutoff* to filter observations: we discard all observations where the estimated probability the mapping is correct is less than the score of the maximum quality observation minus a constant c. This allows the use of more uncertain mappings in repetitive regions of the genome while restricting the use of low-quality mappings in unique regions. Defining $\textsc{Mismatches}(a)$ to be the number of mismatches between a read and the reference genome in the alignment a, we approximate the probability p_c^k of each end alignment being correct by:

$$p_c^k(a_{m,i}^k) = \frac{\exp(-\textsc{Mismatches}(a_{m,i}^k)/2)}{\sum_j \exp(-\textsc{Mismatches}(a_{m,j}^k)/2)}$$

By multiplying $p_c(a_{m,i}^1)$ and $p_c(a_{m,i}^2)$, we can approximate the likelihood that the pair is mapped correctly.

5.5 Genotyping

In theory, it should be possible to use the parameters of the fit GMM to infer the genotype of each predicted variant. Assuming that our pipeline is capturing all relevant read mappings near the locus of the variant, the genotype should be indicated by the estimated parameter α, the mixing parameter that controls the weight of the two components in the GMM. We set a simple cutoff on the average value of α for each prediction to call the predicted variant homozygous or heterozygous, and use the same cutoff for deletion and insertion predictions. Somewhat surprisingly, we observed on the cutoff point that distinguishes homozygous from heterozygous variants is significantly less than the expected .5; based on empirical observations on a simulated data set, we set the threshold to .35 (see Section 6.5 for more details).

5.6 Running in the Cloud

In Section 2.2.2, we discussed the increasing recognition of the genomics community of the need for tools that ease the use of cloud computing resources [164, 178]. This has spurred the development of a variety of applications and toolkits that use the APIs of Infrastructure as a Service (IaaS) providers, such as Amazon EC2, to allocate compute resources on public compute clouds. In this section, we will briefly review some of the existing cloud-enabled tools, including those that are based on traditional grid scheduling engines and those that enable Hadoop and MapReduce pipelines. Many of the latter take advantage of Amazon's Elastic MapReduce (EMR) service, an offering from Amazon specifically for creating and running Hadoop jobs that simplifies the creation and monitoring of clusters on EC2. We will then discuss how Cloudbreak enables the use of cloud computing in a provider-neutral way using the Apache Whirr library.

5.6.1 Cloud-Enabled Genomics Tools

Several groups have created general-purpose toolkits for creating, allocating, and managing cloud resources for biological data processing. For example, the CloudBioLinux project [86] provides virtual machine images for use in Amazon EC2 that are pre-configured with a wide range of open-source bioinformatics applications. Galaxy CloudMan [3] allows for the automatic creation of clusters in the Amazon cloud configured to run workflows in the popular Galaxy environment [61], backed by the SGE grid scheduling engine and with interfaces to Amazon's S3 storage service. CloVR [9] is a cloud cluster manager that includes several pipelines for metagenomics managed by SGE. Finally, Elastream [76] is a newer offering that can provision and manage cloud clusters using either grid schedulers or EMR.

Several commercial providers are also offering services that allow elastic cloud computing for sequencing pipelines, including DNAnexus (Mountain View, CA), Illumina's (San Diego, CA) BaseSpace cloud, Seven Bridges Genomics' (Cambridge, MA) IGOR platform, and an offering from Biodatomics (Bethesda, MD). These commercial cloud services are in many cases backed by Amazon's EC2, but are wrapped in additional sequencing-specific

APIs and user interfaces by the vendor. To our knowledge Biodatomics is the only commercial bioinformatics vendor that offers automatic Hadoop cluster provisioning, however.

There are also a number of bioinformatics suites designed for specific applications that are able to allocate resources on EC2 automatically. Apart from those that use Hadoop, which we will discuss in the next paragraph, these include: Atlas2 Cloud [56], which allows users to run the Genboree Workbench workflow for variant calling and annotation in personal genomics DNA resequencing projects; SIMPLEX [59], which enables cloud processing for an exome alignment and variant calling pipeline based on BWA and the GATK; a set of ChIP-seq tools of the modENCODE and ENCODE projects [183] that can create clusters to run analysis pipelines on EC2 or the Bionumbus private cloud [145].

Finally, several of the Hadoop applications we discussed in Section 4.2 come with command-line or graphical interfaces that automate their deployment on commercial cloud services. Most of these leverage EMR. The most widely used EMR-enabled tool is Crossbow [95], which has a UI that will create Hadoop workflows using EMR. Crossbow's ability to use EMR was recently made more robust by Rainbow [205], which wraps Crossbow with additional facilities for transferring large files in parallel, monitoring clusters for failing nodes, and aggregating results from multiple samples run simultaneously. As mentioned in Section 4.2, FX [71] and Euolsan [78] are both Hadoop workflows for RNA-seq processing; both have the ability to automatically create EMR jobs on Amazon EC2.

5.6.2 Enabling Cloud Computing with Whirr

In contrast to the tools mentioned above, Cloudbreak leverages the Apache Whirr [13] library to automatically create Hadoop clusters in the cloud. Whirr differs from the strategies referred to in the last section in that it provides a unified application programming interface (API) for provisioning and running cloud services that is agnostic to the cloud IaaS provider. Although the largest and most popular cloud IaaS is provided by Amazon Web Services though their Elastic Compute Cloud (EC2), Whirr is a facade API that allows the transparent substitution of other cloud services such as Rackspace, Microsoft Azure, or private clouds such as those built with Eucalyptus. Most of the Hadoop-enabled applications and toolkits mentioned in the previous section depend on Amazon Elastic

MapReduce to provision Hadoop clusters. Cloudbreak's use of Whirr breaks this dependency on a single vendor.

Using Whirr's API, Cloudbreak is able to provision Hadoop clusters which can then be terminated when processing is complete. This eliminates the need to invest in a standing cluster and allows a model in which users can scale their computational infrastructure as their need for it varies over time. Through a command line interface, Cloudbreak uses Whirr functionality to offer commands that:

- **Automatically provision Hadoop clusters in the cloud.** After specifying the parameters of the Hadoop cluster desired in a property file, a single Cloudbreak command will request the creation of the necessary compute instances in the cloud, will configure them with the proper Hadoop services to provide a fully functioning cluster, and will start proxies that make the Hadoop cluster's management and reporting UIs available to the user. Cloudbreak users can configure clusters with their credentials for the cloud service provider, the number of worker nodes to include in the cluster as well the hardware requirements for each node, and, if desired, the pricing model. This last point is particularly useful on EC2, where spot pricing allows users to bid for unclaimed compute capacity on Amazon's cloud. Spot pricing can dramatically reduce costs compared to full price on-demand instances. The disadvantage is that instances can be reallocated if a higher bid comes in, resulting in termination of the processes they are running. Hadoop's facility for automatic redundancy of data and tasks can mitigate this risk.

- **Transfer data into cloud compute clusters.** Typically when using cloud compute services it is fastest to store large data sets in a cloud storage service such as Amazon's Simple Storage Service (S3). From there it is fast to transfer them into and out of cloud compute services like EC2. However, moving data into S3 can be time-consuming depending on the network connections and number and size of the files in the data set. Cloudbreak includes code which communicates with S3 to do a multi-threaded upload of large data files, enabling much faster transfer times.

- **Destroy clusters when processing is complete.** Cloudbreak, by using the Whirr

API, can destroy allocated cloud clusters when processing is complete, ensuring that compute costs can be managed efficiently.

Cloudbreak's user manual contains detailed instructions and examples describing how to leverage cloud computing. We hope that by making cloud computing readily accessible through Cloudbreak's command line interface, more researchers will have the opportunity to leverage Hadoop's distributed computing model, even if they do not have local Hadoop clusters available at their institutions.

5.7 Discussion

The strategy of fitting a GMM to the distribution of insert sizes at a given genomic location has been used by the SV detection tools MoDIL [102] and SVM^2 [39]. SVM^2 fits a mixture of distributions only to candidate regions of the genome identified through a preliminary analysis for the purpose of genotyping variants as homozygous or heterozygous. This leaves MoDIL as the only tool that attempts to model the distribution of insert sizes across the entire genome. As we will see in the next chapter, MoDIL is prone to excessively long run times that make it impractical to run for large-scale genomics data sets. Cloudbreak, on the other hand, through its use of MapReduce and Hadoop, is able to efficiently distribute computation so that given a sufficiently large cluster it can deliver the benefits of this strategy with very fast runtimes.

For those researchers that wish to take advantage of cloud computing in order to avoid the expense and maintenance costs of running their own large Hadoop clusters, Cloudbreak is able to automatically provision, transfer data to, and destroy Hadoop clusters using IaaS providers. Although there are other cloud-enabled sequencing analysis tools, Cloudbreak is unique in using the Apache Whirr library to provide an IaaS provider-agnostic solution. In addition to the obvious benefit of avoiding vendor lock-in, we believe that this will become increasingly important in the future as research agencies and clinical providers begin to create private or semi-private clouds to manage the analysis of sensitive personal genomic data in a controlled setting.

Chapter 6

Evaluating Cloudbreak

In this chapter we evaluate Cloudbreak and compare its accuracy, runtime, and additional features to a variety of other popular tools. We begin by defining some of the methods and parameters of our tests, and then describe experiments we have carried out using simulated data and two real data sets. Finally, we explore the runtime characteristics of the various tools and the extent to which they can be parallelized.

6.1 Evaluation Methods

6.1.1 Choice of SV Detection Tools to Compare To

We compared the performance of Cloudbreak for detecting deletions and insertions to a selection of popular tools: BreakDancer [37], GASVPro [174], Pindel [200], and DELLY [157]. BreakDancer and Pindel are two of the most highly cited SV detection tools, representing classic RP-based methods and SR-based methods, respectively. GASVPro is a hybrid RP method that integrates RD signals and ambiguous mappings into an RP framework based on discordant pairs. DELLY is a recent hybrid RP-SR method that uses split-read mapping to refine candidate calls made with RP information. DELLY produces two sets of calls, one based solely on RP signals, and the other based on RP calls that could be supported by SR evidence; we refer to these sets of calls as DELLY-RP and DELLY-SR. We also attempted to evaluate MoDIL on the same data given that it is the most algorithmically similar method to Cloudbreak. All of these methods detect deletions. Insertions can be detected by BreakDancer, Pindel, and MoDIL.

6.1.2 Simulated and Biological Data Sets

As has been observed elsewhere, there is no available test set of real Illumina sequencing data from a sample that has a complete annotation of SVs from the reference. Therefore, testing with simulated data is important to fully characterize an algorithm's performance characteristics. On the other hand, it is important that the simulated data contain realistic SVs that follow patterns of SVs observed in real data. To address this, we took one of the most complete lists of SVs from a single sample available, the list of homozygous insertions and deletions from the genome of J. Craig Venter [105]. Using these variants, we simulated a 30X read coverage data set for a diploid human Chromosome 2 with a mix of homozygous and heterozygous variants. Since there are relatively few heterozygous insertions and deletions annotated in the Venter genome, we used the set of homozygous indels contained in the HuRef data (`HuRef.homozygous_indels.061109.gff`) and randomly assigned each variant to be either homozygous or heterozygous. Based on this genotype, we applied each variant to one or both of two copies of the human GRCh36 chromosome 2 reference sequence. We then simulated paired Illumina reads from these modified references using *dwgsim* from the DNAA software package [70]. We simulated 100bp reads with a mean fragment size of 300bp and a standard deviation of 30bp, and generated 15X coverage for each modified sequence. Pooling the reads from both simulations gives 30X coverage for a diploid sample with a mix of homozygous and heterozygous insertions and deletions.

We downloaded a data set of reads taken from a DNA sample of Yoruban individual NA18507, experiment ERX009609 from the Sequence Read Archive. This sample was sequenced on the Illumina Genome Analyzer II platform with 100bp paired end reads and a mean fragment size (minus adapters) of 300bp, with a standard deviation of 15bp, to a depth of approximately 37X coverage.

To create a gold standard set of insertions and deletions to test against, we pooled annotated variants discovered by three previous studies on the same sample. These included data from the Human Genome Structural Variation Project reported by Kidd et al. [82], a survey of small indels conducted by Mills et al. [127], and insertions and deletions from the merged call set of the Phase 1 release of the 1000 Genomes Project [1] which were genotyped as present in NA18507. We merged any overlapping calls of the same type into the region spanned by their unions. It should be noted that the 1000 Genomes call set was partially produced using DELLY and BreakDancer, and therefore those calls are ones that those tools are sensitive to, biasing this test in their favor.

6.1.3 Parameters Used for Alignment and SV Detection

We aligned simulated reads to hg18 chromosome 2, NA18507 reads to the hg19 assembly. Alignments for all programs, unless otherwise noted, were found using BWA `aln` version 0.6.2-r126, with parameter `-e 5` to allow for longer gaps in alignments due to the number of small indels near the ends of larger indels in the Venter data set. We also tested the effect of including multiple possible mapping locations for ambiguously mapped reads in results reported in Section 6.7. For those tests, we used two different sets of reads with multiple mapping locations reported. The first used alignments generated with BWA in paired-end mode, reporting up to 25 additional hits for each mapping using the `-n` and `-N` parameters for `bwa sampe` and the script `xa2multi.pl`. For the second, we attempted to generate an exhaustive list of possible mapping locations by running the GEM aligner in single-ended mode on each read in each pair individually, reporting up to 1000 additional hits per alignment. GEM was executed in parallel using Hadoop tasks which wrap GEM version 1.362 (beta), with parameters `-e 6 -m 6 -s 2 -q ignore -d 1000 -max-big-indel-length 0`. These parameters request all hits for a read that are within an edit distance of 6 of the reference, within 2 strata of the best hit, with a maximum of 1000 possible alignments reported for each read. GASVPro also accepts ambiguous mappings but expects them to be realigned with a more sensitive alignment tool; following the details given in their manuscript we extracted read pairs that did not align concordantly with BWA and re-aligned them with Novoalign V2.08.01, with parameters `-a -r -Ex 1100 -t 250`.

We ran BreakDancer version 1.1_2011_02_21 in single threaded mode by first executing `bam2cfg.pl` and then running `breakdancer_max` with the default parameter values. To run BreakDancer in parallel mode we first ran `bam2cfg.pl` and then launched parallel instances of `breakdancer_max` for each chromosome using the `-o` parameter. We ran DELLY version 0.0.9 with the `-p` parameter and default values for other parameters. For the parallel run of DELLY we first split the original BAM file with BamTools [18], and then ran instances of DELLY in parallel for each BAM file. We ran GASVPro version 1.2 using the `GASVPro.sh` script and default parameters. Pindel 0.2.4t was executed with default parameters in single CPU mode, and executed in parallel mode for each chromosome using the `-c` option. We executed MoDIL with default parameters except for a `MAX_DEL_SIZE` of 25000, and processed it in parallel on our cluster with a step size of 121475. To execute other SV detection tools in parallel we wrote simple scripts to submit jobs to the cluster using the HTCondor scheduling engine [181] with directed acyclic graphs to describe dependencies between jobs.

6.1.4 SV Prediction Evaluation

Due to the inability of most tools to determine the exact breakpoint coordinates of an SV given read pair data, as well as the potential for uncertaintly due to error in real gold standard data sets, it is necessary to define a rule for determining whether or not a predicted SV call is correct or not. There are many ways of doing so, each with their own characteristics. For example, the authors of BreakDancer [37] and VariationHunter [72] considered a predicted deletion to be correct if had a 50% reciprocal overlap with a deletion in the test set. For GASVPro [174], Sindi et al. defined a "double uncertainty" metric, in which tolerance parameters ϵ and δ could be defined for the predictions and for the test set respectively, and a deletion prediction of the interval $[x, y]$ is deemed to be correct if there exists an interval in the test set $[a, b]$ for which there is overlap between both pairs of intervals $\langle[x-\epsilon, x+\epsilon],[a-\delta, a+\delta]\rangle$ and $\langle[y-\epsilon, y+\epsilon],[b-\delta, b+\delta]\rangle$. The forthcoming SMASH [180] benchmark of variant callers, including SV detection tools, counted a deletion call as a match if the left breakpoint of the call was within 100bp of the left breakpoint of the deletion interval in the test set and the difference between lengths of the two intervals was less than or equal to 100bp. The authors of the iSVP pipeline [129], meanwhile, assigned a quality score between 0 and 1 based on the ratio of the length of the intersection of the predicted and test variants, with 50bp margins added to each, to the length of the union of the two intervals.

To evaluate Cloudbreak we decided to use looser standards for calling a prediction true, because we would like to test the maximum potential sensitivity of the algorithmic approach even if the resolution of the breakpoints is limited. We feel that this is appropriate given that most calls from SV detection tools will likely be validated either through an additional computational step such as local assembly, or through wet lab techniques such as PCR. Therefore, we use the following criteria to define a true prediction given a gold standard set of deletion and insertion variants to test against: A predicted deletion is counted as a true positive if a) it overlaps with a deletion from the gold standard set, b) the length of the predicted call is within 300bp (the library fragment size in both our real and simulated libraries) of the length of the true deletion, and c) the true deletion has not been already been discovered by another prediction from the same method. For evaluating insertions, each algorithm produces insertion predictions that define an interval in which the insertion is predicted to have occurred with start and end coordinates s and e as well as the predicted length of the insertion, l. The true insertions are defined in terms of their actual insertion coordinate i and their actual length l_a. Given this information, we modify the overlap criteria in a) to include overlaps of the intervals $\langle s, \max(e, s+l) \rangle$ and $\langle i, i+l_a \rangle$. In this study we are interested in detecting events larger than 40bp, because with longer reads, smaller events can be more easily discovered by examining gaps in individual reads. Both Pindel and MoDIL make many calls with a predicted event size of under 40bp, so we remove those calls from the output sets of those programs. Finally, we exclude from consideration calls from all approaches that match a true deletion of less than 40bp where the predicted variant length is less than or equal to 75bp in length.

6.2 Results on Simulated Data

6.2.1 Accuracy and Runtime

Figure 6.1 shows Receiver Operating Characteristics (ROC) curves of the performance of each algorithm for detecting deletions and insertions on the simulated data set. All approaches show excellent specificity at high thresholds in this simulation. Cloudbreak provides the greatest specificity for deletions at higher levels of sensitivity, followed by DELLY. For insertions, Cloudbreak clearly provides the best combinations of sensitivity

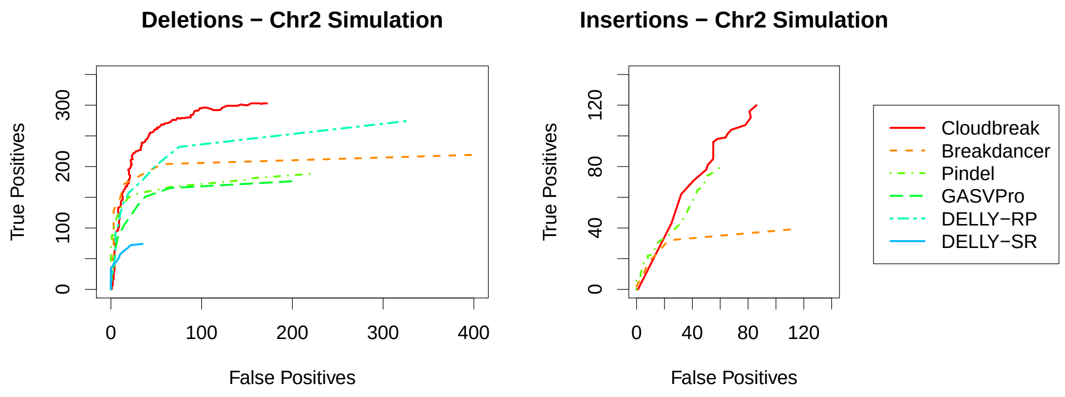

Figure 6.1: Cloudbreak accuracy on a simulated data set. Receiver Operating Characteristic (ROC) curves showing the specificity and sensitivity of each tool to deletions and insertions larger than 40bp on a simulated set of reads giving diploid coverage of 30X on human chromosome 2. Deletions and insertions from the Venter genome were randomly added to one or both haplotypes. Each point on a curve represents a different threshold on the confidence of predictions made by that tool. Thresholds vary by: Cloudbreak - likelihood ratio; BreakDancer, DELLY, GASVPro - number of supporting read pairs; Pindel - simple score.

and specificity. Figure 6.2 shows the runtimes for each tool on the simulated data set, parallelized when possible. Runtimes exclude alignment, which should be similar for all tools. Cloudbreak's runtime is half that of BreakDancer, the next fastest tool, processing the simulated data in under six minutes. Of course, Cloudbreak uses many more CPUs as a distributed algorithm. See Section 6.6 for a more detailed discussion of runtimes and the amount of parallelization that was done for each tool. The output which we obtained from MoDIL did not have a threshold that could be varied to correlate with the trade-off between precision and recall and therefore it is not included in ROC curves; in addition, MoDIL ran for 52,547 seconds using 250 CPUs in our cluster, so results are not included in the runtime figure. Apart from the alignment phase, which is embarrassingly parallel, the feature generation job is the most computationally intensive part of the Cloudbreak workflow. Therefore, to test the algorithm's scalability we measured the runtime of that job on Hadoop clusters made up of varying numbers of nodes and observed that linear speedups can be achieved in this portion of the algorithm by adding additional nodes to the cluster until a point of diminishing returns is reached (Figure 6.3).

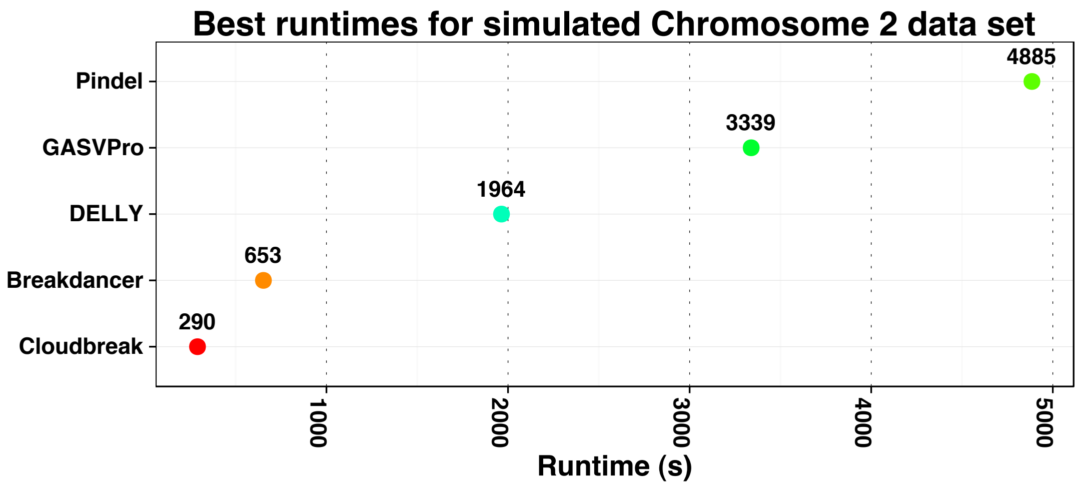

Figure 6.2: Runtimes for each tool on the simulated data set, not including alignment time, parallelized when possible. See Section 6.6 for details on measuring and parallelizing the runtime of each tool.

Choosing the correct operating point or threshold to set on the output of an SV calling algorithm can be difficult when operating on a new data set. The use of simulated data and ROC curves allows for some investigation of the performance characteristics of algorithms at varying levels. First, we characterized the predictions made by each algorithm at the operating point which gives them maximum sensitivity. For Cloudbreak we chose an operating point at which marginal improvements in sensitivity became very low. The results for both deletion and insertion predictions are summarized in Table 6.1. MoDIL and Cloudbreak exhibited the greatest recall for deletions. Cloudbreak has high precision and recall for deletions at this threshold, and discovers many more small deletions. For insertions, Cloudbreak has the highest recall, although recall is low for all four approaches. Cloudbreak again identifies the most small variants. Pindel is the only tool which can consistently identify large insertions, as insertions larger than the library insert size do not produce mapping signatures detectable by read-pair mapping.

We also used the ROC curves to attempt to characterize the predictions made by each algorithm when a low false discovery rate is required. Table 6.2 shows the total number of simulated deletions found by each tool when choosing a threshold that gives an FDR closest to 10% based on the ROC curve. At this more stringent threshold, Cloudbreak identifies

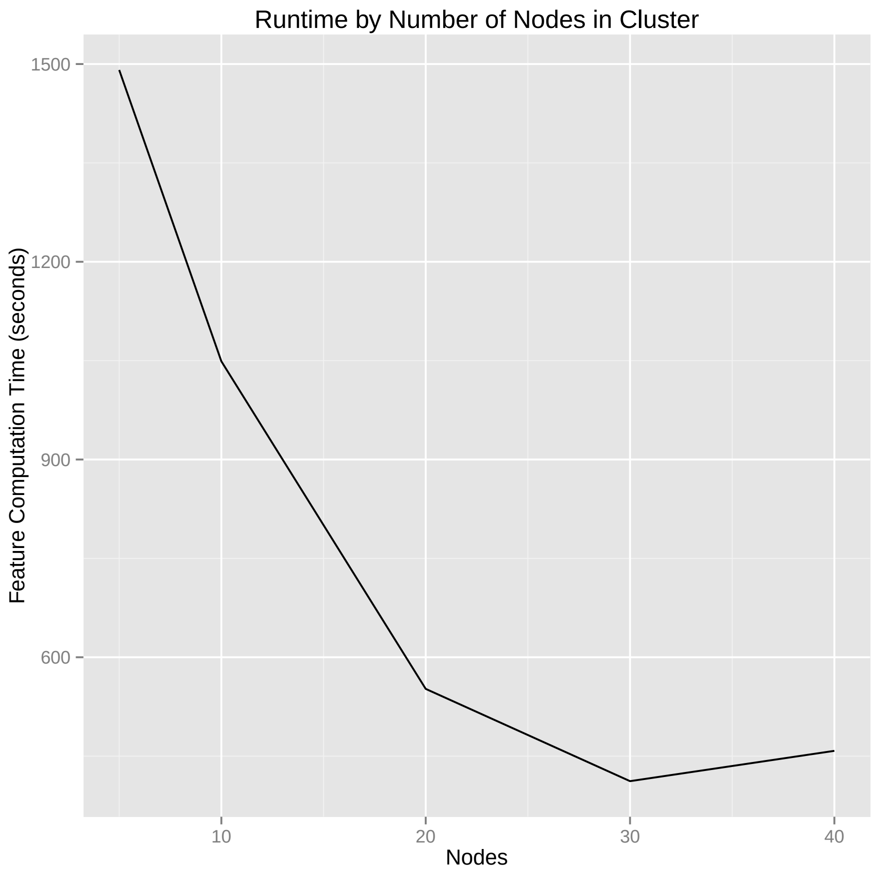

Figure 6.3: Scalability of the Cloudbreak algorithm. Runtime of the Cloudbreak feature generation job for the simulated Chromosome 2 data is shown on Hadoop clusters consisting of varying numbers of compute nodes. Clusters were created in the Amazon Elastic Compute Cloud.

more deletions in every size category than any other tool. Performance on insertions never reached an FDR of 10% for any threshold, so insertion predictions are not included in this table.

6.2.2 Choice of Window Size

A key parameter choice for Cloudbreak is the size of the fixed-width, non-overlapping windows used for local feature computation. The experiments reported thus far in this chapter all used a window size of 25bp. Using the simulated data set, we evaluated the effect of choosing differing window sizes on runtime, breakpoint resolution, and accuracy. Figure 6.4

		Prec.	Recall	F1	40-100bp	101-250bp	251-500bp	501-1000bp	> 1000bp
	Total Number				224	84	82	31	26
Deletions	Cloudbreak	0.638	**0.678**	**0.657**	**153** (9)	61 (0)	62 (0)	12 (0)	15 (0)
	BreakDancer	0.356	0.49	0.412	89 (0)	54 (0)	53 (0)	8 (0)	15 (0)
	GASVPro	0.146	0.432	0.218	83 (2)	32 (0)	55 (0)	8 (0)	15 (0)
	DELLY-RP	0.457	0.613	0.613	114 (3)	**68** (0)	**66** (0)	9 (1)	17 (0)
	DELLY-SR	**0.679**	0.166	0.266	0 (0)	3 (0)	49 (0)	6 (0)	16 (0)
	Pindel	0.462	0.421	0.44	96 (**11**)	24 (0)	48 (0)	5 (0)	15 (0)
	MoDIL	0.132	0.66	0.22	123 (6)	66 (**3**)	**66** (**11**)	**17** (**7**)	**23** (**8**)
Insertions	Total Number				199	83	79	21	21
	Cloudbreak	0.451	**0.305**	**0.364**	**79** (**32**)	**32** (**18**)	**11** (8)	1 (0)	0 (0)
	BreakDancer	0.262	0.0968	0.141	23 (5)	14 (5)	2 (1)	0 (0)	0 (0)
	Pindel	**0.572**	0.196	0.292	52 (25)	5 (1)	10 (**9**)	**3** (**2**)	**9** (**9**)
	MoDIL	0.186	0.0521	0.0814	14 (1)	4 (0)	1 (0)	2 (**2**)	0 (0)

Table 6.1: The number of simulated deletions and insertions in the 30X diploid chromosome 2 with Venter indels found by each tool at maximum sensitivity, as well as the number of those variants that were discovered exclusively by each tool (in parentheses). The total number of variants in each size class in the true set of deletions and insertions is shown in the first row of each section.

	40-100bp	101-250bp	251-500bp	501-1000bp	> 1000bp
Total Number	224	84	82	31	26
Cloudbreak	**68** (17)	**67** (**10**)	**56** (**5**)	**11** (**3**)	**15** (**0**)
BreakDancer	52 (8)	49 (2)	49 (0)	7 (0)	14 (**0**)
GASVPro	35 (2)	26 (0)	26 (0)	2 (0)	6 (**0**)
DELLY-RP	22 (1)	56 (1)	40 (0)	8 (0)	12 (**0**)
DELLY-SR	0 (0)	2 (0)	28 (0)	2 (0)	10 (**0**)
Pindel	60 (**32**)	16 (0)	41 (2)	1 (0)	12 (**0**)

Table 6.2: The number of simulated deletions in the 30X diploid chromosome 2 with Venter indels found by each tool at a 10% FDR, as well as the number of those deletions that were discovered exclusively by each tool (in parentheses). The total number of deletions in each size class in the true set of deletions is shown in the second row of the header.

shows the runtime of the feature computation and variant calling steps of Cloudbreak on the Chromosome 2 data set using differing window sizes. Runtime decreases dramatically with larger window sizes. For very small window sizes, the runtime is dominated by the variant extraction job, which is only parallelized by chromosome and therefore only runs on one core when running on the simulated data set. Table 6.3 shows several accuracy measures for Cloudbreak running with varying window sizes, using the same threshold for each run. The greatest recall is achieved at very low window sizes, probably because

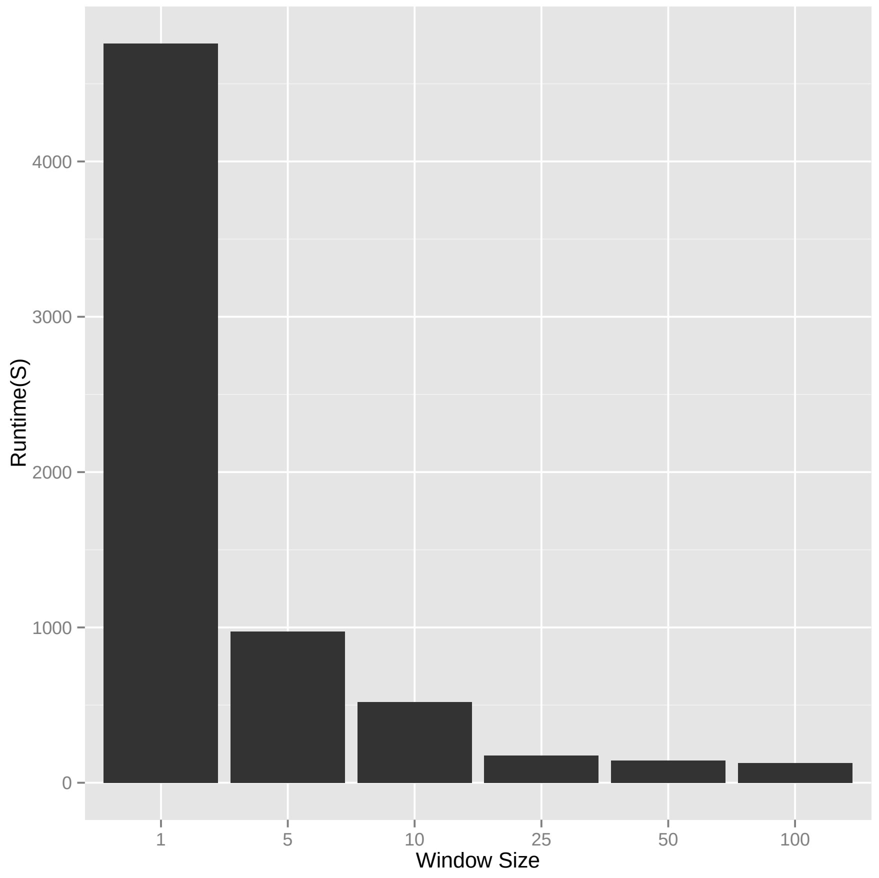

Figure 6.4: Runtimes for Cloudbreak on the Chromosome 2 simulated data set using differing choices of window size. Runtimes include the feature computation and variant calling Cloudbreak Hadoop jobs.

evidence for smaller variants is less likely to be mixed in with non-variant supporting read pairs from the flanking regions. However, overall accuracy is high until window sizes reach 50bp, at which point recall decreases dramatically. Finally, we examined the effect of breakpoint accuracy on window size and found that it had little effect (Figure 6.5).

6.3 Results on Biological Data

6.3.1 Accuracy and Runtime

Figure 6.6 shows the performance of each algorithm on the NA18507 data set when compared against the gold standard set for both deletions and insertions. We were unable

Window Size	Calls	True Positives	Precision	Recall	F1
1	274	228	0.832	0.57	0.677
5	268	228	0.851	0.57	0.683
10	258	223	0.864	0.557	0.678
25	240	217	0.904	0.542	0.678
50	215	194	0.902	0.485	0.631
100	162	141	0.87	0.352	0.502

Table 6.3: Accuracy measures for Cloudbreak on the Chromosome 2 simulated data set using different choices of window size. For each window size the same Cloudbreak score threshold was used (1.98).

to run MoDIL on the whole-genome data set due to the estimated runtime and storage requirements. All other algorithms show far less specificity for the gold standard set than they did for the true variants in the single chromosome simulation, although it is difficult to tell how much of the difference is due to the added complexity of real data and a whole genome, and how much is due to missing variants in the gold standard set that are actually present in the sample. For deletions, Cloudbreak is the best performer at the most stringent thresholds, and has the highest or second highest precision at higher sensitivity levels. Cloudbreak has slightly lower accuracy for insertions than the other tools at more stringent thresholds, although it can identify the most variants at higher levels of sensitivity. Figure 6.7 and Table 6.6 show the runtime of each of the tools on the NA18507 dataset. Cloudbreak processes the sample in under 15 minutes on our cluster, more than six times as fast as the next fastest program, BreakDancer, even when BreakDancer is run in parallel for each chromosome on different nodes in the cluster.

Given the high number of false positives produced by all tools at maximum sensitivity indicated by the ROC curves, we decided to characterize the predictions made by each tool at more stringent thresholds. We examined the deletion predictions made by each algorithm using the same cutoffs that yielded a 10% FDR on the simulated chromosome 2 data set, adjusted proportionally for the difference in coverage from 30X to 37X. For insertions, again, we were forced to use the thresholds that gave maximum sensitivity for each tool due to the high observed FDR rates in the simulated data. The precision and recall at these thresholds with respect to the gold standard set, as well as the performance

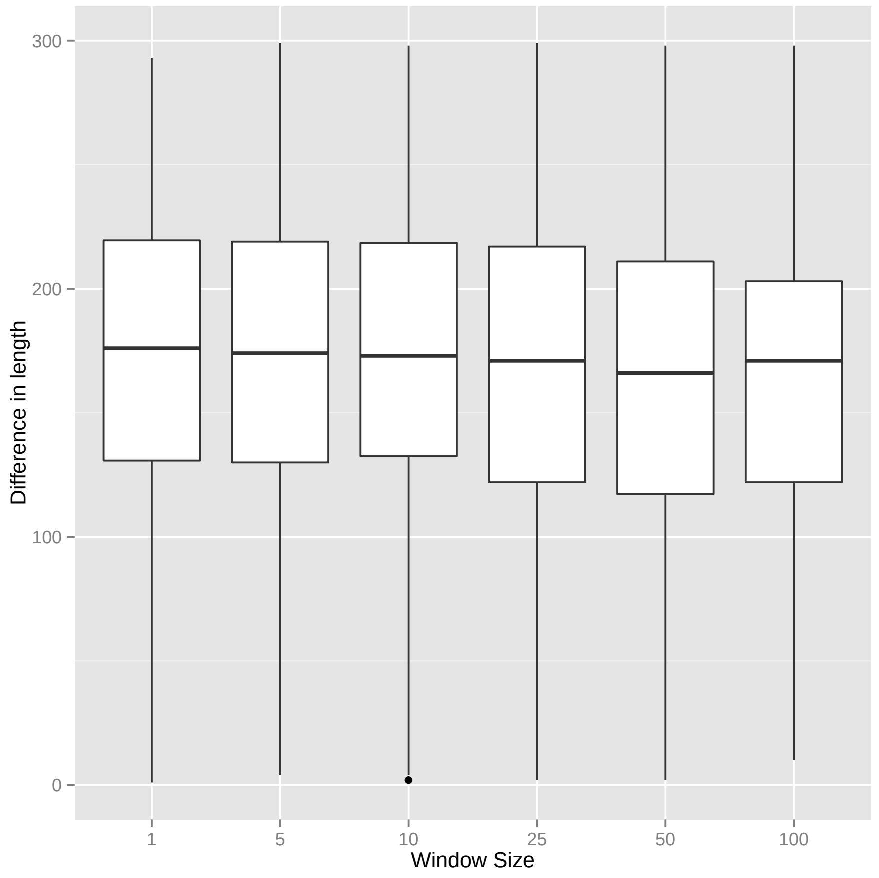

Figure 6.5: Breakpoint resolution for Cloudbreak on the Chromosome 2 simulated data set using differing choices of window size.

of each algorithm at predicting variants of each size class at those thresholds, is shown in Table 6.4. For deletions, Cloudbreak has the greatest sensitivity of any tool at these thresholds, identifying the most variants in each size class. Pindel exhibits the highest precision with respect to the gold standard set. For insertions, Pindel again has the highest precision at maximum sensitivity, although according the ROC curve it is possible to choose a threshold for Cloudbreak with higher precision and recall. At maximum sensitivity, Cloudbreak identifies 120 more insertions from the gold standard set than Pindel, giving it by far the highest recall.

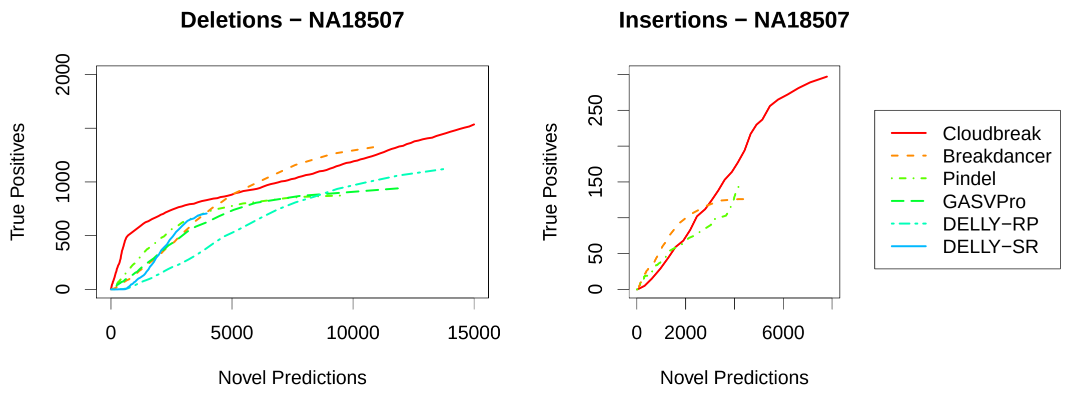

Figure 6.6: Accuracy on the 37X NA18507 sample. ROC curves for deletion and insertion prediction performance, tested against the combined gold standard sets of deletions taken from [82], [127], and [1].

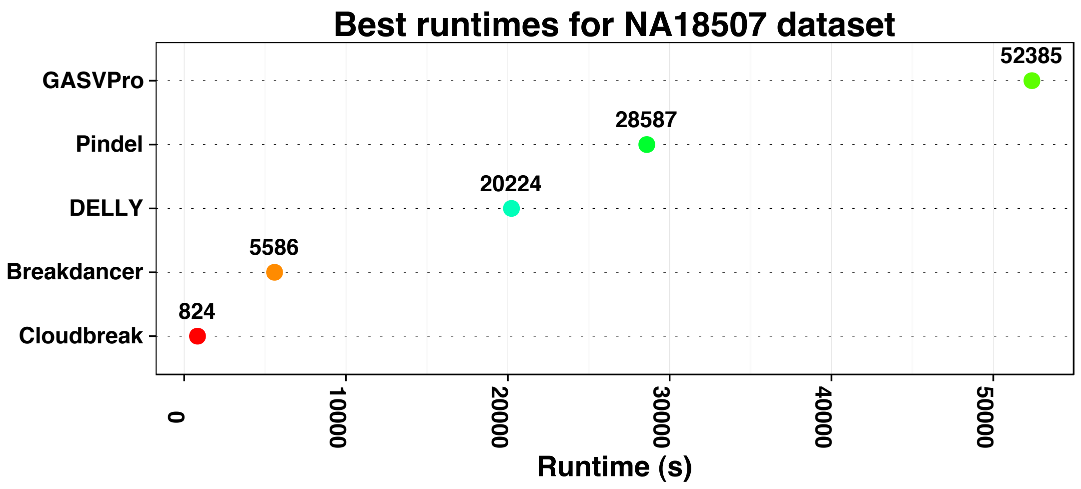

Figure 6.7: Runtimes for each tool on the NA18507 data set, not including alignment time, parallelized when possible. See Section 6.6 for details on measuring and parallelizing the runtime of each tool.

6.3.2 Breakpoint Resolution

We expected that the methods tested here would vary in their breakpoint resolution: SR methods can achieve single nucleotide breakpoint resolution, while RP methods depend on insert size evidence from overlapping fragments that will likely not indicate the exact location of the breakpoint. This distinction can be seen clearly in Figure 6.8. Cloudbreak

		Prec.	Recall	F1	40-100bp	101-250bp	251-500bp	501-1000bp	> 1000bp
	Total Number				7,462	240	232	147	540
Deletions	Cloudbreak	0.0943	**0.17**	0.121	**573** (**277**)	**176** (**30**)	**197** (**18**)	**121** (**6**)	**399** (**24**)
	BreakDancer	0.137	0.123	**0.13**	261 (29)	136 (3)	178 (0)	114 (0)	371 (0)
	GASVPro	0.147	0.0474	0.0717	120 (21)	40 (2)	85 (0)	36 (0)	128 (0)
	DELLY-RP	0.0931	0.1	0.0965	143 (6)	128 (3)	167 (1)	103 (0)	323 (1)
	DELLY-SR	0.153	0.0485	0.0736	0 (0)	26 (0)	123 (0)	66 (0)	203 (0)
	Pindel	**0.179**	0.0748	0.106	149 (8)	61 (0)	149 (0)	69 (1)	217 (0)
Insertions	Total Number				536	114	45	1	0
	Cloudbreak	0.0323	**0.455**	0.0604	**265** (**104**)	**49** (**24**)	3 (1)	0 (0)	0 (0)
	BreakDancer	0.0281	0.181	0.0487	97 (10)	27 (5)	2 (1)	0 (0)	0 (0)
	Pindel	**0.0387**	0.239	**0.0666**	144 (45)	14 (7)	**7** (**6**)	**1** (**1**)	0 (0)

Table 6.4: The precision and recall with respect to the gold standard set of deletions and insertions for each tool on the NA18507 data, as well as the number of variants found in each size class found. Exclusive predictions are in parentheses. For deletions, the same cutoffs were used as for the simulated data as in Table 6.2, adjusted for the difference in coverage from 30X to 37X. For insertions, the maximum sensitivity cutoff was used.

has the lowest resolution of any of the RP tools tested, with GASVPro also having relatively poor resolution. Pindel and DELLY-SR, meanwhile, use split read mappings to pinpoint the exact breakpoint correctly in many cases. Cloudbreak's resolution suffers because the independent calculation of features from the distribution of overlapping insert sizes does not allow the algorithm to keep track of the actual coordinates of the mapped reads from each pair, which other tools use to set bounds on the true breakpoint locations. Cloudbreak's goal, however, is to increase sensitivity and specificity to actual variants in the hopes that such calls could still be useful even if their resolution is limited. For example, it is possible to build SV detection pipelines in which a set of RP-based candidate calls is further validated *in silico* in a post-processing step, and such approaches typically also refine breakpoint resolution. *In silico* validation can be accomplished through a variety of methods; in one example, a recently published approach validated RP-based calls by performing a *de novo* assembly of the reads surrounding candidate calls that had been created by an RP-based approach [116]. Another approach to call refinement can be seen in studies that improve the breakpoint resolution of RP calls to single-nucleotide resolution using SR-based validation [92]. In Chapter 7, we will explore an alternative approach to *in silico* validation of SV calls based on a discriminative machine learning technique, and show that we can use it to greatly improve the breakpoint resolution of Cloudbreak's calls.

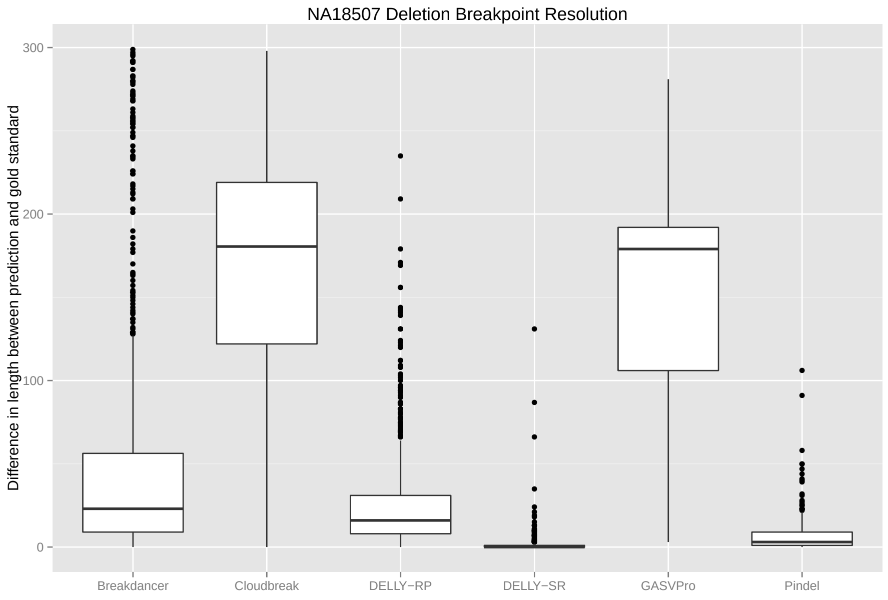

Figure 6.8: Breakpoint resolution for each tool for deletions from the gold standard set on the NA18507 data. For each correctly predicted deletion, we calculated the difference in length between the true deletion and the prediction.

6.4 Results on a Low-Coverage Cancer Data Set

We also tested Cloudbreak on the sequencing data set obtained from a patient with acute myeloid leukemia (AML) described in Section 3.7. This data set consisted of 76bp paired end reads with a mean insert size of 285bp and standard deviation of 50bp, yielding sequence coverage of 5X and physical coverage of 8X. Using a pipeline consisting of Novoalign, BreakDancer, and a set of custom scripts for filtering and annotating candidate SVs, we had previously identified a set of variants present in this sample and validated several using PCR, including 8 deletions. Cloudbreak was able to identify all 8 of the validated deletions, showing that it is still sensitive to variants even when using lower coverage data sets with a greater variance of insert sizes.

		Actual Genotypes			
		Simulated Data		NA18507	
		Homozygous	Heterozygous	Homozygous	Heterozygous
Predicted Genotypes	Homozygous	35	2	96	21
	Heterozygous	0	39	2	448

Table 6.5: Confusion matrices for the predicted genotype of deletions found by Cloudbreak on both the simulated and NA18507 data sets.

6.5 Genotyping Variants

Because Cloudbreak explicitly models zygosity in its feature generation algorithm, it can predict the genotypes of identified variants. We tested this on both the simulated and NA18507 data sets. For the NA18507 data set, we considered the deletions from the 1000 Genomes Project, which had been genotyped using the population-scale SV detection algorithm Genome STRiP [65]. Cloudbreak was able to achieve 92.7% and 95.9% accuracy in predicting the genotype of the deletions it detected at our 10% FDR threshold in the simulated and real data sets, respectively. Table 6.5 shows confusion matrices for the two samples using this classifier. None of the three input sets that made up the gold standard for NA18507 contained a sufficient number of insertions that met our size threshold and also had genotyping information. Of the 123 insertions detected by Cloudbreak on the simulated data set, 43 were heterozygous. Cloudbreak correctly classified 78 of the 80 homozygous insertions and 31 of the 43 heterozygous insertions, for an overall accuracy of 88.6%.

6.6 Notes on Evaluating Runtime

We implemented and executed Cloudbreak on a 56-node Hadoop cluster, with 636 map slots and 477 reduce slots. Not including alignment time, we were able to process the Chromosome 2 simulated data in under six minutes, and the the NA18507 data set in under 15 minutes. For the simulated data set we used 100 reducers for the compute SV features job; for the real data set we used 300. The bulk of Cloudbreak's execution is spent in the feature generation step. Extracting deletion and insertion calls take under two minutes each for both the real and simulated data sets; the times are equal because each reducer is responsible for processing a single chromosome, and so the runtime is bounded by the length of time it takes to process the largest chromosome.

Cloudbreak's elapsed times are faster than all of the other tools tested; however, there are several ways in which to compare runtime performance between tools that support different levels of parallelization. In Table 6.6 we display a comparison of runtimes on the real and simulated data sets for all of the tools evaluated in this work. We report runtimes for each tools run in its default single-threaded mode, as well as for levels of parallelization achievable with basic scripting, noting that one of the key advantages of Hadoop/MapReduce is the ability to scale parallel execution to the size of the available compute cluster without any custom programming. Pindel allows multi-threaded operation on multicore servers. Pindel and BreakDancer allow processing of a single chromosome in one process, so it is possible to execute all chromosomes in parallel on a cluster that has a job scheduler and shared filesystem. BreakDancer has an additional preprocessing step (`bam2cfg.pl`) which runs in a single thread. DELLY suggests splitting the input BAM file by chromosome, after which a separate DELLY process can be executed on the data for each chromosome; splitting a large BAM file is a time consuming process and consumes most of the time in this parallel workflow, in fact making it faster to run in single-threaded mode. GASVPro allows parallelization of the MCMC component for resolving ambiguously mapped read pairs; however, this requires a significant amount of custom scripting, and we did not find that the MCMC module consumed most of the runtime in our experiments, so we do not attempt to parallelize this component. The MoDIL distribution contains a set of scripts that can be used to submit parallel jobs to the SGE scheduling engine or modified for other schedulers; we adapted these for use in our cluster.

In parallel execution, the total time to execute is bounded by the runtime of the longest-running process. In the case of chromosome-parallelizable tools including BreakDancer, Pindel, and DELLY, this is typically the process working on the largest chromosome.[1] In the case of MoDIL's run on the simulated data, we found that the different processes

[1]We note that one BreakDancer process, handling an unplaced contig in the hg19 reference genome, never completed in our runs and had to be killed manually; we exclude that process from our results.

		Simulated Data			NA18507		
	SV Types	Single CPU	Parallel	Proc.	Single CPU	Parallel	Proc.
Cloudbreak	D,I	NA	290	312	NA	824	636
BreakDancer	D,I,V,T	653	NA	NA	134,170	5,586	84
GASVPro	D,V	3,339	NA	NA	52,385	NA	NA
DELLY	D	1,964	NA	NA	30,311	20,224	84
Pindel	D,I,V,P	37,006	4,885	8	284,932	28,587	84
MoDIL	D,I	NA	52,547	250	NA	NA	NA

Table 6.6: Runtimes (elapsed) on both data sets of each tool tested, in single-processor and parallel mode. For parallel runs, Proc. is the maximum number of simultaneously running processes or threads. All times are in seconds. The types of variants detected by each program are listed with the abbreviations: D - deletion; I - insertion; V - Inversion; P - duplication; T - translocation. Interchromosomal translocations are only detected by BreakDancer in single CPU mode.

varied widely in their execution times, likely caused by regions of high coverage or with many ambiguously mapped reads. Cloudbreak mitigates this problem during the time-consuming feature generation process by using Hadoop partitioners to randomly assign each genomic location to one of the set of reducers, ensuring that the work is evenly distributed across all processes. This distribution of processing across the entire cluster also serves to protect against server slowdowns and hardware failures - for example, we were still able to complete processing of the NA18507 data set during a run where one of the compute nodes was rebooted midway through the feature generation job.

6.7 Choice of Aligner and Use of Multiple Mappings

As mentioned in Section 5.4, one goal for developing Cloudbreak was to see if the use of Hadoop would allow productive use of large sets of possible mappings for ambiguously mapped read pairs. Therefore, we also tested the effect of using different aligners and including multiple mappings for ambiguously mapped reads. In addition to the BWA paired end best alignments used in the results reported above, we also used BWA in paired end mode set to report up to 25 possible mapping locations for each ambiguously mapped read pair, as well as the GEM aligner run in single ended mode on each read in each pair and set to report up to 1000 additional mappings per read. Interestingly, we

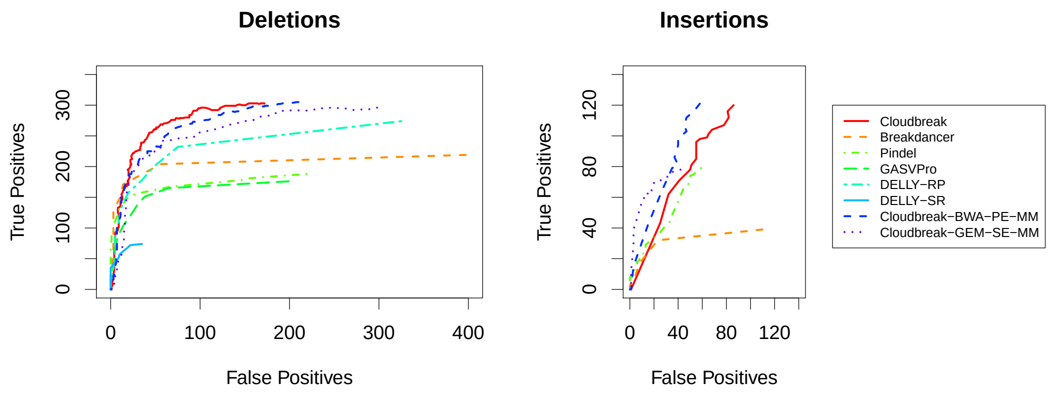

Figure 6.9: Cloudbreak performance on the chromosome 2 simulation using different alignment strategies. ROC curves show the number of true positives and false positives for each operating point for deletions and insertions. The Cloudbreak alignment strategies are: 1) “Cloudbreak”: Alignments generated with BWA in paired-end mode, reporting the best hit for each pair. 2) “Cloudbreak-BWA-PE-MM”: Alignments generated with BWA in paired-end mode, reporting up to 25 additional hits for each mapping. 3) “Cloudbreak-GEM-SE-MM”: Alignments generated by running the GEM aligner in single-ended mode, reporting up to 1000 additional hits per alignment. Considering multiple mappings improves Cloudbreak’s specificity for insertions but decreases sensitivity to deletions.

found that including multiple mappings increases accuracy on insertions, but decreases performance for deletions (Figure 6.9). This seems to indicate that while BWA’s paired end mode is very good at picking correct alignments for pairs that map with long implied insert sizes, it seems to not be as good at finding correct pairings for the shorter distances indicative of insertions. We also examined the ability of all of the approaches, including Cloudbreak using both the BWA unambiguous best alignments and the GEM alignments, to detect events in repetitive regions of the genome, as shown in Table 6.7. We found that all of the tested methods detected a similar proportion of such variants, although we did find that when used with the GEM multiple mappings, Cloudbreak does exclusively find a set of both deletion and insertion variants. In terms of runtime, use of the large set of multiple mappings did not change elapsed time to run the simulated data sample greatly, taking 308 seconds versus the 290 sections reported in the previous section. For the NA18507 dataset, the difference in runtime was more substantial, with Cloudbreak moving from taking 824 seconds to process the best alignments to 2,310 seconds to process

		Simulated Data		NA18507	
		Non-repeat	Repeat	Non-repeat	Repeat
Deletions	Total Number	120	327	562	8059
	Cloudbreak	62 (0)	**241** (3)	**300** (**44**)	**1166** (**150**)
	Cloudbreak-GEM	65 (2)	230 (4)	226 (8)	1001 (17)
	BreakDancer	42 (0)	177 (0)	192 (11)	868 (20)
	GASVPro	37 (1)	156 (1)	79 (6)	330 (16)
	DELLY-RP	57 (1)	217 (2)	152 (4)	712 (3)
	DELLY-SR	3 (0)	71 (0)	27 (0)	391 (0)
	Pindel	26 (4)	162 (7)	109 (2)	536 (6)
	MoDIL	**72** (**12**)	223 (**16**)	NA	NA
Insertions	Total Number	133	270	341	355
	Cloudbreak	**32** (7)	**91** (**30**)	**169** (**41**)	**148** (**43**)
	Cloudbreak-GEM	21 (0)	58 (7)	137 (30)	135 (32)
	Breakdancer	17 (5)	22 (3)	82 (22)	44 (8)
	Pindel	25 (**16**)	54 (26)	84 (13)	82 (15)
	MoDIL	5 (0)	16 (2)	NA	NA

Table 6.7: Detected deletions and insertions on the simulated and NA18507 data sets identified by each tool, broken down by whether the deletion overlaps with a RepeatMasker-annotated element. For all calls the maximum sensitivity cutoff was used.

the multiple mappings data set, a runtime which is still almost 50% less than the next fastest approach, BreakDancer parallelized by chromosome. Although the runtimes were very manageable, we see no substantial benefit in terms of accuracy in using large sets of ambiguous mappings, at least in our current implementation.

6.8 Discussion

We have demonstrated the Cloudbreak can deliver excellent accuracy at identifying regions that contain deletion and insertion SVs while at the same time achieving dramatically faster runtimes than other approaches through the use of the Hadoop framework and its implementation of parallel computing with data locality. In particular, the algorithm we developed for Cloudbreak is better able to identify small variants (50bp-150bp) than other tools. The downside to Cloudbreak's performance on our evaluations is its poor breakpoint resolution. As we mentioned previously, we believe that in most cases, predictions from SV detection tools require further validation, either through wet lab techniques or by further computational validation through techniques such as the split-read refinement offered by tools like DELLY, or through breakpoint assembly techniques such as TIGRA [36]. In the next chapter, we will demonstrate an additional technique that improves Cloudbreak's resolution: a discriminative machine learning framework that can identify breakpoint locations by integrating read pairing, depth of coverage, and split read signals from sequencing data sets.

Chapter 7

Extending Local Feature Based Models of SV Detection in a Discriminative Machine Learning Framework

In the formulation of a general algorithmic framework for SV prediction in MapReduce that we described in Chapter 4, a crucial step is the computation of a set of features for each of the small, non-overlapping windows with which we tiled the genome reference. In the implementation of Cloudbreak described in Chapter 5, these features were the parameters which are estimated by fitting a GMM to the distribution of insert sizes that span that location. However, in general the nature of these features are purposefully not specified in the algorithmic framework to allow flexibility in the nature of the applications that it could potentially be used to build.

Given a set of arbitrary features that encode information about a set of loci that are connected in a sequence, a natural approach is to apply techniques from machine learning to identify regions of interest in that sequence, rather than the heuristics and noise reduction techniques from signal processing that are used in the Cloudbreak implementation described in Chapter 5. In this chapter we will reformulate the problem as a sequence labeling task, discuss possible machine learning frameworks that could be used to solve it, explore the ways in which features can be engineered to integrate multiple signals of SVs, and show that it is possible to achieve modest but positive improvements using conditional random fields, a discriminative machine learning technique.

7.1 Related Work

Discriminative machine learning techniques have been applied to SV detection in the tools forestSV [126] and SVM^2 [39]. SVM^2 computes a set of statistics based on differences between the coverage and insert size distributions observed at nearby locations on the genome. If the statistics pass a set of coarse filtering rules, they are used as features for a support vector machine (SVM) classifier that the authors trained on a set of simulated short insertion and deletions. forestSV trained a Random Forest classifier [23] to detect deletions and duplications based on validated and false positive predictions from many samples in the 1000 Genomes Project data set. Their tool is based upon computing features for a sliding window moving across the genome, as well as for the flanking regions of that window. The authors demonstrated that it was possible to combine disparate signals by including features that were based on read depth as well as discordant read pair mappings in their feature set. The forestSV algorithm was similar to Cloudbreak in that the classifier made calls at each window, and a postprocessing function then merged windows classified with the same label into variant calls. Both of these methods demonstrate innovative ways of learning from existing data, and integrating read pair and split read based signals. However, neither of these tools use machine learning techniques that take into account the sequential nature of the data; the classifiers are run independently on each candidate variant location (in the case of SVM^2) or window (in the case of forestSV), and the labels that are assigned in each invocation of the classifier do not affect the neighboring regions. We will show that by treating the problem as a sequence labeling task, it is possible to take advantage of learning techniques that operate on the entire sequence of observations at once.

7.2 SV Detection as a Sequence Labeling Problem

In our MapReduce algorithmic framework, the features computed at each window are transformed into variant calls by a function that examines the features along each chromosome of the reference sequentially and identifies contiguous blocks of regions with features values that combine to coherently indicate the presence of a variant. This part of the process, which we named the POSTPROCESS function in Algorithm 1, can be thought of as a sequence labeling problem based on a series of observation features. To take the example of deletion detection, we could define two labels: "Deletion" if the window participates in a deletion variant, and "No Deletion" if it does not. More formally, we can let Y_i be the label at genomic window i, where $1 \leq i \leq n$ and n is the number of windows in the chromosome. If we similarly name the features for genomic window i X_i, we can then refer to the entire sequence of labels and features as Y and X, respectively.

The goal of the POSTPROCESS function can be broken down into two parts: first, to assign a label to each window in the reference sequence, and second, to consolidate neighboring windows with the same label into variant calls which affect larger regions. Analyzing the implementation of Cloudbreak described in Chapter 5 in this model, we used a simple linear threshold on the likelihood ratio of the insert size data to label each window, and then consolidated neighboring windows with hand-tuned rules such as the median filter and restriction on the estimated mean of the second GMM component. These latter rules were made necessary by the noisy nature of the data and the simplicity of the window-labeling procedure. However, if we had a completely accurate window-labeling procedure, we could remove much of the complexity of the consolidation step. Since real world data is noisy the best that we can do is to try to find the most likely sequence of labels given the observed data:

$$\arg\max_{Y} P(Y|X)$$

7.3 Graphical Models for Sequence Labeling

To achieve the goal of a more principled and more accurate window-labeling procedure we would need to be able to take into account information from multiple features, as well as the labels of nearby windows. In other words, the label of each window should be dependent not only on the observed features at that window but also on the labels of other windows nearby. Probabilistic graphical models provide a useful framework for defining and learning models that describe these types of dependencies. One class of graphical

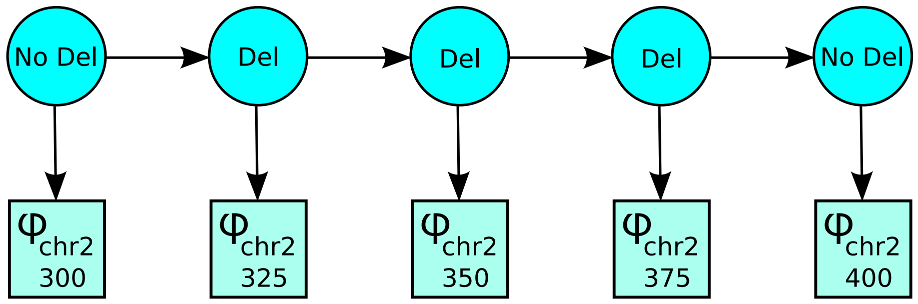

Figure 7.1: A Hidden Markov Model for the SV detection problem.

model that has been used extensively for sequence labeling tasks in bioinformatics are Hidden Markov Models (HMMs). HMMs model sequence labeling problems by assuming that the observations at each point are dependent on a hidden state, and that the state at a given time is dependent on the state at the previous time step. If one creates an HMM such that the hidden states correspond to the labels of interest, the task of learning an HMM is equivalent to learning a probability distribution $P(Y_i|Y_{i-1})$ that describes the likelihood of moving from one state to another, and a distribution $P(X_i|Y_i)$ that gives the likelihood of seeing a particular observation given the true label at that time point, which can be modeled by a directed graphical network (Figure 7.1). Therefore, HMMs are generative models that model the entire distribution over labels and observations, or the joint distribution $P(Y, X)$. One published CNV detection algorithm, Zinfandel [171], used an HMM to predict the presence of deletions and duplications. That algorithm used a feature set consisting of the read depth at each location and likelihood of the distribution of insert sizes given varying fixed potential sizes of deletions.

One downside to modeling the joint distribution over features and labels is that what we are really interested is the conditional distribution $P(Y|X)$: given the observations, what is the most likely sequence of labels? Since $P(Y, X) = P(Y|X) * P(X)$, by training an HMM we are forced to learn $P(X)$, or the distribution over sequences of observations, even though it does not help us achieve our goal. A second drawback to HMMs, particularly in the proposed application of SV detection, is the fact that the model assumes that $P(X_i)$ is independent of $P(X_{i-1})$ given Y_i. In other words, observations drawn from the same state should all be independent. For the SV detection problem described here, this seems highly restrictive: for many features we might imagine, observations from two different genomic windows that are participating in the same deletion variant are likely to be much more similar than two windows that are participating in different deletion variants, even though they all share the label "Deletion". As the developers of Zinfandel noted, this required them to create a grid of "Deletion" and "Deletion Flank" states, with each row corresponding to a different size class of deletion variant.

Conditional random fields [88] (CRFs) offer solutions to each of these drawbacks. In contrast to HMMs, CRFs are Markov Random Fields which can be represented by undirected graphical models. Because the model is undirected, it is possible to directly learn the conditional distribution $P(Y|X)$ from a set of training data. Training and inference of these models is tractable for simplified graphical structures; for our application we will use a *linear chain CRF*, which mimics the structure of a sequence of observations and its labels (Figure 7.2). Furthermore, CRFs represent the factors that relate the observation sequence to the label node at position i in the sequence in the graph through a log-linear model over a set of k feature functions:

$$\exp\left(\sum_k \theta_k f_k(y_i, y_{i-1}, X)\right)$$

This means that the feature functions for each position i in the sequence can incorporate information from observations any point in the observation sequence. This removes the requirement that observations be independent from one another, and allows the consideration of features taken from neighboring observations, something that could be useful in our context of SV detection, where multiple small genomic windows are spanned by single reads and fragments.

7.4 Integrating Features with Conditional Random Fields

We implemented a linear-chain conditional random field model to label genomic insertions and deletions. The model was developed using Factorie [121]. Factorie is a toolkit written in Scala that supports a wide variety of factor-graph based models, trainers,

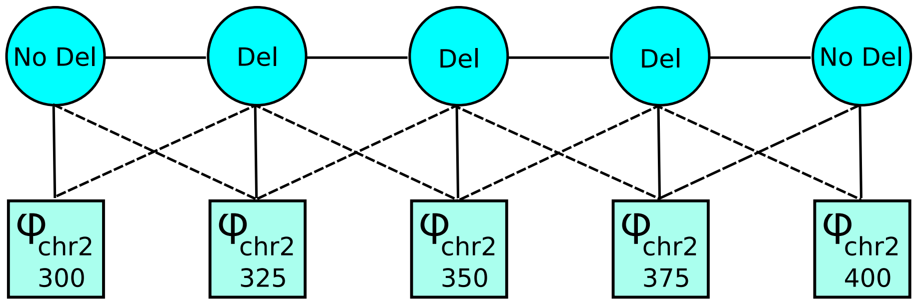

Figure 7.2: A linear chain Conditional Random Field model of the SV detection problem. Label nodes can be connected by feature functions to any of the observations in the sequence.

and tools, particularly for NLP tasks such as named entity recognition. Our Factorie-based implementation of a structural variation detection program consists of two parts: a modular and configurable set of code for data management of features defined along the genome, along with conversion of real-valued features into into binary binned feature values, and code to construct, train, and run inference on linear-chain CRF models. The code developed for this project was written in Scala and is available on GitHub at `http://github.com/cwhelan/svfactorie`.

7.5 Features for SV Detection

As described in Chapter 2, there are three main signals available for SV detection in short read sequencing data sets: those that come from read pairing information, those that come from read depth, and those that come from split reads. The creation of a discriminative machine learning framework as described above can be used with any arbitrary set of features. Therefore it is possible to create feature sets that combine information from all three SV detection signals and integrate them into this framework. In addition, we can model interactions between features and use those in our predictions. Finally, the arbitrary nature of the feature function allows us to incorporate prior knowledge about given genomic regions, including sequence annotations. We have constructed a feature set that includes all of these types of features, as described below:

7.5.1 Read Pair Features

Read pair features are those that are based upon the inferred insert sizes and orientations linking paired reads, as described previously. We use the following RP features:

- The three features generated by Cloudbreak and described in Chapter 5: the log likelihood ratio of the insert sizes observed at each window in the two-component GMM fit vs. the likelihood under the expected normal distribution for the sample; the estimated mean μ_2 of the second component of the two-component GMM, and α, the estimated weight of the second component in the two-component GMM.

- Insert size change point features: in addition to the insert size calculations described in the Cloudbreak implementation, we also wanted to consider alternative features based on insert sizes. In particular, we wanted to test whether the addition of features that indicate whether a window represents a point at which the insert size distribution in a local surrounding region is changing would improve identification of variants. We compute such a score as follows: let i be the index of a window in the genome, and $S_{i..j}$ be the list of observed insert sizes spanning the windows labeled i through j. Now consider a local neighborhood of size n which comprises the windows $i - \frac{n}{2}, ..., i, ..., i + \frac{n}{2}$. Let $\Theta_{i..j}$ be model parameters that can be estimated from the distribution of insert sizes $S_{i..j}$, and $P(S_{i..j}|\Theta_{i..j})$ be the likelihood of observing $S_{i..j}$ under the model with parameters Θ. We can calculate a change score for each window i by examining the likelihood of the data in the halves of the neighborhood to the right and left of that window, by estimating $P(S_{i-\frac{n}{2}..i})$ and $P(S_{i+1..i+\frac{n}{2}})$. Simultaneously, we can perform the same estimates for all of the data in the neighborhood and compute $P(S_{i-\frac{n}{2}..i+\frac{n}{2}})$. The likelihood ratio between these two models:

$$\frac{P(S_{i-\frac{n}{2}..i}) * P(S_{i+1..i+\frac{n}{2}})}{P(S_{i-\frac{n}{2}..i+\frac{n}{2}})}$$

  then serves as a score indicating the presence of a change between the segments to the left and right of point i. This has the advantage of being a parameter-free calculation, since parameters are re-estimated from each segment for each calculation. According to the Cloudbreak's model, it would be natural to estimate two component GMMs for each segment of insert size data. However, in practice we found that changes were well captured by the likelihood ratio score when only a single-component Gaussian is used for each segment, and so we have used that model for the sake of efficiency. We use a neighborhood size of 200 bp.

7.5.2 Split-read features

Split read signals are caused when a read overlaps a genome breakpoint, disrupting the full alignment of that read. If only simple variants are considered, such a scenario should reveal the exact location of the breakpoint. However, these are difficult to resolve in practice, because the algorithm must first identify potentially split reads from those that fail to align or partially align, and then align the two ends of the split read to the genome, potentially at large distances from one another. True split-read based SV detection methods use dynamic programming and heuristics to try to accomplish this; however, the difficulty of doing so with current read lengths means that they have low sensitivity (although very high specificity). Rather than conduct an exhaustive split read alignment search, we instead use indirect evidence of split reads that can be extracted from standard first-pass alignments:

- Soft clipping: When reads are aligned to the genome by short read mappers like BWA, they can in some cases be only partially aligned. In BWA, this occurs when a parameterized clipping penalty is less than the additional cost of aligning the rest of the read according to a Smith-Waterman dynamic programming alignment. Aligners refer to this "soft clipping", and flag the read with specific indicators when it occurs. Although there are other possible reasons for soft clipping (for example, the quality could fall at the end of the read, producing many erroneous mismatches with the reference), this could potentially be indicative of a structural variation breakpoint disrupting the read's alignment to the reference. For integration into the CRF feature set, we created a feature track that counts the number of times a soft clip occurred in each 25bp genomic window.

- Singleton alignments: For paired end reads, sometimes only one read in the pair can be successfully aligned to the genome. These are referred to as "singleton" alignments. Potentially a singleton alignment could indicate that the other read in the pair contains a variant that prevented it from being aligned, and therefore that a breakpoint could lie somewhere within the distance of the insert size, or could have disrupted the alignment of the other read in the pair. We count the number of singleton alignments in each window.

7.5.3 Read depth features

The third major signals of structural variations from short-read sequencing data are related to read depth. In the context of deletions, one would expect to see fewer reads aligning to sections of the genome which have been deleted in the sample; if the deletion is 100% represented in the sample and is homozygous, any reads aligning to that location must be incorrect alignments. In practice, coverage depth is affected by noise resulting from incorrect mappings and biases of the sequencing process (for example, the amount of GC content in a given fragment of DNA will affect its representation in the sequencing data set).

- Coverage depth: As one feature, we calculate the average coverage depth of each base in each 25bp window.

- Coverage change points: We applied the change point detection algorithm outlined in the previous section to identify windows at which the coverage changes. Although we experimented with different neighborhood sizes, we found that this was not a very valuable feature for breakpoint identification. For the results reported below, we used a neighborhood size of 600bp.

- Coverage Drops: We speculated that for the specific case of deletion detection, the most informative coverage statistic might be one that indicates whether or not the coverage at a given window drops relative to its local neighborhood. Therefore, we compute for each window the drop in coverage depth from the mean coverage depth of the windows that lie in the surrounding 500bp.

7.5.4 Genome annotations

Finally, the ability to include arbitrary features in the model allows us to incorporate prior knowledge about the genome into our feature set. In particular, some types of genomic features frequently cause incorrect alignments; training the model based on those feature annotations could help the variant caller calibrate its confidence in each prediction.

- Repetitive regions: We took the UCSC RepeatMasker track for the reference genomes used in the testing and training sets and created a binary feature for each window in the genome, which was true if the window overlaps with a repetitive element designated by RepeatMasker [175].

- Simple repeats: these are a subset of the repetitive regions track that are made up of repeated k-mers of lengths from one to six. Simple repeats are very difficult to align to, and are therefore the source of many alignment errors. We set a binary feature for each window if it overlapped with an element from the UCSC Simple Repeats track [22] for the reference genome.

- Segmental duplications: These are larger regions (10kb+) that have at least one other copy in the genome with high sequence similarity between the two regions, and are again the source of many alignment errors. We used the UCSC Segmental Duplications track to set binary features on each window in the genome.

7.5.5 Binarization of real-valued features

Although conditional random fields can support real-valued features, the fact that they are linear models means that it is often helpful to convert real-valued features into related set of binary-valued features for labeling problems. For example, this can help prevent the domination of the output by single real-valued features with very high values. Therefore, we built two binarization schemes into our code base for real valued features. The first allows the user who is training the system to specify a set of cut points $c_{1..k}$ which divide the range of the variable into $k+1$ bins. If the value of the feature lies in bin i, the system sets binary feature b_i to true and all others to false. The second binarization procedure is

Feature Name	Type	Column	Description	Cutpoints
mu1	binnedReal	1	μ_2 from Cloudbreak GMM	260.0,340.0
lr	cumulativeBinnedReal	2	Likelihood Ratio from Cloudbreak GMM	0.75,2.5,10.0,75.0,500.0
singletons	cumulativeBinnedReal	6	Singleton alignments	1.0,2.0
changePoint	cumulativeBinnedReal	7	Insert size change point	5.0,15.0,50.0
softClip	cumulativeBinnedReal	8	Soft clipped alignments	1.0,2.0,3.0
rdepth	cumulativeBinnedReal	9	Average read depth	0.1,0.25,0.5,0.75,1.0
covChangePoint	cumulativeBinnedReal	10	Depth of coverage change point scores	10.0,20.0,30.0
covDrop20	cumulativeBinnedReal	11	Coverage drop	5.0,10.0,20.0
simpleRepeat	boolean	13	Simple Repeat	
repeat	boolean	14	Repeat	
segdup	boolean	15	Segmental Duplication	

Table 7.1: Feature definition for the CRF training and test data. Each line indicates the feature's name and type, as well as the type of binning scheme (simple or cumulative), and the cut points to use when binning.

a cumulative binning scheme. Again, the user specifies cut points which divide the range of the variable into i bins. In this case, however, binary features are set to true for bin whose end point is greater than or equal to the actual value of the variable. To specify all feature definitions and their types, the user defines each feature variable, its location in the training and test data files, the binning scheme, and the cutoff points for binning. Table 7.1 shows the feature definitions used for the results reported later in this chapter. This scheme allows for rapid testing of different feature combinations and binning schemes, depending on what data is available.

7.5.6 A feature example

Figure 7.3 shows an example deletion from the NA18507 gold standard data set, along with tracks showing each of the features described above, as well as the original Cloudbreak call identifying the deletion and a call made by running inference on the trained CRF model. One can observe an elevated likelihood ratio in the first track covering the deletion. At or near the breakpoints of the deletion there are higher scores for the insert size change point, soft clip count, and singleton alignment features. The deletion span is also associated with lower coverage values.

7.5.7 Interaction and neighbor features

Several of these features may be most informative when considered in conjunction with other features. For example, low coverage depth may mean something different when it occurs in a repetitive element, since that region of the genome might be harder to align reads to. Similarly, the features of neighboring windows could also inform the choice of label; since the CRF makes no demands of independence between neighboring feature functions, it is possible to include these neighboring features in the model. To that end, in addition to adding the features defined above to each bin, we also add as separately labeled features:

- Interaction terms for all of the features for the current window.
- The features of the windows immediately before and after the current window.
- The features of the windows within a certain radius of the current window (currently 7 neighboring bins).

Although this creates a large feature space, we hope that by training with regularization the optimizer will be able to select the most important features and avoid overfitting.

7.6 Training the CRF

As noted in Chapter 6, there are limited complete SV annotations of real data sets available. Therefore, we decided to train on simulated data. It should be noted that training on simulated data only exposes the model to one form of noise: incorrect or incomplete mappings. In real data sets, there are additional sources of noise in short read data sets, including chimeric fragments; complex structural variations, and errors in the genome reference. Therefore, any gains that can be made by training on simulated data and testing on real data are less than the possible gains that could be achieved with a more realistic training set.

To increase the number of training examples, we took the original simulation of Chromosome 2 described in Chapter 6 and expanded it to a whole genome simulation. This

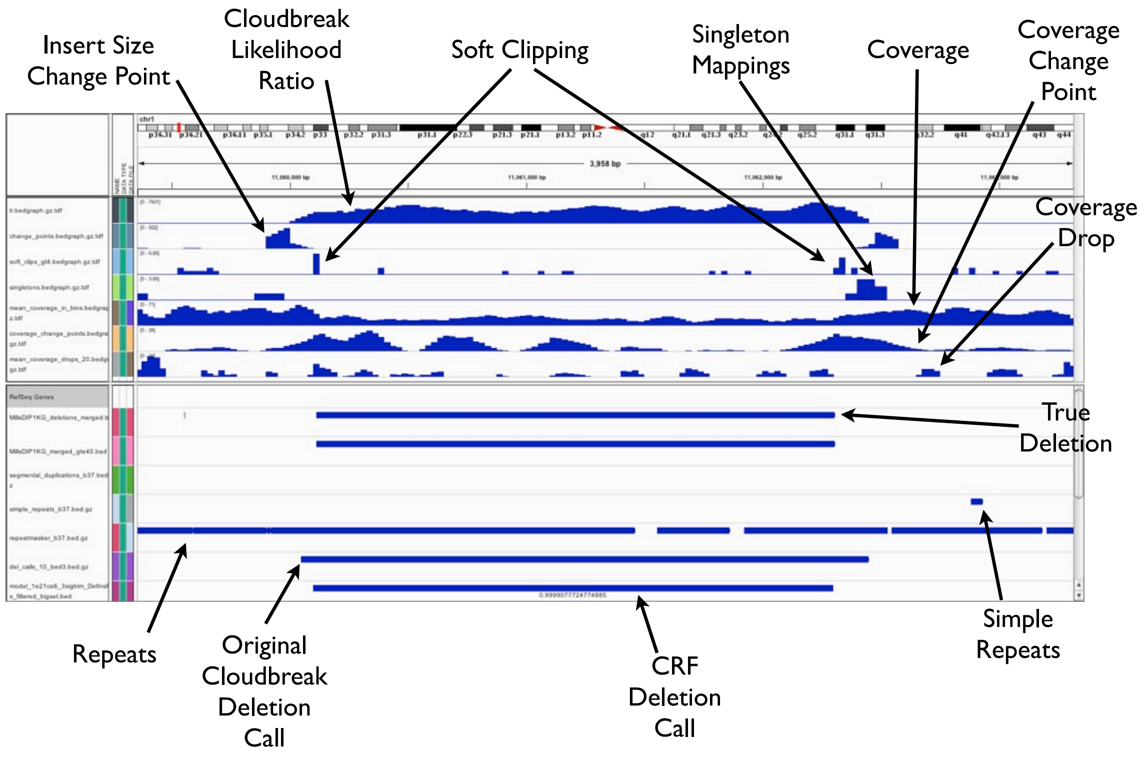

Figure 7.3: An example deletion with features from the NA18507 data set. The true deletion is shown in track 8. Features shown are in each track are: 1) Likelihood ratio of distribution of insert sizes. 2) Insert size change points. 3) Number of soft clipping events in each window. 4) Number of singleton mappings in each window. 5) Read coverage in each window. 6) Coverage change points. 7) Coverage drops. 11) Simple repeats. 12) Repeats. The original Cloudbreak call is shown in Track 13, and a call made by running inference in the CRF model is shown in Track 14.

also has the effect of increasing the possibility for mapping ambiguity, adding noise to the signal. The simulation includes all of the insertions and deletions annotated for J. Craig Venter's genome. Considering variants with a length over 40bp, there are 5,610 deletions and 6068 insertions. We simulated 30X coverage 100bp paired-end reads with an insert size of 300bp and aligned them to the reference genome using BWA, as described in Section 6.1.2.

We created a set of labels based on a bitmask representation of the following four states that a window could be in with respect to deletions and insertions: "Deletion",

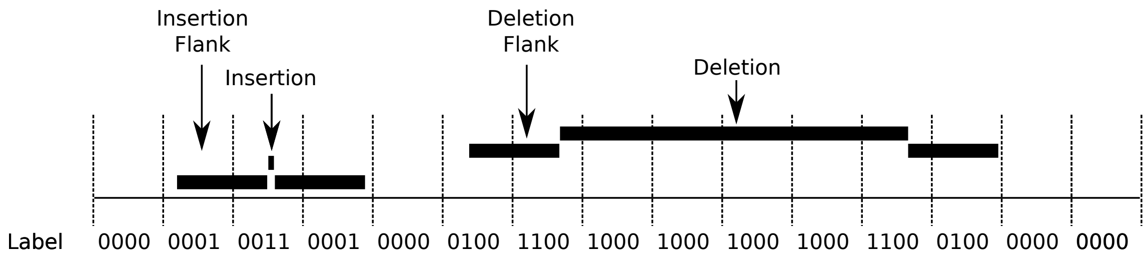

Figure 7.4: An example of the labeling scheme used for CRF model training and testing. Labels are a bitmask composed of variant and flanking information. First bit: Deletion. Second bit: Deletion Flank. Third bit: Insertion. Fourth bit: Insertion Flank.

“Deletion Flank”, “Insertion”, and “Insertion Flank”. We decided to include separate labels for flanking regions because several of the features we selected would be most likely to occur in the immediate flanks of deletion or insertion variants, such as singleton mappings, or the change point detection features. The use of a bitmasked label also means that we capture the windows that actually contain the breakpoints for each deletion or insertion under a separate label, since those windows contain both flanking sequence and variant sequence. See Figure 7.4 for an example of how labels are assigned to windows. This labeling scheme translates to a set of six commonly used labels: “Outside Variant”, “Insertion Flank”, “Insertion Breakpoint”, “Deletion Flank”, “Deletion Breakpoint”, and “Inside Deletion”. In our view, this set of labels represents the different categories of windows that are likely to have informative features. However, it is possible that other labeling schemes might prove to represent the feature space better; this would be a useful area for a more thorough future study.

We also had to choose what portions of the data to train the model on, since it would be infeasible to train on the entire reference genome, and would likely bias the model against predicting any deletions because the vast majority of the reference does not participate in a deletion variant. First, we took all of the simulated insertion and deletion variants, and added regions of 250bp with the label “Deletion Flank” to either side. In addition, we selected 10,774 Cloudbreak calls. Some of these overlapped with the true variants, but the others added regions to the training set which had produced false positives. We then expanded all of the 9,497 regions selected for training by first adding 400bp to either end, and then adding the length of the resulting region to both sides. This ensured that all true variants and false positives were flanked by many windows that were properly labeled "Outside Variant".

To train our model, we coded it in Factorie and used their implementation of the LGBFS algorithm with L2 regularization. We also experimented with other optimizers including the online AdaGrad regularized dual averaging algorithm [49] with L1 regularization but did not see large differences in preliminary testing. A limited difference in results on the test data combined with faster training times guided our choice of LGBFS; however, exploring different optimization methods with different feature sets could still be a useful area for research.

7.7 Improving Cloudbreak Calls with CRF Predictions

For our initial implementation, we adopted the approach of rapidly identifying candidate regions using a fast tool (Cloudbreak in this instance), adding flanking regions to them, and then running the CRF model on those candidate regions to try to label the true variants. Of course, this bounds the recall of the CRF approach to the recall of the candidate regions selected by the preliminary screen, but allows for a more tractable computation than trying to label the entire genome.

For the test set, we used the same data set for individual NA18507 described in Section 6.3, with 37X coverage by high-quality 100bp paired end reads with an insert size of 300bp. To choose regions to test on, we used Cloudbreak predictions for the same data set, at a low threshold to increase sensitivity. We then created test windows to run inference on in the CRF model by again adding 200bp of flank to each side of each deletion prediction interval, and then adding the length of the interval in each direction. To conduct inference we use the Viterbi max-product belief propagation algorithm implemented in Factorie.

We then took all windows that the CRF model labeled deletions and merged contiguous blocks of windows to form intervals. For each such interval, we assign a confidence score as follows: first, for each window in the interval we calculate the sum of the likelihoods assigned to labels which indicate a deletion (i.e. have the "Deletion" bit set to one). We then compute the average of this window score across the entire deletion interval, giving us an overall CRF confidence in the deletion interval as a whole.

Finally, we refined the initial set of Cloudbreak deletion calls using the CRF calls according to the following rule: if a Cloudbreak deletion call and a CRF deletion call share a reciprocal overlap of at least 60%, we consider that Cloudbreak call to be confirmed by the CRF model. By reciprocal overlap, we mean that 60% of the length of the Cloudbreak call overlaps the CRF call, and 60% of the CRF call overlaps the Cloudbreak call. For confirmed calls, we take the boundaries of the CRF deletion interval to be the deletion variant boundaries, and multiply the Cloudbreak deletion likelihood ratio by the CRF confidence score to get an adjusted confidence score for the confirmed call.

7.8 Results

Figure 7.5 shows a ROC curve that compares the confirmed CRF calls with the other methods including Cloudbreak. The sensitivity of the merged CRF predictions is limited compared to the other methods. However, the updated score, which combines the Cloudbreak likelihood ratio with the CRF confidence, does provide a small improvement in discriminative power, as the true positives occur at higher thresholds than when using the unadjusted Cloudbreak score.

Although the CRF method provides only a limited benefit to discriminative power over the raw Cloudbreak calls, it proves to be excellent at improving the breakpoint resolution of the calls. Figure 7.6 shows the breakpoint resolution of each of the methods, as reported previously in Chapter 6, with the addition of the set of Cloudbreak calls confirmed by the CRF model. The confirmed calls have dramatically better resolution than the original Cloudbreak call set. With CRF confirmation, Cloudbreak's median difference between the true deletion length and the predicted length, 17bp, is now similar to that of DELLY-RP, which at 16bp is the best performing read-pair based method according to this metric (methods that take advantage of split-read mappings, such as DELLY-SR and Pindel, of course do better in this category). When one considers that Cloudbreak's predictions and the CRF features and labels are all based on the use of 25bp windows, it is clear that most of the time the CRF predictions are accurately identifying the window which contains the

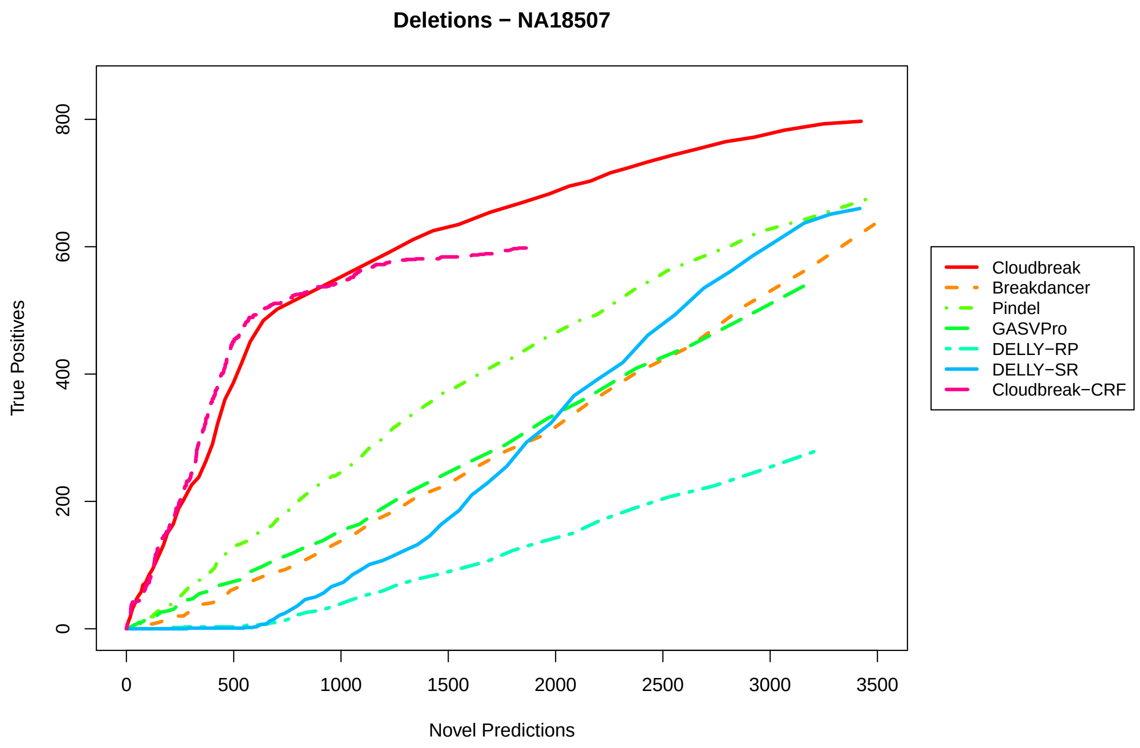

Figure 7.5: ROC curve showing the accuracy of Cloudbreak calls that have been verified and refined with conditional random field predictions (Cloudbreak-CRF)

true breakpoint.

7.9 Features Selected by the CRF

It can often be informative to examine the features that are given high scores by a machine learning technique, as they can sometimes provide insights into non-obvious features associated with certain labels. It should be noted, however, that a simple list of the highest-scoring features does not necessarily give a complete picture of the features that were important for learning performance. For example, if several features are correlated or appear with each other only in certain types of interactions, individual features within that correlated group may not be given high scores even though the group as a whole may have been quite important in learning the training data. This problem is compounded by the use of regularizers in training; in the case of L2 regularization, as used in these results,

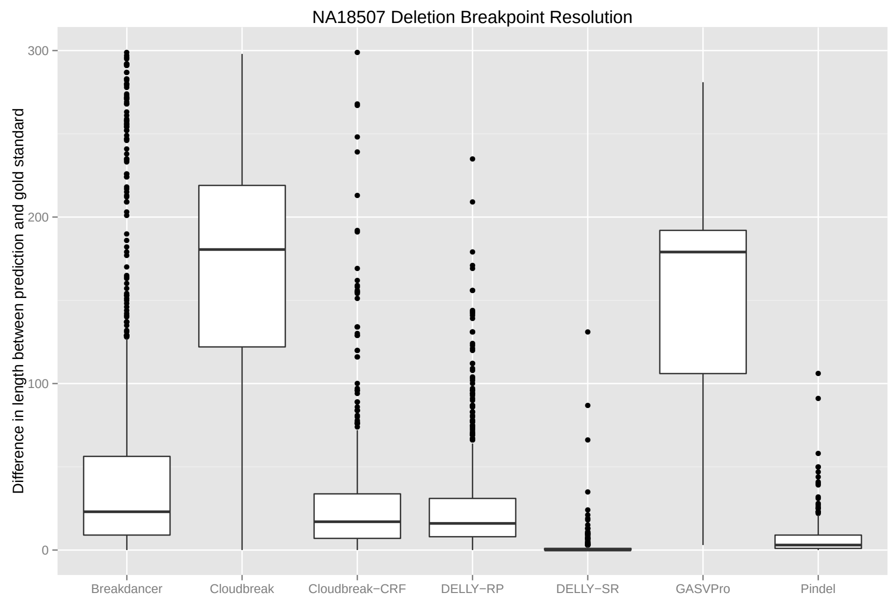

Figure 7.6: Breakpoint resolution of each tool's correct deletion predictions on the NA18507 data set, including Cloudbreak calls which were verified and refined with CRF predictions (Cloudbreak-CRF)

correlated features will all be reduced in weight similarly by the regularization term, potentially leaving none of them with high weights even though the group of features was highly significant in training.

Despite these caveats, we can still identify several interesting features with high scores in the trained CRF model. Table 7.9 shows the top 25 features learned by the CRF for the window labels that correspond to "Deletion Breakpoint" (the windows which contain the endpoints of deleted regions) and "Insertion". The CRF has learned a strong association between the deletion breakpoint label and the presence of simple repeats, showing how much correspondence there is between those features and the variants in the training set. As we might expect given the large improvement that the model achieved in localizing breakpoint locations, soft clipping-related features are given high scores, particularly when they occur in conjunction with coverage drops. The mu2 feature is used heavily for both deletions and insertions. Further examination of the feature scores may allow for improvements in feature engineering to create more parsimonious and effective models.

Deletion		Insertion	
Feature Name	Score	Feature Name	Score
simpleRepeat	0.6	softClip > 1.0	0.47
260 < mu2 < 340 and simpleRepeat	0.34	simpleRepeat	0.41
simpleRepeat neighbor	0.34	softClip > 3.0	0.39
rdepth < .1 and simpleRepeat	0.32	260 < mu2 < 340 and softClip > 1.0	0.38
260 < mu2 < 340 and simpleRepeat neighbor	0.29	softClip > 1.0 neighbor	0.31
softClip > 1.0 and covDrop20 > 5.0	0.23	softClip > 2.0	0.29
softClip > 3.0 and covDrop20 > 5.0	0.2	mu2 < 260 in region	0.26
mu2 < 260 and lr = 0 in region	0.19	260 < mu2 < 340 and softClip > 3.0	0.25
softClip > 1.0	0.18	rdepth < .1 and simpleRepeat	0.24
softClip > 3.0 and covDrop20 > 10.0	0.17	rdepth > 1 and simpleRepeat neighbor	0.23
softClip > 2.0 and covDrop20 > 5.0	0.17	softClip > 2.0 neighbor	0.22
mu2 > 340 and simpleRepeat	0.17	softClip > 1.0 in region	0.22
covDrop20 > 5.0 in region	0.17	mu2 < 260 and simpleRepeat	0.21
changePoint > 5.0 in region	0.16	simpleRepeat and repeat	0.21
softClip > 2.0 and covDrop20 > 10.0	0.15	singletons > 1.0 and covDrop20 > 5.0	0.21
covChangePoint > 10.0 in region	0.15	softClip > 2.0 in region	0.2
softClip > 1.0 neighbor	0.14	softClip > 3.0 neighbor	0.2
mu2 > 340 in region	0.14	260 < mu2 < 340 and softClip > 2.0	0.2
mu2 > 340 and lr > 0.75	0.14	rdepth > 1 and simpleRepeat	0.2
changePoint > 15.0 and rdepth < .5 in region	0.14	softClip > 1.0 and rdepth > 1	0.16
mu2 > 340 and softClip > 1.0	0.13	mu2 < 260 neighbor	0.16
softClip > 1.0 and covDrop20 > 10.0	0.13	lr > 0.75 and simpleRepeat	0.15
260 < mu2 < 340 and covDrop20 > 10.0 neighbor	0.13	260 < mu2 < 340 and lr = 0 in region	0.15
rdepth < .1 and covDrop20 > 5.0 neighbor	0.13	softClip > 1.0 and repeat	0.15
rdepth < .25	0.13	changePoint > 5.0 and softClip > 3.0	0.14

Table 7.2: Most important features learned by the CRF model for deletion and insertion breakpoints. "Neighbor": feature occurs in neighboring window. "In region": feature occurs within 300bp of current window. Feature names are the same as those in Table 7.1

7.10 Discussion

With the CRF model presented in this chapter, we have shown that the use of local features is a useful abstraction that can enable innovative algorithmic approaches to the SV detection problem. We redefined SV detection as a sequence labeling problem, and created a framework for generating and managing local features across the genome that can integrate read pair, split read, and read depth related features, as well as features that incorporate prior knowledge such as genome annotations. Despite only training with simulated data that likely does not incorporate all of the aspects of the SV detection problem that make it difficult, we were able to generate useful predictions from our model.

Although the improvements in accuracy we achieved with the CRF model are marginal at best, it did prove to be very effective in improving Cloudbreak's breakpoint resolution, which was one of the weak points of the initial implementation of Cloudbreak as described in the previous chapters. The fact that for the majority of correctly predicted variants the CRF model was able to correctly identify the windows in which the breakpoint occurred indicates that the resolution could potentially be further improved with reductions in window size. In addition, those windows identified could be excellent targets for exhaustive split read analysis or local assembly.

Opening the door to arbitrary features potentially allows the inclusion of a variety of types of information useful for SV detection in a single unified framework. For example, other other types of genomic annotations could be very useful for detecting where the SV signals might be distorted due to sequence content. In this work we used binary features indicating the presence of repeats and segmental duplications. There are more fine-grained measures of repetitiveness and its effect on short read alignment, such as the genome mappability score [100] that could help to distinguish difficult regions. In addition, GC content in genomic sequence is known to have an effect on sequencing depth and copy number estimation [19] and could be added as an additional feature to help correct for these biases. Another type of feature that similarly represents prior knowledge about the genome and could be incorporated into this framework is sets of predictions from different SV detection tools. The SV detection methods that have achieved the greatest overall accuracy for non-cancer domains are those that pool multiple samples from across the same population, such as Genome STRiP [65]. The pooled signals are used to identify candidate variant regions, which are then genotyped in each sample separately in a second pass over the data. These candidate variant regions could also be used as features within an integrated machine learning technique.

Finally, the characterization of the problem as a generic sequence labeling task over arbitrary features could potentially enable the use of a variety of different statistical and machine learning techniques for the SV detection problem in addition to the CRFs explored in this chapter. For example, deep learning techniques such as sequential deep belief networks [7] have recently been used very successfully in sequence processing tasks in speech recognition and other domains. Since it is difficult to find fully annotated training data for the SV detection problem, the fact that deep learning networks are amenable to semi-supervised training with unlabeled data [192] makes them a very attractive area for future research.

Chapter 8

Analysis of Evolutionary Breakpoint Features

This chapter describes a data analysis project unrelated to the algorithmic development of Cloudbreak and its extensions, although it is related at the biological level because it deals with genome breakpoints and rearrangements. In addition to being able to locate genomic breakpoints, we also seek to understand the genomic contexts in which they are located to gain insights into their origins. As we discussed in Chapter 2, the sequence features of the genome that neighbor structural variations can leave clues about the mechanisms that caused the rearrangements, as in the cases of large homologies that point to the activity of non-allelic homologous recombination (NAHR) or the microhomologies that indicate non-homologous end joining (NEHJ). In this chapter we examine a set of structural variations that have reshaped a genome at the evolutionary timescale: the chromosomal rearrangements of the gibbon genome. We analyze the locations in which the gibbon genome has been rearranged relative to humans and the other apes, first using a limited set of breakpoint data, and then using the entire gibbon genome reference sequence created by the International Consortium for Sequencing and Annotation of the Gibbon Genome. The associations we discover could help to provide insights into the mechanisms by which the gibbon genome was rearranged. The main contributions of this chapter are 1) the statistical analysis of the enrichment or depletion of certain genomic features in regions of the gibbon genome that have participated in genomic breakpoints, 2) development of an open-source software tool that can harness compute clusters to conduct permutation analysis for statistical enrichment of features in the genome, and 3) a novel, cross-species analysis of the locations of binding sites of the transcription factor CTCF with respect to genome breakpoints in the gibbon family.

8.1 Background

In Chapter 2, we described how SVs can occur in the genomes of many different populations, ranging from rearrangements that occur in populations of cells within a tumor in a single individual, to SVs that are polymorphic within the population of a species, to the structural differences between the genomes of different species. This last type of rearrangement represents SVs that occurred within a population and then became fixed in that species and were preserved as it evolved and differentiated.

Rather than identifying them by comparing short-read sequencing data to a reference genome, as has been the focus of the previous chapters, we can detect these *evolutionary breakpoints* by directly comparing the genomes of species to one another. This can be done experimentally, by large fragments of DNA from one species onto the chromosomes of another; in Section 8.2 we will describe the analysis of data from one such experiment that used a technique based on *bacterial artificial chromosomes* (BACs). With the advent of whole genome sequences they can be found by conducting sequence alignments (or multiple sequence alignments) of the the reference genomes for the species under study. When species share regions that contain the same genes and other genomic features, in the same order, it is referred to as *shared synteny.* By examining the endpoints of syntenic blocks of the genomes of two species, or *synteny breakpoints*, it is possible to find the breakpoints of the structural variations that rearranged the structure of the two genomes.

Identifying evolutionary breakpoints can give insights into the mechanisms of their formation. It was long thought that chromosomal breakpoints are largely random, such that they are equally likely to occur at any location in the genome. This theory is known as the *random breakage* model [144, 134]. Higher-resolution analysis, however, revealed that certain regions of the genome, or "hot-spots", are much more likely to give rise to evolutionary breakpoints [148], and this model, called the *fragile breakage* model, was re-confirmed once whole-genome reference sequences for many species became available [132].

Tantalizingly, Murphy et al. also showed that these reused evolutionary breakpoint regions overlapped significantly with the breakpoints of SVs found in cancer genomes [132].

This raises the question of whether these regions of the genome have special properties that make them more likely to participate in evolutionary breakpoints. In Chapter 2, we described the fact that certain mechanisms for SV formation, in particular NAHR, are a result of having regions with high sequence similarity at different locations in the genome, as the cell's machinery attempts to repair double-stranded breaks in DNA by joining homologous regions together. As one might expect if evolutionary breakpoints were the result of NAHR, a large number of chromosomal rearrangements in mammals, approaching 40%, are associated with segmental duplications [15, 16]. Closer examination of breakpoints from a variety of species, including human, chimp, macaque, rat, mouse, pig, cattle, dog, opossum, and chicken, confirmed that they contain more sequence that is not unique relative to the rest of the genome than would be expected by chance, including segmental duplications (SDs) and other low-copy number repeats [97]. However, analysis of evolutionary sequence divergence in these repeats indicates that some of the duplications may have occurred after the rearrangements, casting doubt on the theory that an NAHR-like process is primarily responsible [15], and indeed the precise mechanisms behind evolutionary breakpoints remain unknown.

Gibbons represent a unique opportunity to study the process of evolutionary genomic rearrangement because of their highly rearranged genomes. In most branches of the evolutionary tree, large genomic rearrangements are rare events, occurring at a rate of approximately two every ten million years [193]. However, some species appear to have faster or slower rates of rearrangements. For example, the orang-utan genome has very few medium and large scale rearrangements relative to the expected number given its time of evolutionary divergence [114]. At the other end of the spectrum, gibbon genomes have 10 to 20 times more rearrangements than would be expected given the rate in other mammals, and in fact the four genera of gibbons have numbers of chromosomes ranging from 28 to 52, suggesting that gibbon genome has been shuffled rapidly since it diverged from the other apes 17 to 18 million years ago [130].

Given their highly rearranged karyotypes, gibbons represent an excellent model in which to study mechanisms of chromosomal rearrangements. Until recently, however, the whole genome sequence of the gibbon was not available. Therefore, several studies analyzed gibbon genome breakpoints using techniques involving BACs [62, 30, 29]. BACs are medium sized (150-350kb) fragments of DNA from a sample that are isolated, turned into plasmid bacterial chromosomes, and then amplified in a culture. By probing gibbon BACs with arrays or testing their hybridization to human chromosomes using fluorescent *in situ* hybridization (FISH) experiments, it is possible to isolate BACs which span human-gibbon evolutionary breakpoints. This allows one to locate the breakpoint regions to within several hundred kilobases, and the small size of the BACs relative to the entire genome makes them much easier to sequence, allowing the analysis of sequence features near the breakpoints. Initial studies examined selected BACs from the northern white-cheeked gibbon species *Nomascus leucogenys leucogenys* (NLE) and found a high level of segmental duplications and repetitive sequence at the breakpoints [30, 159]. Similar results were found in BACs from the white-handed gibbon species *Hylobates lar* (HLA) [130]. A later study increased the scope by examining 24 sequenced NLE BACs that spanned breakpoints, and analyzed gibbon-specific insertions of repeats and segmental duplications at the breakpoints to postulate several possible rearrangement mechanisms [62]. More recently, Carbone et al. [29] analyzed 57 NLE BACs and, in addition to segmental duplications, found enrichment for *Alu* repetitive elements. The latter study also found differences in CpG methylation marks on *Alu* elements near the breakpoints, suggesting that there may be an epigenetic process involved in gibbon genomic rearrangements.

8.2 Evolutionary Breakpoints in the Gibbon Genome Identified by BACs

To determine whether or not the characteristics observed in the breakpoints identified by BACs from NLE and HLA gibbons were generalizable to the entire gibbon family, we conducted an expanded analysis of gibbon breakpoints based on the identification of additional BACs covering breakpoints in the remaining two gibbon genera using samples from the species *Symphalangus syndactylus* (SSY) and *Hoolock leuconedys* (HLE). Since each of the gibbon genera has a different karyotype, identifying breakpoints across all four allows for greater insight into the mechanisms and timing of the evolution of the gibbon family. I contributed to this research by analyzing the overlap between breakpoints from all four gibbon genera with genomic features. This work was published in Capozzi et al. [28].

8.2.1 Methods

To determine whether there exists an enrichment or depletion for genomic features associated with gibbon chromosomal breakpoints we computed the significance of their overlap using Monte Carlo permutation tests. The purpose of the permutation analysis is to discover the null distribution for the number of overlaps a set of intervals has with a particular feature in the genome. In essence, the test asks: if my intervals were placed in random locations on the genome, what is the probability of seeing the number of overlaps we observed with that features in the actual data? In this case, the intervals whose locations we permute are the gibbon breakpoint regions.

For this test, we identified the human regions that are syntenic to the locations of the breakpoints in gibbons. We then permuted their start coordinates 10,000 times using BEDTools version 2.16.2 [155], while maintaining the chromosomal assignment and length of breakpoint regions. Genomic regions annotated as centromeres and telomeres in the "Gaps" track of the hg19 build were excluded from possible random placements of the regions. Locations of the features were held constant. We then compared the actual number of features that overlapped a breakpoint region to the distribution of overlap counts among the randomly permuted regions, and used the quantile of the real observed value in that distribution as an estimate of the p-value of observing a value equal to or greater than the real observation. For events that are rare, it is necessary to consider a large number of permutations or Monte Carlo samples in order to accurately estimate the p-value [160]. In order to facilitate conducting many permutations, we created a pipeline for distributing the necessary computation across a compute cluster using the grid management system HTCondor [181].

The analysis was performed on the human hg19 assembly. The features examined were genes, human segmental duplications, and some repeat families (*Alu*, LINE, ERV, and

SVA). We also investigated the associations between breakpoint regions and chromatin structure by testing the overlap with open chromatin regions in human embryonic stem cells reported by the ENCODE consortium [52].

8.2.2 Results

We found a significant enrichment for genes (Bonferroni adjusted P-value = 0.0287), human segmental duplications ($P = 0.0366$), *Alu* ($P < 0.0001$), and SVA ($P = 0.0008$) Figure 8.1 displays the histograms of overlap counts in the random distributions for segmental duplications, *Alu* elements, and SVA elements. We did not find significant enrichment for LINE and ERV repeats, nor for the ENCODE open chromatin regions. SVA elements are not found in gibbons; therefore, their status as significantly enriched in the human syntenic breakpoint regions is surprising. However, SVAs are known to correlate with *Alu* elements due to their preference for G+C rich regions of the genome (Wang et al. 2005). It seems likely that the association between human SVA locations and gibbon breakpoint regions is therefore an indirect one, dependent on the presence of additional genomic features present in both humans and gibbons.

Systematically shifting the location of breakpoint regions by increments of 10 kb up- and downstream of their actual location, up to a maximum of 1 MB, shows that the locations of the breakpoint regions gives the greatest or close to the greatest number of overlaps with the four significantly overlapping features (genes, segmental duplications, *Alu*, and SVA) in the local genomic neighborhood, also shown in Figure 8.1.

8.3 Analysis of Breakpoints from the Gibbon Genome Reference Sequence

While analysis using breakpoints identified with BAC clones has provided useful results, the International Consortium for Sequencing and Annotation of the Gibbon Genome has recently created a draft assembly of the whole gibbon genome for an NLE individual. Comparison of the assembly sequence with the human reference genome has enabled the identification of a complete set of synteny breakpoints between the human and gibbon

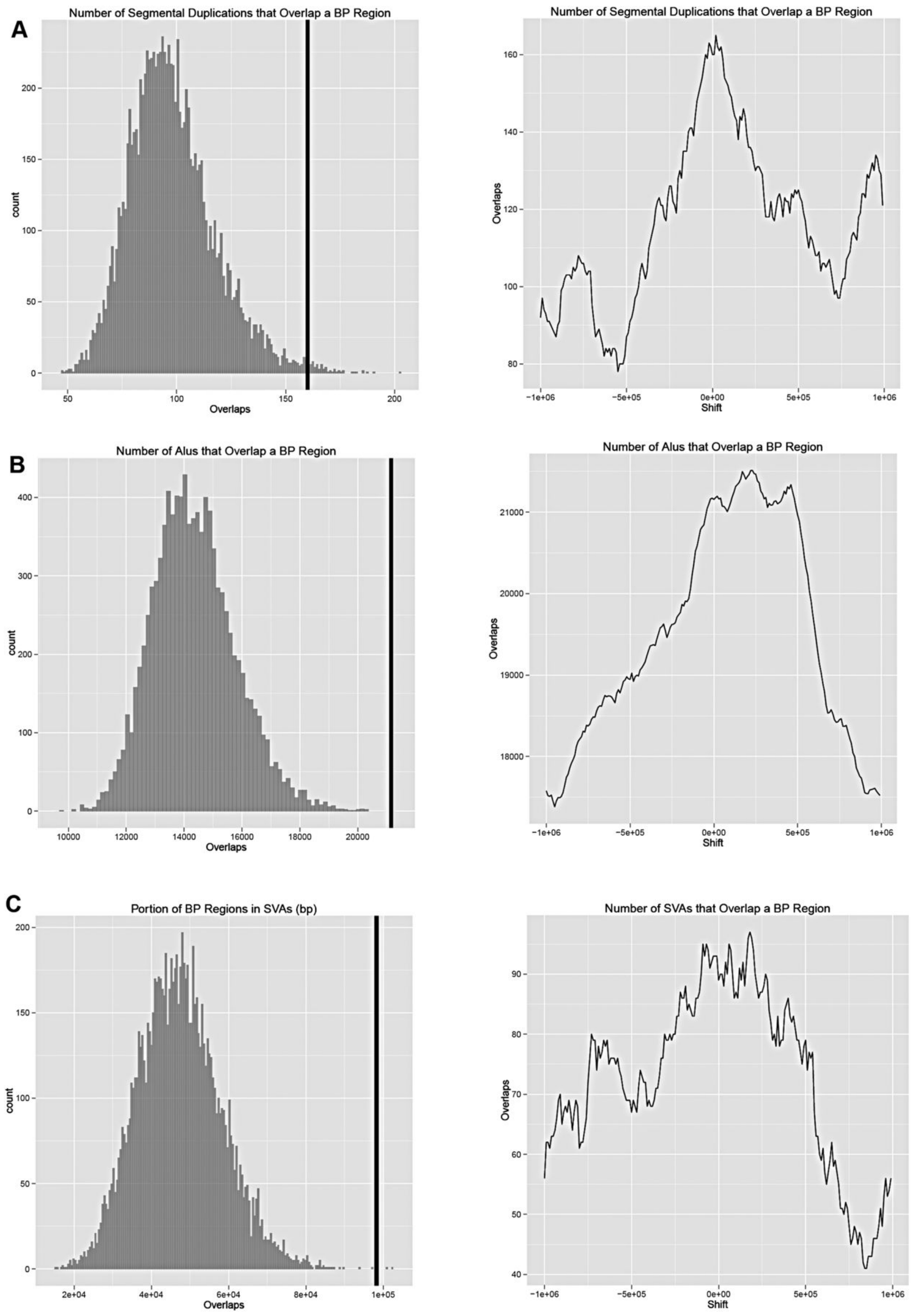

Figure 8.1: Enrichment of genomic features in regions of the human genome syntenic to gibbon breakpoint regions identified using BACs. Permutation tests were used to assess the overlap between the gibbon breakpoints and genomic features. (A) Segmental duplications; (B) *Alu* elements; and (C) SVA elements. The black vertical line indicates the observed value for the breakpoints identified in the study. In all three cases it is evident that the genomic features have a higher overlap with the breakpoints than one could expect by chance. Figure reproduced from Capozzi et al. [28].

genomes at single nucleotide resolution. This allows for a much more comprehensive picture of the sequence features surrounding gibbon breakpoints than was previously possible. As a member of the project, I was responsible for the breakpoint analyses described below.

8.3.1 Overlap with Genomic Features: Repeats and Genes

To analyze the enrichment of genomic features in the regions flanking evolutionary breakpoints, we used a similar permutation based approach. We compared the number of overlaps between breakpoint flanks and each feature of interest in the assembled genome (the observed overlap count) compared to a background distribution estimated by randomly permuting the locations of breakpoint regions 100,000 times. The Nleu1.1 version of the assembly was used for these analyses. To create breakpoint regions, for those breakpoints for which we had single nucleotide resolution we added flanking regions to either side of the breakpoint; for breakpoints that fell within a gibbon-specific repeat element, we chose the flanking regions of the repeat. In each permutation, the location of each breakpoint region was randomly changed, while keeping its length and scaffold assignment the same. We then counted the number of overlaps between the randomized breakpoint regions and the feature of interest (the permuted overlap count). Enrichment p-values were computed as the proportion of permuted overlap counts that were more extreme than the observed overlap count. We also visualized the spatial relationship of breakpoint regions to each type of feature by simultaneously shifting the locations of the breakpoint regions up to 1MB in each direction, in increments of 25kb, and counting the proportion of shifted breakpoint regions that overlapped a feature of interest, after discarding regions that were shifted beyond the beginning or end of a scaffold. Permutation testing and shift testing were carried out using custom Python scripts and the BEDtools [155], pybedtools[43], and BEDOPS [136] libraries. For this project we enhanced our testing pipeline, adding command-line options to support the distribution of permutations across Condor clusters, and made the code publicly available at `https://github.com/cwhelan/permuting-feature-enrichment-test`.

We tested for enrichment in the breakpoint regions of genes, segmental duplications, and several classes of repetitive elements: *Alu*, L1, LAVA, and LTR. In addition to testing the entire *Alu* family, we also tested the subfamilies *AluS*, *AluJ*, and *AluY* individually.

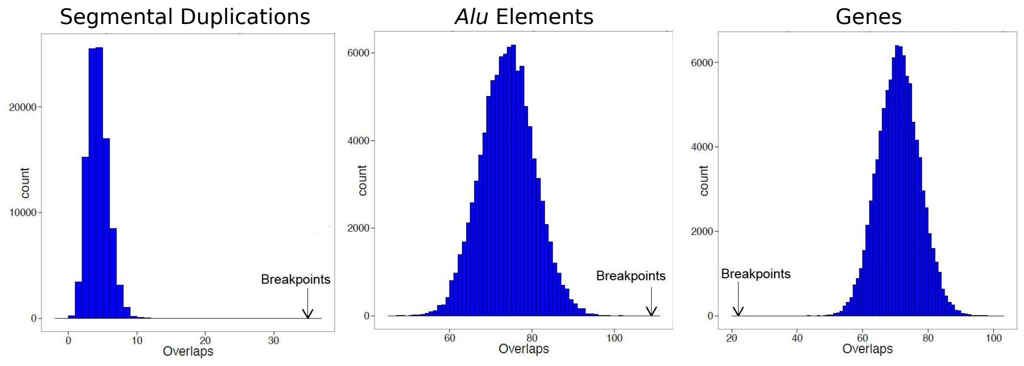

Figure 8.2: Enrichment or depletion of gibbon genome breakpoints for several features. The number of overlaps between the 1kb regions flanking gibbon-human synteny breakpoints and segmental duplications, *Alu* elements, and genes in the gibbon genome is shown by the arrow labeled "breakpoints". Histograms represent the random distribution for these overlap counts obtained by permuting the locations of the breakpoints 100,000 times.

Gene locations were taken from Ensembl build 70. For the segmental duplication analysis, we used the segmental duplications identified by our collaborators in the Eichler laboratory using the WSSD method [17]. *Alu*, L1, and LTR locations were identified using RepeatMasker output. In order to determine the distance from the breakpoints at which enrichments are strongest, we varied the size of the breakpoint flanking regions by adding differently sized intervals; we tested flanking regions of size 100bp, 250bp, 500bp, and 1000bp. We corrected for multiple testing using the FDR under dependency method of Bejamini and Yekutieli [20].

Breakpoint regions are depleted for genes, but enriched for *Alu* elements and segmental duplications (Figure 8.2, Table 8.1). The enrichment for *Alu* becomes particularly strong at distances of 750bp or 1000bp from the breakpoint (Figure 8.3). The enrichment for *Alu* is primarily due to a strong enrichment of the *AluS* subfamily.

We also conducted a shift analysis similar to the one we described in Section 8.2. Breakpoint regions were simultaneously shifted in increments of 25kb, up to a maximum of 1MB in each direction, and the proportion of breakpoint regions that overlap a feature of interest is reported. In Figure 8.4, the shifts show that breakpoints are centered on regions that are depleted in genes but close to regions that contain genes, while the opposite is true for segmental duplications. *Alu* elements are more evenly spread across the shift regions.

Feature	Breakpoints that Overlap	Quantile	Adjusted -1 * log(P-value)
Alu	109	1.0000	8.8696
AluJ	35	0.9952	3.2191
AluS	81	1.0000	8.8696
AluY	26	0.9918	2.7599
CTCF Peak	12	0.9990	4.5930
Gene	22	0.0000	-8.8696
L1	70	0.9516	1.2222
LAVA	1	0.9755	1.7284
LTR	48	0.9697	1.6054
WSSD Segmental Duplication	35	1.0000	8.8696

Table 8.1: Enrichment counts and scores of features in breakpoint flanking regions. For each feature type, we display the number of 1kb regions flanking breakpoints that overlap with a feature of that type, the quantile of that count in the empirical distribution obtained by permuting breakpoint flank locations 100,000 times, and the negative log FDR of that quantile treated as a p-value. Negative values indicate a depletion rather than an enrichment. Prior to FDR correction quantiles of zero were adjusted to p-values of 0.00001.

In conjunction with our collaborator Larry Wilhelm, in addition to testing the count of overlaps between breakpoint flanking regions and repeats, we also conducted a complementary test that examined the distance of each breakpoint to the nearest repeat of a given class. For this test we used only the 44 breakpoints for which we had single nucleotide breakpoint resolution, with no Gibbon-specific repeats directly intersecting the breakpoint. We compared these breakpoint locations to 10,000 randomly selected regions in the nomLeu2 genome. For each random location, we determined the distance to the nearest repeat. We compared the distribution of distances to a repeat from the breakpoints to the distribution of distances to a repeat from the randomly selected positions using a Kolmogorov-Smirnov test. We examined the distance to any repeat, as well as those for *Alu*, LINE, and LTR elements, and finally the *AluJ*, *AluS*, and *AluY* subfamilies. After FDR correction for multiple hypothesis testing, the distance test showed similar results to the overlap test described above, with significant results for the *Alu* family as a whole, and for the *AluJ* and *AluS* subfamilies, indicating that the breakpoints tend to be closer to those repeats than random locations in the genome.

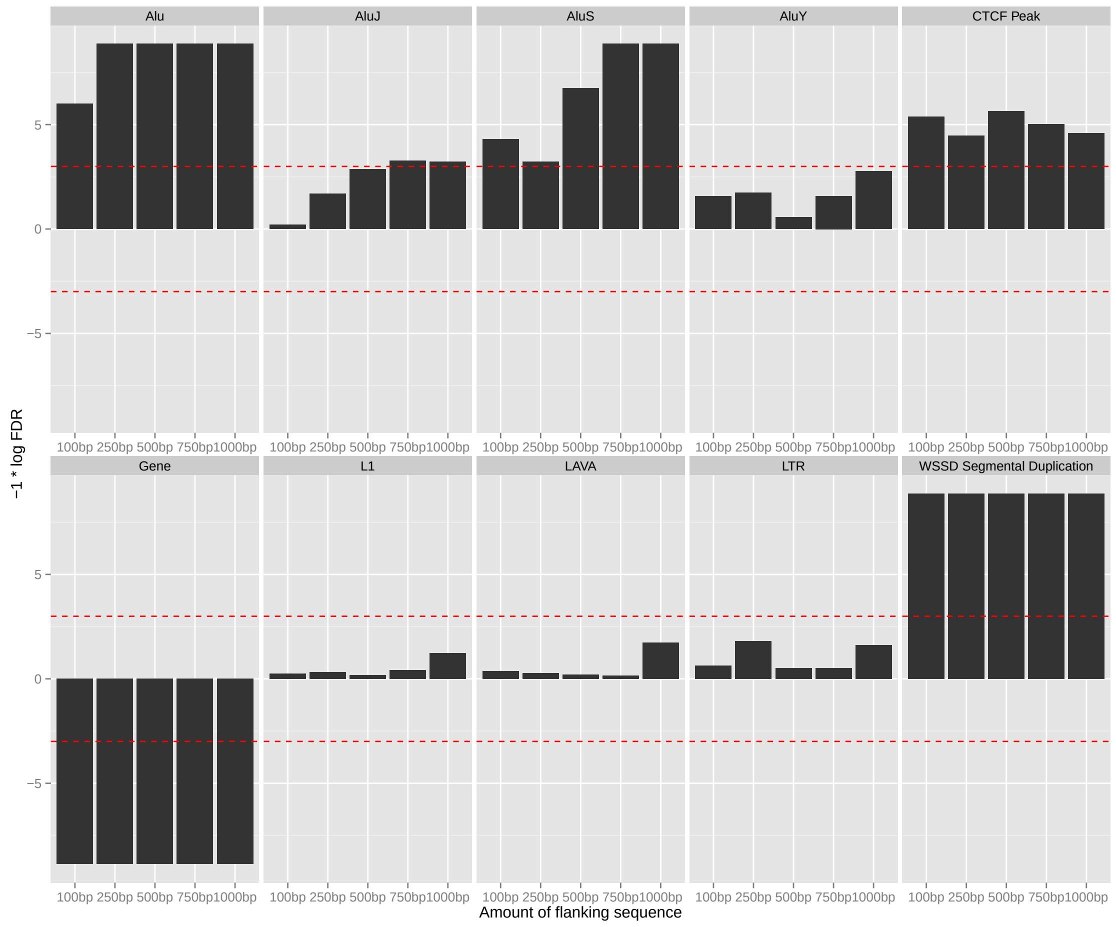

Figure 8.3: Enrichment or depletion of regions of the gibbon genome for various features, as a function of distance from the breakpoint. Positive y values are the negative log q-values of enrichment for each feature in the regions flanking the breakpoint of the size indicated on the x-axis. Negative y-values represent the q-values of depletion. Q values were calculated using the FDR correction under dependency procedure. The dotted line indicates an FDR threshold of 0.05. Breakpoints are enriched for *Alu* elements, and particularly the *AluS* subfamily, especially at distances of 750bp and 1000bp from the breakpoints. Breakpoints are also enriched for segmental duplications, depleted for genes, and enriched for CTCF binding sites (see Section 8.3.2.)

8.3.2 Overlap with CTCF Binding Sites

Using the gibbon genome reference, we conducted another set of analyses to determine the relationship of gibbon genome breakpoints with binding sites of the evolutionarily conserved binding factor CTCF. CTCF is a transcription factor with many functions,

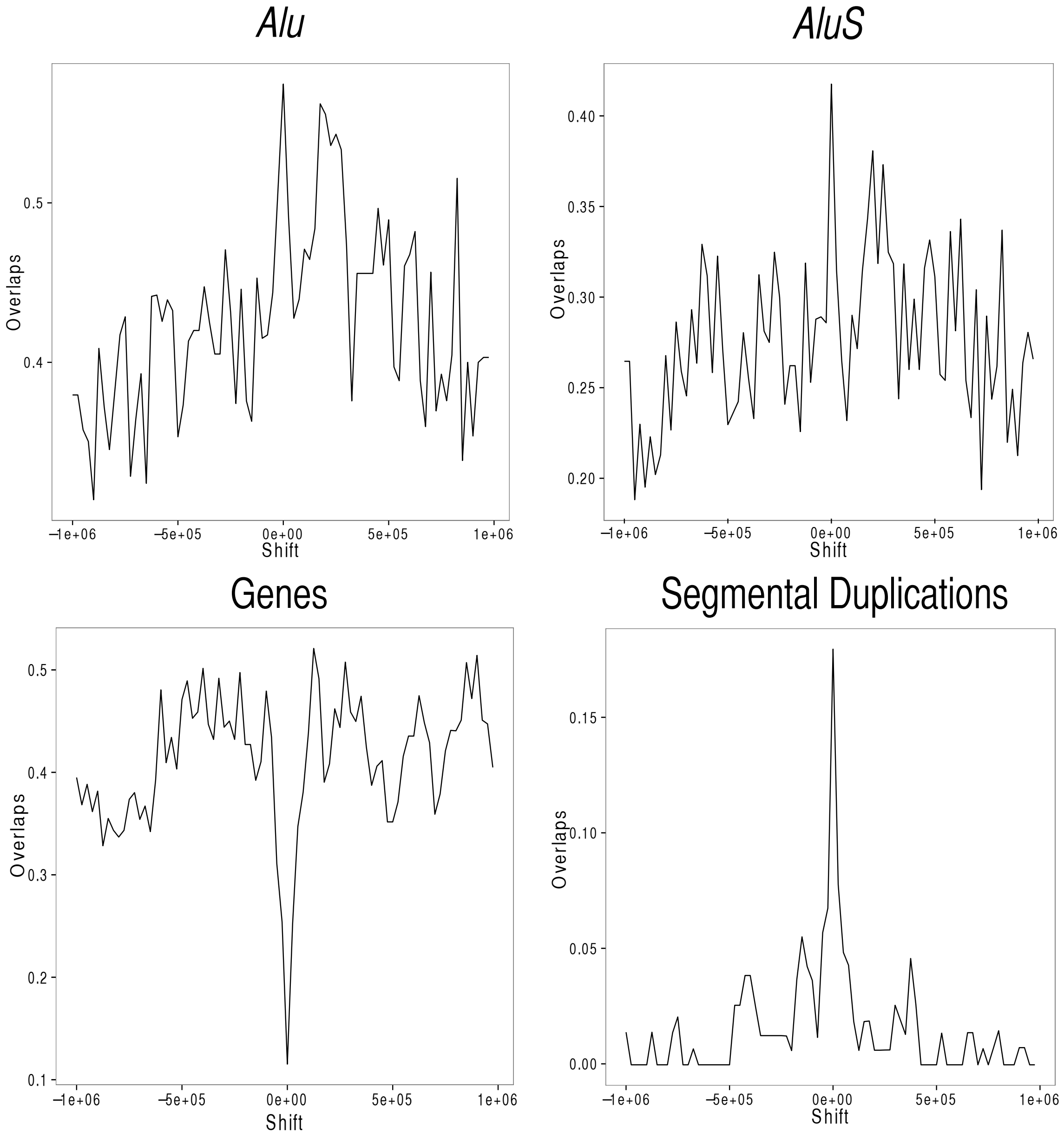

Figure 8.4: The proportion of breakpoint flanking intervals (defined as the 1000bp intervals flanking exact breakpoint locations or gibbon-specific repeats located at the breakpoints) which overlap with a feature of interest, when the flanking intervals are shifted left or right from their actual locations. Breakpoints are located in locations enriched in *Alu* (especially *AluS*) and segmental duplications relative to their genomic neighborhoods, but which are depleted for genes relative to their genomic neighborhoods.

including the ability to separate regulatory domains in the genome from one another [149]. CTCF has also recently been shown to be associated with the three-dimensional structure of DNA and chromatin in the cell [48], and therefore may have important associations with genomic DNA breakpoints and structural variations. We examined CTCF in the gibbon through the analysis of chromatin immunoprecipitation followed by sequencing (ChIP-seq) data [147]. Briefly, in ChIP-seq DNA from a sample is cross-linked *in vivo* and then fragmented, so that proteins (in particular transcription factors) remain bound to the fragments of DNA that contain their binding sites. After selecting only those fragments with a bound protein of interest using immunoprecipitation, the DNA is then sequenced. Mapping the reads back to the genome reference sequence creates "peaks" of coverage along the genome that indicate the location of transcription factor binding sites. In this analysis we identified a set of CTCF binding sites using gibbon ChIP-seq data, and conducted a comparative analysis using CTCF peaks from other primate species to separate CTCF peaks shared with a common ancestor from those that are unique to gibbons.

ChIP-seq for CTCF

CTCF ChIP-seq assays were performed in the Carbone laboratory by Elizabeth Terhune and Michelle Ward (visiting from the laboratory of Duncan Odom), according to the protocols described in Schmidt et al. [166] on eight EBV-transformed gibbon lymphoblastoid cell lines. In brief, CTCF-bound DNA was immunoprecipitated using an Anti-CTCF rabbit polyclonal antibody (07-729, Millipore). End-repair was performed on immunoprecipitated and input DNA prior to A-tailing and ligation to single-end Illumina sequencing adapters. DNA was amplified using Illumina primers 1.1 and 2.1 in an 18-cycle PCR reaction. Gel electrophoresis was used to select 200-300 bp DNA fragments. DNA libraries were sequenced using 36 bp reads on an Illumina Genome Analyser II according to the manufacturer's instructions.

Peak Calling from CTCF ChIP-seq Data

We aligned reads to the nomLeu2 reference using BWA (version 0.62) [106] with default parameters, and removed non-uniquely mapping reads. We then called peaks using

CCAT [196], with parameters fragmentSize 100, slidingWindowSize 150, movingStep 10, isStrandSensitiveMode 1, minCount 10, minScore 4.0, and bootstrapPass 50. We then combined the peaks called across the different individuals and chose the following set for further analysis: any peak called in an individual individual by CCAT with an FDR of less than 0.05, as well as any peak that was called in more than one individual with an FDR of less than 0.1.

Determination of Gibbon-specific and Shared CTCF Binding Sites

We classified Gibbon peaks as shared or gibbon-specific by comparing them to a set of CTCF peaks called on human, macaque, and orangutan individuals [168]. This work was carried out in conjunction with our collaborators Javier Herrero of the European Bioinformatics Institute and Larry Wilhelm. First, orthologous locations of gibbon CTCF peaks in other primate species were determined using a local installation of the EnsemblCompara multi-species alignment database [53, 186]. This database contains alignments of the references for human (GRCh37), chimp (CHIMP2.1.4), gorilla (gorGor3.1), orangutan (PPYG2), rhesus macaque (MMUL_1), and gibbon (Nleu1.1). Gibbon CTCF peaks for which no multi-species alignment were present, or where the nucleotide alignment identity to human and macaque was less than 70%, were excluded from analysis. We then created a non-redundant list of non-gibbon CTCF peaks by converting the human, orangutan and macaque peaks to gibbon coordinates using the Compara database and removing redundant entries. Shared and gibbon-specific CTCF peaks were then identified as those gibbon CTCF peaks that did or did not intersect a peak in the non-redundant list of non-gibbon CTCF peaks.

Analysis of Enrichment for CTCF Binding Sites in Gibbon Genome Breakpoints

We identified 52,685 gibbon CTCF binding sites from the eight individuals. We found that 12 of the 1kb regions flanking breakpoints overlap with CTCF binding sites (see Figure 8.5 for an example). Using the same permutation analysis described above, this overlap has an enrichment p-value of 0.0009. This effect was even stronger when we expanded the breakpoint flanking regions to 25kb, as 95 expanded flanking regions overlap CTCF peaks for an enrichment p-value <0.0001.

We then tested whether the CTCF peaks causing the enrichment in breakpoint regions are specific to gibbons. Using the CTCF ChIP-seq data from other species described above, we classified CTCF binding sites as unique to gibbons (11,449 sites) or shared with a primate ancestor (41,236 sites). We found that the gibbon breakpoint regions are heavily enriched for CTCF binding sites shared with a primate ancestor (enrichment p-value $= 0.0006$) but are not significantly enriched for gibbon-specific CTCF binding sites. Again, the enrichment is stronger in the 25kb expanded breakpoint regions (enrichment p-value <0.0001) for shared binding sites, but not for gibbon-specific binding sites. This suggests that the formation or selection of gibbon genome rearrangements was associated with ancestral CTCF binding sites.

8.4 Discussion

In our work examining breakpoints that included data from HLE and SSY gibbons, we extended previous analyses that had only considered data from NLE. This confirmed that the associations between breakpoints and segmental duplications, genes, and repetitive elements of the *Alu* family previously observed in NLE extended to breakpoints in other genera of the gibbon family. Extending this analysis to the other genera should assist the study of gibbon chromosome evolution, especially when used in conjunction with the release of the gibbon genome reference, which is based on data from an NLE individual.

In Capozzi et al.[28], we were surprised to find an association between breakpoints and genes, given that most genome breaks will have deleterious effects if they occur in coding or regulatory sequence. However, we noted that those results were based on analyzing the features of the human genome reference that are orthologous to the gibbon breakpoint regions, and that the result might change when the full gibbon genome sequence was available to be tested. Indeed, that proved to be the case, as the depletion for genes in the breakpoints from the gibbon reference data shows.

Our discovery that gibbon genome breakpoints are enriched for CTCF binding sites

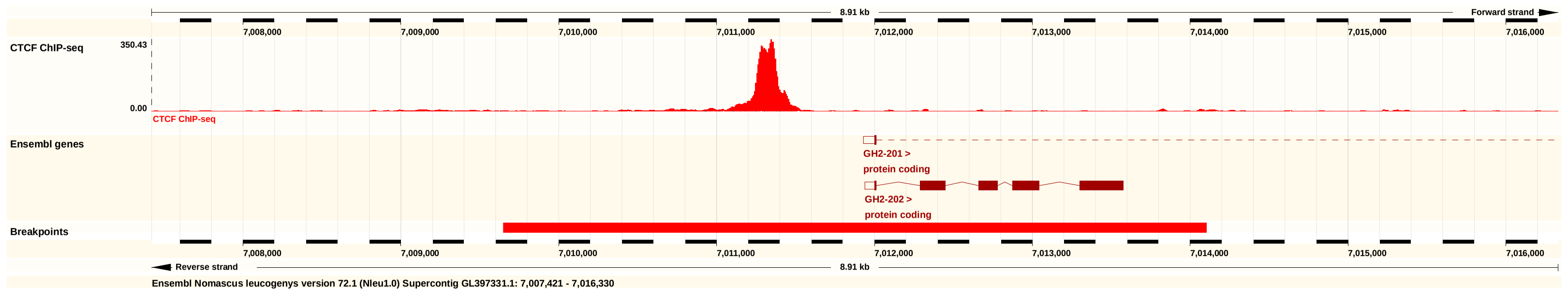

Figure 8.5: An example of CTCF binding sites in a breakpoint region. The track labeled "CTCF ChIP-seq" shows merged read coverage depth from the eight gibbon CTCF ChIP-seq samples, clearly showing a peak representing a CTCF binding site. The "Breakpoints" track shows the location of a gibbon breakpoint region. This breakpoint also contains a protein-coding gene, GH2-202.

indicates that there is a connection between regions with similar regulatory environments and the genome rearrangements preserved in the gibbon family, and the fact that breakpoints are only enriched for CTCF binding sites shared with gibbon's common ancestors suggests that the heavy rearrangement of the gibbon genome may have had to respect existing regulatory domains; breakages in other locations may have been too deleterious to survival to be preserved through evolution. In addition, the interaction between genomic structural variation and the three-dimensional structure of DNA in the cell is just beginning to be explored [48], but it is interesting to hypothesize that locations where the three-dimensional conformation of chromatin in the cell change, as represented by CTCF binding sites, might be vulnerable to genomic breakage under certain circumstances that may have been present in the gibbon's evolutionary past.

Chapter 9

Future Work

In this dissertation we have demonstrated that distributed computing, in the form of Hadoop and MapReduce, can be applied to the SV detection problem, enabling highly accurate algorithms with very fast runtimes. In addition, we showed that the use of discriminative sequence labeling techniques, in the form of conditional random fields, can be used to integrate many different signals of SVs and by doing so can improve Cloudbreak's results. We hope that the work presented in this dissertation is not only useful in itself but is extensible enough to contribute to additional innovations in the detection of structural variations.

9.1 Cloudbreak

There are many possible extensions that could be made to enhance Cloudbreak's effectiveness as a general SV analysis tool for high-throughput sequencing data. These include:

- *Support the detection of additional SV types such as longer deletions, inversions, and translocations.* These would be useful and necessary additions to a complete variant detection pipeline. We hope that the abstract formulation of our MapReduce framework in Chapter 4 will create a base from which to extend Cloudbreak's capabilities and implement many additional SV detection algorithms in the future. For example, detection of small to medium-sized inversions could be realized by adding the orientation of mapped read pairs as an additional component of the mixtures used in Cloudbreak's feature generation function. One difficulty in this effort will be the relative lack of validated SVs that are not deletions and inversion to develop and test against, but as large scale projects such as the 1000 Genomes continue to generate data this concern should be mitigated.

- *Extend Cloudbreak's GMM-based feature generation function to relax the assumption of a diploid genome sample.* Currently Cloudbreak forces a GMM with two components. This assumption would not apply for some sequencing projects, especially in cancer-related research. In a heterogeneous tumor sample, for example, there may be multiple tumor sub-clones, as well as normal DNA, in the sequenced sample, and each set of input DNA might have different genotypes. With additional modeling of these mixtures, it might be possible for Cloudbreak not only to detect different variants in a sample but also to estimate their relative abundances, something that would be useful for tracking tumor clone evolution.

- *Add a local assembly step to increase breakpoint resolution and validate SV candidates.* Cloudbreak's breakpoint resolution is lower than that of many other RP algorithms. Although we improved that result in Chapter 7, we could not obtain single nucleotide resolution, because our features were defined in terms of fixed-size genomic windows. As we mentioned in Chapter 3, one approach to improving breakpoint resolution, and providing additional validation of predicted results, is to attempt to conduct a *de novo* assembly of the reads that mapped near the breakpoints as well as their paired sequences. If effectively implemented, this would be a very useful addition to any SV detection algorithm. Theoretically, each breakpoint could be assembled simultaneously in reduce tasks in an additional MapReduce job. This would greatly increase the confidence and utility of Cloudbreak's predictions.

- *Support the simultaneous analysis of multiple samples and libraries.* Approaches such as Genome STRiP [65] have demonstrated that they can achieve higher accuracy than single-sample SV detection algorithms by simultaneously considering many samples at once. Even if a variant is only supported by a very low number of reads in each individual sample, by pooling data from multiple samples the presence of a variant can become clear. Additional modeling in Cloudbreak of multiple samples, each with its own insert size distribution, might provide a similar increase in power.

Alternatively, data could be pooled in Cloudbreak and the resulting predictions could be used as features in our discriminative machine learning models.

- *Adopt emerging standardized Hadoop libraries and file formats.* As more effort is put into developing Hadoop-based sequencing analysis applications, it will become increasingly important that these applications use consistent APIs and data formats so that they can be composed into large pipelines. Once they contain the requisite features, refactoring Cloudbreak to use libraries such as Hadoop-BAM [138] would be a step in that direction.

9.2 SV Detection with Discriminative Machine Learning

We believe that the application of CRFs to SV detection we described in Chapter 7 is very promising and could be extended in many directions:

- *Construction of improved training data sets.* The most obvious limiting factor for this project was the training and test data used. Training on simulated data ignores many potential sources of noise as well as possible signals that could be used to detect SVs. Training on real data sets, such as those that are being generated for the 1000 Genomes project, would allow much greater potential for learning.

- *Using additional or different sets of labels.* In some sequence labeling tasks, it can improve performance to model the start and end of variants with separate labels, or to use different types of labels for flanking sequence. It would be useful to test different labeling schemes to determine the optimal one for this task. In addition, the authors of forestSV [126] demonstrated some utility in directly labeling regions known to cause false positives for other algorithms in training their classifier; this might be useful in our model as well.

- *Add additional features to the model.* There are many other possible features that could be added to our model. One example would be possible SNVs near the breakpoint locations, indicated by mismatches between the reads and the reference. There are also many other possible genome annotations or sources of prior knowledge that could be used as features, including the locations of previously identified variants in other samples.

- *Learning for additional variant types.* As mentioned above, it would be useful to train our model to detect additional variant types by expanding the label set and set of features used

- *Alternative learning methods.* Recently, many advances have been made in machine learning for sequencing labeling tasks through the use of deep learning using artificial neural networks. With additional data to train on, it may be possible to apply these techniques to the SV detection problem.

9.3 Gibbon Genome Breakpoint Analysis

The identification of enrichment of genomic features at the location of gibbon-human synteny breakpoints can be used as background for further research into the evolutionary history of gibbons, with an ultimate goal of understanding the processes that caused them:

- *Analysis of epigenetic marks associated with breakpoints.* The Carbone lab has been investigating epigenetic signatures near genomic breakpoints, and has found distinct patterns of DNA methylation in breakpoint-associated transposable elements [99]. DNA methylation may have a role in the formation or conservation of rearranged genomes. Further analysis may reveal new mechanisms of gibbon genome rearrangement.

- *Exploration of association of large DNA structures and domains with breakpoints.* The association of CTCF binding sites with gibbon breakpoints suggests a possible relationship between rearrangement sites and the three dimensional structure of DNA. Alternatively, CTCF's role as separator of regulatory domains may have rendered genome rearrangements that did not respect those boundaries unviable, causing only those present in today's gibbons to be preserved. Additional analysis of other types of data may allow us to gain insight into these questions.

Bibliography

[1] 1000 Genomes Project Consortium, Abecasis, G. R., Auton, A., Brooks, L. D., DePristo, M. A., Durbin, R. M., Handsaker, R. E., Kang, H. M., Marth, G. T., and McVean, G. A. An integrated map of genetic variation from 1,092 human genomes. *Nature 491*, 7422 (Nov. 2012), 56–65.

[2] Abyzov, A., Urban, A. E., Snyder, M., and Gerstein, M. CNVnator: an approach to discover, genotype, and characterize typical and atypical CNVs from family and population genome sequencing. *Genome Research 21*, 6 (June 2011), 974–984.

[3] Afgan, E., Baker, D., Coraor, N., Chapman, B., Nekrutenko, A., and Taylor, J. Galaxy CloudMan: delivering cloud compute clusters. *BMC Bioinformatics 11*, Suppl 12 (2010), S4.

[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 (Oct. 2009), 1061–1067.

[6] Alkan, C., Sajjadian, S., and Eichler, E. E. Limitations of next-generation genome sequence assembly. *Nature Methods 8*, 1 (Jan. 2011), 61–65.

[7] Andrew, G., and Bilmes, J. Sequential deep belief networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on* (2012), pp. 4265–4268.

[8] Andrews, S. FastQC: A Quality Control tool for High Throughput Sequence Data. `http://www.bioinformatics.babraham.ac.uk/projects/fastqc/`.

[9] Angiuoli, S. V., Matalka, M., Gussman, A., Galens, K., Vangala, M., Riley, D. R., Arze, C., White, J. R., White, O., and Fricke, W. F. CloVR: A virtual machine for automated and portable sequence analysis from the desktop using cloud computing. *BMC Bioinformatics 12*, 1 (2011), 356.

[10] Annala, M. J., Parker, B. C., Zhang, W., and Nykter, M. Fusion genes and their discovery using high throughput sequencing. *Cancer Letters 340*, 2 (Nov. 2013), 192–200.

[11] Apache Software Foundation. Hbase. `http://hbase.apache.org`.

[12] Apache Software Foundation. Pig. `http://pig.apache.org`.

[13] Apache Software Foundation. Whirr. `http://whirr.apache.org`.

[14] Aten, E., White, S. J., Kalf, M. E., Vossen, R. H. A. M., Thygesen, H. H., Ruivenkamp, C. A., Kriek, M., Breuning, M. H. B., and den Dunnen, J. T. Methods to detect CNVs in the human genome. *Cytogenetic and Genome Research 123*, 1-4 (2008), 313–321.

[15] Bailey, J. A., Baertsch, R., Kent, W. J., Haussler, D., and Eichler, E. E. Hotspots of mammalian chromosomal evolution. *Genome Biology 5*, 4 (2004), R23.

[16] Bailey, J. A., and Eichler, E. E. Primate segmental duplications: crucibles of evolution, diversity and disease. *Nature Reviews Genetics 7*, 7 (July 2006), 552–564.

[17] Bailey, J. A., Gu, Z., Clark, R. A., Reinert, K., Samonte, R. V., Schwartz, S., Adams, M. D., Myers, E. W., Li, P. W., and Eichler, E. E. Recent segmental duplications in the human genome. *Science 297*, 5583 (Aug. 2002), 1003–1007.

[18] Barnett, D. W., Garrison, E. K., Quinlan, A. R., Stromberg, M. P., and Marth, G. T. BamTools: a C++ API and toolkit for analyzing and managing BAM files. *Bioinformatics 27*, 12 (June 2011), 1691–1692.

[19] Benjamini, Y., and Speed, T. P. Summarizing and correcting the GC content bias in high-throughput sequencing. *Nucleic Acids Research 40*, 10 (May 2012), e72.

[20] Benjamini, Y., and Yekutieli, D. The control of the false discovery rate in multiple testing under dependency. *Annals of Statistics* (2001), 1165–1188.

[21] Benko, S., Fantes, J. A., Amiel, J., Kleinjan, D.-J., Thomas, S., Ramsay, J., Jamshidi, N., Essafi, A., Heaney, S., Gordon, C. T., McBride, D., Golzio, C., Fisher, M., Perry, P., Abadie, V., Ayuso, C., Holder-Espinasse, M., Kilpatrick, N., Lees, M. M., Picard, A., Temple, I. K., Thomas, P., Vazquez, M.-P., Vekemans, M., Roest Crollius, H., Hastie, N. D., Munnich, A., Etchevers, H. C., Pelet, A., Farlie, P. G., Fitzpatrick, D. R., and Lyonnet, S. Highly conserved non-coding elements on either side of SOX9 associated with Pierre Robin sequence. *Nature Genetics 41*, 3 (Mar. 2009), 359–364.

[22] Benson, G. Tandem repeats finder: a program to analyze dna sequences. *Nucleic Acids Research 27*, 2 (1999), 573–580.

[23] Breiman, L. Random forests. *Machine Learning 45*, 1 (2001), 5–32.

[24] Broad Institute. Picard tools. `http://picard.sourceforge.net`.

[25] Burrows, M., and Wheeler, D. J. A block-sorting lossless data compression algorithm. Tech. rep., Digital SRC Research Report, 1994.

[26] Cahan, P., Li, Y., Izumi, M., and Graubert, T. A. The impact of copy number variation on local gene expression in mouse hematopoietic stem and progenitor cells. *Nature Genetics 41*, 4 (Apr. 2009), 430–437.

[27] Campbell, P. J., Stephens, P. J., Pleasance, E. D., O'meara, S., Li, H., Santarius, T., Stebbings, L. A., Leroy, C., Edkins, S., Hardy, C., Teague, J. W., Menzies, A., Goodhead, I., Turner, D. J., Clee, C. M., Quail, M. A., Cox, A., Brown, C., Durbin, R., Hurles, M. E., Edwards, P. A. W., Bignell, G. R., Stratton, M. R., and Futreal, P. A. Identification of somatically acquired rearrangements in cancer using genome-wide massively parallel paired-end sequencing. *Nature Genetics 40*, 6 (June 2008), 722–729.

[28] Capozzi, O., Carbone, L., Stanyon, R. R., Marra, A., Yang, F., Whelan, C. W., de Jong, P. J., Rocchi, M., and Archidiacono, N. A comprehensive molecular cytogenetic analysis of chromosome rearrangements in gibbons. *Genome Research* (Aug. 2012).

[29] Carbone, L., Harris, R. A., Vessere, G. M., Mootnick, A. R., Humphray, S., Rogers, J., Kim, S. K., Wall, J. D., Martin, D., Jurka, J., Milosavljevic, A., and de Jong, P. J. Evolutionary Breakpoints in the Gibbon Suggest

Association between Cytosine Methylation and Karyotype Evolution. *PLoS Genetics 5*, 6 (June 2009), e1000538.

[30] Carbone, L., Vessere, G. M., ten Hallers, B. F. H., Zhu, B., Osoegawa, K., Mootnick, A., Kofler, A., Wienberg, J., Rogers, J., Humphray, S., Scott, C., Harris, R. A., Milosavljevic, A., and de Jong, P. J. A high-resolution map of synteny disruptions in gibbon and human genomes. *PLoS Genetics 2*, 12 (Dec. 2006), e223.

[31] Chan, Y. F., Marks, M. E., Jones, F. C., Villarreal, G., Shapiro, M. D., Brady, S. D., Southwick, A. M., Absher, D. M., Grimwood, J., Schmutz, J., Myers, R. M., Petrov, D., Jonsson, B., Schluter, D., Bell, M. A., and Kingsley, D. M. Adaptive Evolution of Pelvic Reduction in Sticklebacks by Recurrent Deletion of a Pitx1 Enhancer. *Science 327*, 5963 (Jan. 2010), 302–305.

[32] Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A., and Gruber, R. E. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst. 26*, 2 (June 2008), 4:1–4:26.

[33] Chang, Y.-J., Chen, C.-C., Chen, C.-L., and Ho, J.-M. A de novo next generation genomic sequence assembler based on string graph and MapReduce cloud computing framework. *BMC Genomics 13 Suppl 7* (2012), S28.

[34] Chang, Y.-J., Chen, C.-C., Ho, J.-M., and Chen, C.-L. De novo assembly of high-throughput sequencing data with cloud computing and new operations on string graphs. *2012 IEEE Fifth International Conference on Cloud Computing* (2012), 155–161.

[35] Chen, J., Lu, Y., Xu, J., Huang, Y., Cheng, H., Hu, G., Luo, C., Lou, M., Cao, G., Xie, Y., and Ying, K. Identification and characterization of NBEAL1, a novel human neurobeachin-like 1 protein gene from fetal brain, which is up regulated in glioma. *Molecular Brain Research 125*, 1-2 (June 2004), 147–155.

[36] Chen, K., Chen, L., Fan, X., Wallis, J., Ding, L., and Weinstock, G. TIGRA: A Targeted Iterative Graph Routing Assembler for breakpoint assembly. *Genome Research* (Dec. 2013).

[37] 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.

[38] Chiang, D. Y., Getz, G., Jaffe, D. B., O'Kelly, M. J. T., Zhao, X., Carter, S. L., Russ, C., Nusbaum, C., Meyerson, M., and Lander, E. S. High-resolution mapping of copy-number alterations with massively parallel sequencing. *Nature Methods 6*, 1 (Jan. 2009), 99–103.

[39] Chiara, M., Pesole, G., and Horner, D. S. SVM^2: an improved paired-end-based tool for the detection of small genomic structural variations using high-throughput single-genome resequencing data. *Nucleic Acids Research 40*, 18 (Oct. 2012), e145.

[40] Cock, P. J. A., Fields, C. J., Goto, N., Heuer, M. L., and Rice, P. M. The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic Acids Research 38*, 6 (Apr. 2010), 1767–1771.

[41] Conrad, D. F., Bird, C., Blackburne, B., Lindsay, S., Mamanova, L., Lee, C., Turner, D. J., and Hurles, M. E. Mutation spectrum revealed by breakpoint sequencing of human germline CNVs. *Nature Genetics 42*, 5 (May 2010), 385–391.

[42] Conrad, D. F., Pinto, D., Redon, R., Feuk, L., Gokcumen, O., Zhang, Y., Aerts, J., Andrews, T. D., Barnes, C., Campbell, P., Fitzgerald, T., Hu, M., Ihm, C. H., Kristiansson, K., MacArthur, D. G., MacDonald, J. R., Onyiah, I., Pang, A. W. C., Robson, S., Stirrups, K., Valsesia, A., Walter, K., Wei, J., Wellcome Trust Case Control Consortium, Tyler-Smith, C., Carter, N. P., Lee, C., Scherer, S. W., and Hurles, M. E. Origins and functional impact of copy number variation in the human genome. *Nature 464*, 7289 (Apr. 2010), 704–712.

[43] Dale, R. K., Pedersen, B. S., and Quinlan, A. R. Pybedtools: a flexible Python library for manipulating genomic datasets and annotations. *Bioinformatics 27*, 24 (Dec. 2011), 3423–3424.

[44] Danecek, P., Auton, A., Abecasis, G., Albers, C. A., Banks, E., DePristo, M. A., Handsaker, R. E., Lunter, G., Marth, G. T., Sherry, S. T., McVean, G., Durbin, R., and 1000 Genomes Project Analysis Group. The variant call format and VCFtools. *Bioinformatics 27*, 15 (Aug. 2011), 2156–2158.

[45] Dean, J., and Ghemawat, S. MapReduce: Simplified data processing on large clusters. *Communications of the ACM 51*, 1 (2008), 107–113.

[46] 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*, 5 (Apr. 2011), 491–498.

[47] Ding, M., Zheng, L., Lu, Y., Li, L., Guo, S., and Guo, M. More convenient more overhead: The performance evaluation of hadoop streaming. In *Proceedings of the 2011 ACM Symposium on Research in Applied Computation* (New York, NY, USA, 2011), RACS '11, ACM, pp. 307–313.

[48] Dixon, J. R., Selvaraj, S., Yue, F., Kim, A., Li, Y., Shen, Y., Hu, M., Liu, J. S., and Ren, B. Topological domains in mammalian genomes identified by analysis of chromatin interactions. *NATURE 485*, 7398 (Apr. 2012), 376–380.

[49] Duchi, J., Hazan, E., and Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research 12* (July 2011), 2121–2159.

[50] Eilers, M., and Eisenman, R. N. Myc's broad reach. *Genes and Development 22*, 20 (Oct. 2008), 2755–2766.

[51] Emde, A.-K., Schulz, M. H., Weese, D., Sun, R., Vingron, M., Kalscheuer, V. M., Haas, S. A., and Reinert, K. Detecting genomic indel variants with exact breakpoints in single- and paired-end sequencing data using SplazerS. *Bioinformatics 28*, 5 (Mar. 2012), 619–627.

[52] ENCODE Project Consortium. A user's guide to the encyclopedia of DNA elements (ENCODE). *PLoS Biology 9*, 4 (Apr. 2011), e1001046.

[53] ENSEMBL. EnsemblCompara. `http://uswest.ensembl.org/info/docs/api/compara/index.html`.

[54] Escaramis, G., Tornador, C., Bassaganyas, L., Rabionet, R., Tubío, J. M. C., Martínez-Fundichely, A., Cáceres, M., Gut, M., Ossowski, S., and Estivill, X. PeSV-Fisher: Identification of Somatic and Non-Somatic Structural Variants Using Next Generation Sequencing Data. *PLoS ONE 8*, 5 (May 2013), e63377.

[55] Escot, C., Theillet, C., Lidereau, R., Spyratos, F., Champeme, M. H., Gest, J., and Callahan, R. Genetic alteration of the c-myc protooncogene

(MYC) in human primary breast carcinomas. *Proceedings of the National Academy of Sciences of the United States of America 83*, 13 (July 1986), 4834–4838.

[56] Evani, U. S., Challis, D., Yu, J., Jackson, A. R., Paithankar, S., Bainbridge, M. N., Jakkamsetti, A., Pham, P., Coarfa, C., Milosavljevic, A., and Yu, F. Atlas2 Cloud: a framework for personal genome analysis in the cloud. *BMC Genomics 13 Suppl 6* (2012), S19.

[57] Feng, X., Grossman, R., and Stein, L. PeakRanger: A cloud-enabled peak caller for ChIP-seq data. *BMC Bioinformatics 12*, 1 (2011), 139.

[58] Ferragina, P., and Manzini, G. Opportunistic data structures with applications. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on* (2000), pp. 390–398.

[59] Fischer, M., Snajder, R., Pabinger, S., Dander, A., Schossig, A., Zschocke, J., Trajanoski, Z., and Stocker, G. SIMPLEX: cloud-enabled pipeline for the comprehensive analysis of exome sequencing data. *PLoS ONE 7*, 8 (2012), e41948.

[60] Ghemawat, S., Gobioff, H., and Leung, S.-T. The google file system. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles* (New York, NY, USA, 2003), SOSP '03, ACM, pp. 29–43.

[61] Giardine, B., Riemer, C., Hardison, R. C., Burhans, R., Elnitski, L., Shah, P., Zhang, Y., Blankenberg, D., Albert, I., Taylor, J., Miller, W., Kent, W. J., and Nekrutenko, A. Galaxy: a platform for interactive large-scale genome analysis. *Genome Research 15*, 10 (Oct. 2005), 1451–1455.

[62] Girirajan, S., Chen, L., Graves, T., Marques-Bonet, T., Ventura, M., Fronick, C., Fulton, L., Rocchi, M., Fulton, R. S., Wilson, R. K., Mardis, E. R., and Eichler, E. E. Sequencing human-gibbon breakpoints of synteny reveals mosaic new insertions at rearrangement sites. *Genome Research 19*, 2 (Feb. 2009), 178–190.

[63] Google, Inc. Snappy. `https://code.google.com/p/snappy/`.

[64] Hampton, O. A., Den Hollander, P., Miller, C. A., Delgado, D. A., Li, J., Coarfa, C., Harris, R. A., Richards, S., Scherer, S. E., and Muzny, D. M. A sequence-level map of chromosomal breakpoints in the MCF-7 breast cancer cell line yields insights into the evolution of a cancer genome. *Genome Research 19*, 2 (2009), 167–177.

[65] Handsaker, R. E., Korn, J. M., Nemesh, J., and McCarroll, S. A. Discovery and genotyping of genome structural polymorphism by sequencing on a population scale. *Nature Genetics 43*, 3 (Mar. 2011), 269–276.

[66] Hart, S. N., Sarangi, V., Moore, R., Baheti, S., Bhavsar, J. D., Couch, F. J., and Kocher, J.-P. A. SoftSearch: Integration of Multiple Sequence Features to Identify Breakpoints of Structural Variations. *PLoS ONE 8*, 12 (2013), e83356.

[67] Hastings, P. J., Lupski, J. R., Rosenberg, S. M., and Ira, G. Mechanisms of change in gene copy number. *Nature Reviews Genetics 10*, 8 (Aug. 2009), 551–564.

[68] Hayes, M., and Li, J. Bellerophon: a hybrid method for detecting interchromosomal rearrangements at base pair resolution using next-generation sequencing data. *BMC Bioinformatics 14 Suppl 5* (2013), S6.

[69] Hayes, M., Pyon, Y. S., and Li, J. A model-based clustering method for genomic structural variant prediction and genotyping using paired-end sequencing data. *PLoS ONE 7*, 12 (Dec. 2012), e52881.

[70] Homer, N. DNAA. `http://sourceforge.net/apps/mediawiki/dnaa/`.

[71] Hong, D., Rhie, A., Park, S.-S., Lee, J., Ju, Y. S., Kim, S., Yu, S.-B., Bleazard, T., Park, H.-S., and Rhee, H. FX: an RNA-Seq analysis tool on the cloud. *Bioinformatics 28*, 5 (2012), 721–723.

[72] Hormozdiari, F., Alkan, C., Eichler, E. E., and Sahinalp, S. C. Combinatorial algorithms for structural variation detection in high-throughput sequenced genomes. *Genome Research 19*, 7 (July 2009), 1270–1278.

[73] Hosokawa, Y., Suzuki, R., Joh, T., Maeda, Y., Nakamura, S., Kodera, Y., Arnold, A., and Seto, M. A small deletion in the 3'-untranslated region of the cyclin D1/PRAD1/bcl-1 oncogene in a patient with chronic lymphocytic leukemia. *International journal of cancer 76*, 6 (June 1998), 791–796.

[74] Huang, H., Tata, S., and Prill, R. J. BlueSNP: R package for highly scalable genome-wide association studies using Hadoop clusters. *Bioinformatics 29*, 1 (Nov. 2012), 135–136.

[75] Iqbal, Z., Caccamo, M., Turner, I., Flicek, P., and McVean, G. De novo assembly and genotyping of variants using colored de Bruijn graphs. *Nature Genetics* (Jan. 2012).

[76] Issa, S. A., Kienzler, R., El-Kalioby, M., Tonellato, P. J., Wall, D., Bruggmann, R., and Abouelhoda, M. Streaming support for data intensive cloud-based sequence analysis. *BioMed Research International 2013* (2013), 791051.

[77] Jiang, Y., Wang, Y., and Brudno, M. PRISM: pair-read informed split-read mapping for base-pair level detection of insertion, deletion and structural variants. *Bioinformatics 28*, 20 (Oct. 2012), 2576–2583.

[78] Jourdren, L., Bernard, M., Dillies, M.-A., and Le Crom, S. Eoulsan: A Cloud Computing-Based Framework Facilitating High Throughput Sequencing Analyses. *Bioinformatics* (Apr. 2012).

[79] Kelley, D. R., Schatz, M. C., and Salzberg, S. L. Quake: quality-aware detection and correction of sequencing errors. *Genome Biology 11*, 11 (2010), R116.

[80] Kent, W. J. BLAT: The BLAST-Like Alignment Tool. *Genome Research 12*, 4 (2002), 656–664.

[81] keun Kim, D., hee Yoon, J., hwa Kong, J., kyun Hong, S., and joo Lee, U. Cloud-scale snp detection from rna-seq data. In *Data Mining and Intelligent Information Technology Applications (ICMiA), 2011 3rd International Conference on* (2011), pp. 321–323.

[82] 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., Tüzün, 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.

[83] Koboldt, D. C., Larson, D. E., Chen, K., Ding, L., and Wilson, R. K. Massively parallel sequencing approaches for characterization of structural variation. *Methods in Molecular Biology 838* (2012), 369–384.

[84] Korbel, J. O., Abyzov, A., Mu, X. J., Carriero, N., Cayting, P., Zhang, Z., Snyder, M., and Gerstein, M. B. PEMer: a computational framework with

simulation-based error models for inferring genomic structural variants from massive paired-end sequencing data. *Genome Biology 10*, 2 (2009), R23.

[85] 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. E., 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 (Oct. 2007), 420–426.

[86] Krampis, K., Booth, T., Chapman, B., Tiwari, B., Bicak, M., Field, D., and Nelson, K. E. Cloud BioLinux: pre-configured and on-demand bioinformatics computing for the genomics community. *BMC Bioinformatics 13*, 1 (2012), 42.

[87] Kurzrock, R., Kantarjian, H. M., Druker, B. J., and Talpaz, M. Philadelphia chromosome–positive leukemias: from basic mechanisms to molecular therapeutics. *Annals of Internal Medicine 138*, 10 (2003), 819–830.

[88] Lafferty, J. D., McCallum, A., and Pereira, F. C. N. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning* (San Francisco, CA, USA, 2001), ICML '01, Morgan Kaufmann Publishers Inc., pp. 282–289.

[89] Lam, H. Y. K., Mu, X. J., Stütz, A. M., Tanzer, A., Cayting, P. D., Snyder, M., Kim, P. M., Korbel, J. O., and Gerstein, M. B. Nucleotide-resolution analysis of structural variants using BreakSeq and a breakpoint library. *Nature Biotechnology 28*, 1 (2010), 47–55.

[90] Lam, H. Y. K., Pan, C., Clark, M. J., Lacroute, P., Chen, R., Haraksingh, R., O'Huallachain, M., Gerstein, M. B., Kidd, J. M., Bustamante, C. D., and Snyder, M. Detecting and annotating genetic variations using the HugeSeq pipeline. *Nature Biotechnology 30*, 3 (Mar. 2012), 226–229.

[91] Lämmel, R. Google's MapReduce programming model – Revisited. *Science of Computer Programming 70*, 1 (2008), 1 – 30.

[92] Landry, J. J. M., Pyl, P. T., Rausch, T., Zichner, T., Tekkedil, M. M., Stütz, A. M., Jauch, A., Aiyar, R. S., Pau, G., Delhomme, N., Gagneur, J., Korbel, J. O., Huber, W., and Steinmetz, L. M. The genomic and transcriptomic landscape of a HeLa cell line. *G3 (Bethesda, Md.) 3*, 8 (Aug. 2013), 1213–1224.

[93] Langmead, B., Hansen, K. D., and Leek, J. T. Cloud-scale RNA-sequencing differential expression analysis with Myrna. *Genome Biology 11*, 8 (2010), R83.

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

[95] Langmead, B., Schatz, M. C., Lin, J., Pop, M., and Salzberg, S. L. Searching for SNPs with cloud computing. *Genome Biology 10*, 11 (2009), R134.

[96] 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.

[97] Larkin, D. M., Pape, G., Donthu, R., Auvil, L., Welge, M., and Lewin, H. A. Breakpoint regions and homologous synteny blocks in chromosomes have different evolutionary histories. *Genome Research 19*, 5 (May 2009), 770–777.

[98] Layer, R. M., Hall, I. M., and Quinlan, A. R. LUMPY: A probabilistic framework for structural variant discovery. *ArXiv e-prints* (Oct. 2012).

[99] Lazar, N. H., Wilhelm, L. J., Terhune, E., Meyer, T. J., Whelan, C. W., Sönmez, K., and Carbone, L. Gibbon-human synteny breakpoints carry a distinctive epigenetic signature. *Genome Research* (under review).

[100] Lee, H., and Schatz, M. C. Genomic dark matter: the reliability of short read mapping illustrated by the genome mappability score. *Bioinformatics 28*, 16 (Aug. 2012), 2097–2105.

[101] Lee, J. A., Carvalho, C. M. B., and Lupski, J. R. A DNA replication mechanism for generating nonrecurrent rearrangements associated with genomic disorders. *Cell 131*, 7 (Dec. 2007), 1235–1247.

[102] Lee, S., Hormozdiari, F., Alkan, C., and Brudno, M. MoDIL: detecting small indels from clone-end sequencing with mixtures of distributions. *Nature Methods 6*, 7 (July 2009), 473–474.

[103] Lee, W.-P., Stromberg, M., Ward, A., Stewart, C., Garrison, E., and Marth, G. T. MOSAIK: A hash-based algorithm for accurate next-generation sequencing read mapping. *ArXiv e-prints* (Sept. 2013).

[104] Lehrman, M. A., Schneider, W. J., Südhof, T. C., Brown, M. S., Goldstein, J. L., and Russell, D. W. Mutation in LDL receptor: Alu-Alu recombination deletes exons encoding transmembrane and cytoplasmic domains. *Science 227*, 4683 (Jan. 1985), 140–146.

[105] Levy, S., Sutton, G., Ng, P. C., Feuk, L., Halpern, A. L., Walenz, B. P., Axelrod, N., Huang, J., Kirkness, E. F., Denisov, G., Lin, Y., MacDonald, J. R., Pang, A. W. C., Shago, M., Stockwell, T. B., Tsiamouri, A., Bafna, V., Bansal, V., Kravitz, S. A., Busam, D. A., Beeson, K. Y., McIntosh, T. C., Remington, K. A., Abril, J. F., Gill, J., Borman, J., Rogers, Y.-H., Frazier, M. E., Scherer, S. W., Strausberg, R. L., and Venter, J. C. The diploid genome sequence of an individual human. *PLoS Biology 5*, 10 (Sept. 2007), e254.

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

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

[108] Li, H., and Homer, N. A survey of sequence alignment algorithms for next-generation sequencing. *Briefings in Bioinformatics 11*, 5 (Sept. 2010), 473–483.

[109] Li, R., Li, Y., Fang, X., Yang, H., Wang, J., Kristiansen, K., and Wang, J. SNP detection for massively parallel whole-genome resequencing. *Genome Research 19*, 6 (June 2009), 1124–1132.

[110] Li, Y., Zheng, H., Luo, R., Wu, H., Zhu, H., Li, R., Cao, H., Wu, B., Huang, S., Shao, H., Ma, H., Zhang, F., Feng, S., Zhang, W., Du, H., Tian, G., Li, J., Zhang, X., Li, S., Bolund, L., Kristiansen, K., De Smith, A. J., Blakemore, A. I. F., Coin, L. J. M., Yang, H., Wang, J., and Wang, J. Structural variation in two human genomes mapped at single-nucleotide resolution by whole genome de novo assembly. *Nature Biotechnology 29*, 8 (July 2011), 723–730.

[111] Liskay, R. M., Letsou, A., and Stachelek, J. L. Homology requirement for efficient gene conversion between duplicated chromosomal sequences in mammalian cells. *Genetics 115*, 1 (Jan. 1987), 161–167.

[112] Liu, P., Carvalho, C. M. B., Hastings, P. J., and Lupski, J. R. Mechanisms for recurrent and complex human genomic rearrangements. *Current Opinion in Genetics & Development 22*, 3 (June 2012), 211–220.

[113] Liu, P., Erez, A., Nagamani, S. C. S., Dhar, S. U., Kołodziejska, K. E., Dharmadhikari, A. V., Cooper, M. L., Wiszniewska, J., Zhang, F., Withers, M. A., Bacino, C. A., Campos-Acevedo, L. D., Delgado, M. R., Freedenberg, D., Garnica, A., Grebe, T. A., Hernández-Almaguer, D., Immken, L., Lalani, S. R., McLean, S. D., Northrup, H., Scaglia, F., Strathearn, L., Trapane, P., Kang, S.-H. L., Patel, A., Cheung, S. W., Hastings, P. J., Stankiewicz, P., Lupski, J. R., and Bi, W. Chromosome catastrophes involve replication mechanisms generating complex genomic rearrangements. *Cell 146*, 6 (Sept. 2011), 889–903.

[114] Locke, D. P., Hillier, L. W., Warren, W. C., Worley, K. C., Nazareth, L. V., Muzny, D. M., Yang, S.-P., Wang, Z., Chinwalla, A. T., Minx, P., Mitreva, M., Cook, L., Delehaunty, K. D., Fronick, C., Schmidt, H., Fulton, L. A., Fulton, R. S., Nelson, J. O., Magrini, V., Pohl, C., Graves, T. A., Markovic, C., Cree, A., Dinh, H. H., Hume, J., Kovar, C. L., Fowler, G. R., Lunter, G., Meader, S., Heger, A., Ponting, C. P., Marques-Bonet, T., Alkan, C., Chen, L., Cheng, Z., Kidd, J. M., Eichler, E. E., White, S., Searle, S., Vilella, A. J., Chen, Y., Flicek, P., Ma, J., Raney, B., Suh, B., Burhans, R., Herrero, J., Haussler, D., Faria, R., Fernando, O., Darré, F., Farré, D., Gazave, E., Oliva, M., Navarro, A., Roberto, R., Capozzi, O., Archidiacono, N., Della Valle, G., Purgato, S., Rocchi, M., Konkel, M. K., Walker, J. A., Ullmer, B., Batzer, M. A., Smit, A. F. A., Hubley, R., Casola, C., Schrider, D. R., Hahn, M. W., Quesada, V., Puente, X. S., Ordóñez, G. R., López-Otín, C., Vinar, T., Brejova, B., Ratan, A., Harris, R. S., Miller, W., Kosiol, C., Lawson, H. A., Taliwal, V., Martins, A. L., Siepel, A., RoyChoudhury, A., Ma, X., Degenhardt, J., Bustamante, C. D., Gutenkunst, R. N., Mailund, T., Dutheil, J. Y., Hobolth, A., Schierup, M. H., Ryder, O. A., Yoshinaga, Y., de Jong, P. J., Weinstock, G. M., Rogers, J., Mardis, E. R., Gibbs, R. A., and Wilson, R. K. Comparative and demographic analysis of orang-utan genomes. *NATURE 469*, 7331 (Jan. 2011), 529–533.

[115] Lupski, J. R. Genomic disorders: structural features of the genome can lead to DNA rearrangements and human disease traits. *Trends in Genetics 14*, 10 (Oct. 1998), 417–422.

[116] Malhotra, A., Lindberg, M., Faust, G. G., Leibowitz, M. L., Clark, R. A., Layer, R. M., Quinlan, A. R., and Hall, I. M. Breakpoint profiling of

64 cancer genomes reveals numerous complex rearrangements spawned by homology-independent mechanisms. *Genome Research 23*, 5 (May 2013), 762–776.

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

[118] Marschall, T., Costa, I. G., Canzar, S., Bauer, M., Klau, G. W., Schliep, A., and Schönhuth, A. CLEVER: clique-enumerating variant finder. *Bioinformatics 28*, 22 (Nov. 2012), 2875–2882.

[119] Martin, M. Cutadapt removes adapter sequences from high-throughput sequencing reads. *EMBnet.journal 17*, 1 (2011), 10–12.

[120] Marx, V. Biology: The big challenges of big data. *NATURE 498*, 7453 (June 2013), 255–260.

[121] McCallum, A., Schultz, K., and Singh, S. FACTORIE: Probabilistic programming via imperatively defined factor graphs. In *Neural Information Processing Systems (NIPS)* (2009).

[122] McCarroll, S. A., Huett, A., Kuballa, P., Chilewski, S. D., Landry, A., Goyette, P., Zody, M. C., Hall, J. L., Brant, S. R., Cho, J. H., Duerr, R. H., Silverberg, M. S., Taylor, K. D., Rioux, J. D., Altshuler, D., Daly, M. J., and Xavier, R. J. Deletion polymorphism upstream of IRGM associated with altered IRGM expression and Crohn's disease. *Nature Genetics 40*, 9 (Sept. 2008), 1107–1112.

[123] McCarroll, S. A., Kuruvilla, F. G., Korn, J. M., Cawley, S., Nemesh, J., Wysoker, A., Shapero, M. H., de Bakker, P. I. W., Maller, J. B., Kirby, A., Elliott, A. L., Parkin, M., Hubbell, E., Webster, T., Mei, R., Veitch, J., Collins, P. J., Handsaker, R., Lincoln, S., Nizzari, M., Blume, J., Jones, K. W., Rava, R., Daly, M. J., Gabriel, S. B., and Altshuler, D. Integrated detection and population-genetic analysis of SNPs and copy number variation. *Nature Genetics 40*, 10 (Oct. 2008), 1166–1174.

[124] McKenna, A., Hanna, M., Banks, E., Sivachenko, A., Cibulskis, K., Kernytsky, A., Garimella, K., Altshuler, D., Gabriel, S., Daly, M., and DePristo, M. A. The Genome Analysis Toolkit: A MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Research 20*, 9 (Sept. 2010), 1297–1303.

[125] Medvedev, P., Fiume, M., Dzamba, M., Smith, T., and Brudno, M. Detecting copy number variation with mated short reads. *Genome Research 20*, 11 (Nov. 2010), 1613–1622.

[126] Michaelson, J. J., and Sebat, J. forestSV: structural variant discovery through statistical learning. *Nature Methods 9*, 8 (Aug. 2012), 819–821.

[127] 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 (June 2011), 830–839.

[128] Mills, R. E., Walter, K., Stewart, C., Handsaker, R. E., Chen, K., Alkan, C., Abyzov, A., Yoon, S. C., Ye, K., Cheetham, R. K., Chinwalla, A., Conrad, D. F., Fu, Y., Grubert, F., Hajirasouliha, I., Hormozdiari, F., Iakoucheva, L. M., Iqbal, Z., Kang, S., Kidd, J. M., Konkel, M. K., Korn, J., Khurana, E., Kural, D., Lam, H. Y. K., Leng, J., Li, R., Li, Y., Lin, C.-Y., Luo, R., Mu, X. J., Nemesh, J., Peckham, H. E., Rausch, T., Scally, A., Shi, X., Stromberg, M. P., Stütz, A. M., Urban, A. E., Walker, J. A., Wu, J., Zhang, Y., Zhang, Z. D., Batzer, M. A., Ding, L., Marth, G. T., McVean, G., Sebat, J., Snyder, M., Wang, J., Ye, K., Eichler, E. E., Gerstein, M. B., Hurles, M. E., Lee, C., McCarroll, S. A., and Korbel, J. O. Mapping copy number variation by population-scale genome sequencing. *Nature 470*, 7332 (Feb. 2011), 59–65.

[129] Mimori, T., Nariai, N., Kojima, K., Takahashi, M., Ono, A., Sato, Y., Yamaguchi-Kabata, Y., and Nagasaki, M. iSVP: an integrated structural variant calling pipeline from high-throughput sequencing data. *BMC Systems Biology 7* (2013), S8.

[130] Misceo, D., Capozzi, O., Roberto, R., Dell'oglio, M. P., Rocchi, M., Stanyon, R., and Archidiacono, N. Tracking the complex flow of chromosome rearrangements from the Hominoidea Ancestor to extant Hylobates and Nomascus Gibbons by high-resolution synteny mapping. *Genome Research 18*, 9 (Sept. 2008), 1530–1537.

[131] Mohamed, N. M., Lin, H., and Feng, W.-c. Accelerating Data-Intensive Genome Analysis in the Cloud. In *5th International Conference on Bioinformatics and Computational Biology (BICoB)* (Honolulu, Hawaii, USA, March 2013).

[132] Murphy, W. J., Larkin, D. M., Everts-van der Wind, A., Bourque, G., Tesler, G., Auvil, L., Beever, J. E., Chowdhary, B. P., Galibert, F., Gatzke, L., Hitte, C., Meyers, S. N., Milan, D., Ostrander, E. A., Pape, G., Parker, H. G., Raudsepp, T., Rogatcheva, M. B., Schook, L. B., Skow, L. C., Welge, M., Womack, J. E., O'brien, S. J., Pevzner, P. A., and Lewin, H. A. Dynamics of mammalian chromosome evolution inferred from multispecies comparative maps. *Science 309*, 5734 (July 2005), 613–617.

[133] Myers, E. W. The fragment assembly string graph. *Bioinformatics 21 Suppl 2* (Sept. 2005), ii79–85.

[134] Nadeau, J. H., and Taylor, B. A. Lengths of chromosomal segments conserved since divergence of man and mouse. *Proceedings of the National Academy of Sciences of the United States of America 81*, 3 (Feb. 1984), 814–818.

[135] Nagarajan, N., and Pop, M. Sequence assembly demystified. *Nature Reviews Genetics 14*, 3 (Jan. 2013), 157–167.

[136] Neph, S., Kuehn, M. S., Reynolds, A. P., Haugen, E., Thurman, R. E., Johnson, A. K., Rynes, E., Maurano, M. T., Vierstra, J., Thomas, S., Sandstrom, R., Humbert, R., and Stamatoyannopoulos, J. A. BEDOPS: high-performance genomic feature operations. *Bioinformatics 28*, 14 (July 2012), 1919–1920.

[137] Nguyen, T., Shi, W., and Ruden, D. CloudAligner: A fast and full-featured MapReduce based tool for sequence mapping. *BMC Research Notes 4* (2011), 171.

[138] Niemenmaa, M., Kallio, A., Schumacher, A., Klemelä, P., Korpelainen, E., and Heljanko, K. Hadoop-BAM: directly manipulating next generation sequencing data in the cloud. *Bioinformatics 28*, 6 (Mar. 2012), 876–877.

[139] Nord, A. S., Lee, M., King, M.-C., and Walsh, T. Accurate and exact CNV identification from targeted high-throughput sequence data. *BMC Genomics 12* (2011), 184.

[140] Nordberg, H., Bhatia, K., Wang, K., and Wang, Z. BioPig: a Hadoop-based analytic toolkit for large-scale sequence data. *Bioinformatics 29*, 23 (Dec. 2013), 3014–3019.

[141] Norton, N., Li, D., Rieder, M. J., Siegfried, J. D., Rampersaud, E., Züchner, S., Mangos, S., Gonzalez-Quintana, J., Wang, L., McGee, S., Reiser,

J., Martin, E., Nickerson, D. A., and Hershberger, R. E. Genome-wide studies of copy number variation and exome sequencing identify rare variants in BAG3 as a cause of dilated cardiomyopathy. *American Journal of Human Genetics 88*, 3 (Mar. 2011), 273–282.

[142] Novocraft, Inc. Novoalign. `http://www.novocraft.com`.

[143] O'connor, B. D., Merriman, B., and Nelson, S. F. SeqWare Query Engine: storing and searching sequence data in the cloud. *BMC Bioinformatics 11*, Suppl 12 (2010), S2.

[144] Ohno, S. Ancient linkage groups and frozen accidents. *NATURE 244*, 5414 (Aug. 1973), 259–262.

[145] Open Cloud Consortium. Bionimbus. `http://bionimbus.opensciencedatacloud.org/`.

[146] Oshlack, A., Robinson, M. D., and Young, M. D. From RNA-seq reads to differential expression results. *Genome Biology 11*, 12 (2010), 220.

[147] Park, P. J. ChIP-seq: advantages and challenges of a maturing technology. *Nature Reviews Genetics 10*, 10 (Oct. 2009), 669–680.

[148] Pevzner, P., and Tesler, G. Human and mouse genomic sequences reveal extensive breakpoint reuse in mammalian evolution. *Proceedings of the National Academy of Sciences of the United States of America 100*, 13 (June 2003), 7672–7677.

[149] Phillips, J. E., and Corces, V. G. CTCF: master weaver of the genome. *Cell 137*, 7 (June 2009), 1194–1211.

[150] Pireddu, L., Leo, S., and Zanetti, G. Mapreducing a genomic sequencing workflow. In *Proceedings of the Second International Workshop on MapReduce and Its Applications* (New York, NY, USA, 2011), MapReduce '11, ACM, pp. 67–74.

[151] Pireddu, L., Leo, S., and Zanetti, G. SEAL: a distributed short read mapping and duplicate removal tool. *Bioinformatics 27*, 15 (July 2011), 2159–2160.

[152] Pleasance, E. D., Cheetham, R. K., Stephens, P. J., Mcbride, D. J., Humphray, S. J., Greenman, C. D., Varela, I., Lin, M.-L., Ordóñez, G. R., Bignell, G. R., Ye, K., Alipaz, J., Bauer, M. J., Beare, D., Butler, A., Carter, R. J., Chen, L., Cox, A. J., Edkins, S., Kokko-Gonzales, P. I., Gormley, N. A., Grocock, R. J., Haudenschild, C. D., Hims, M. M.,

James, T., Jia, M., Kingsbury, Z., Leroy, C., Marshall, J., Menzies, A., Mudie, L. J., Ning, Z., Royce, T., Schulz-Trieglaff, O. B., Spiridou, A., Stebbings, L. A., Szajkowski, L., Teague, J., Williamson, D., Chin, L., Ross, M. T., Campbell, P. J., Bentley, D. R., Futreal, P. A., and Stratton, M. R. A comprehensive catalogue of somatic mutations from a human cancer genome. *NATURE 463*, 7278 (Jan. 2010), 191–196.

[153] Qi, J., and Zhao, F. inGAP-sv: a novel scheme to identify and visualize structural variation from paired end mapping data. *Nucleic Acids Research 39*, Web Server issue (July 2011), W567–75.

[154] Quinlan, A. R., Clark, R. A., Sokolova, S., Leibowitz, M. L., Zhang, Y., Hurles, M. E., Mell, J. C., and Hall, I. M. Genome-wide mapping and assembly of structural variant breakpoints in the mouse genome. *Genome Research 20*, 5 (May 2010), 623–635.

[155] Quinlan, A. R., and Hall, I. M. BEDTools: a flexible suite of utilities for comparing genomic features. *Bioinformatics 26*, 6 (Mar. 2010), 841–842.

[156] Raphael, B. J., Volik, S., Collins, C., and Pevzner, P. A. Reconstructing tumor genome architectures. *Bioinformatics 19 Suppl 2* (Oct. 2003), ii162–71.

[157] Rausch, T., Zichner, T., Schlattl, A., Stütz, A. M., Benes, V., and Korbel, J. O. DELLY: structural variant discovery by integrated paired-end and split-read analysis. *Bioinformatics 28*, 18 (Sept. 2012), i333–i339.

[158] Robberecht, C., Voet, T., Zamani Esteki, M., Nowakowska, B. A., and Vermeesch, J. R. Nonallelic homologous recombination between retrotransposable elements is a driver of de novo unbalanced translocations. *Genome Research 23*, 3 (Mar. 2013), 411–418.

[159] Roberto, R., Capozzi, O., Wilson, R. K., Mardis, E. R., Lomiento, M., Tüzün, E., Cheng, Z., Mootnick, A. R., Archidiacono, N., Rocchi, M., and Eichler, E. E. Molecular refinement of gibbon genome rearrangements. *Genome Research 17*, 2 (Feb. 2007), 249–257.

[160] Rubinstein, R. Y., and Kroese, D. P. *Simulation and the Monte Carlo Method (Wiley Series in Probability and Statistics)*, 2 ed. Wiley, 2007.

[161] Ruffalo, M., Laframboise, T., and Koyutürk, M. Comparative analysis of algorithms for next-generation sequencing read alignment. *Bioinformatics* (Aug. 2011).

[162] 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*, 9 (Sept. 2010), 647–657.

[163] Schatz, M. C. CloudBurst: highly sensitive read mapping with MapReduce. *Bioinformatics 25*, 11 (June 2009), 1363–1369.

[164] Schatz, M. C., Langmead, B., and Salzberg, S. L. Cloud computing and the DNA data race. *Nature Biotechnology 28*, 7 (July 2010), 691–693.

[165] Schatz, M. C., Sommer, D., Kelley, D., and Pop, M. De novo assembly of large genomes using cloud computing. In *The Biology of Genomes* (2010).

[166] Schmidt, D., Schwalie, P. C., Wilson, M. D., Ballester, B., Gonçalves, Â., Kutter, C., Brown, G. D., Marshall, A., Flicek, P., and Odom, D. T. Waves of Retrotransposon Expansion Remodel Genome Organization and CTCF Binding in Multiple Mammalian Lineages. *Cell* (Jan. 2012).

[167] Schumacher, A., Pireddu, L., Niemenmaa, M., Kallio, A., Korpelainen, E., Zanetti, G., and Heljanko, K. SeqPig: simple and scalable scripting for large sequencing data sets in Hadoop. *Bioinformatics* (Nov. 2013).

[168] Schwalie, P. C., Ward, M., Cain, C., Faure, A., Gilad, Y., Odom, D., and Flicek, P. Co-binding by YY1 identifies the transcriptionally active, highly conserved set of CTCF-bound regions in primates. *Genome Biology* (under review).

[169] Sebat, J., Lakshmi, B., Malhotra, D., Troge, J., Lese-Martin, C., Walsh, T., Yamrom, B., Yoon, S., Krasnitz, A., Kendall, J., Leotta, A., Pai, D., Zhang, R., Lee, Y.-H., Hicks, J., Spence, S. J., Lee, A. T., Puura, K., Lehtimäki, T., Ledbetter, D., Gregersen, P. K., Bregman, J., Sutcliffe, J. S., Jobanputra, V., Chung, W., Warburton, D., King, M.-C., Skuse, D., Geschwind, D. H., Gilliam, T. C., Ye, K., and Wigler, M. Strong association of de novo copy number mutations with autism. *Science 316*, 5823 (Apr. 2007), 445–449.

[170] Sharp, A. J., Hansen, S., Selzer, R. R., Cheng, Z., Regan, R., Hurst, J. A., Stewart, H., Price, S. M., Blair, E., Hennekam, R. C., Fitzpatrick, C. A., Segraves, R., Richmond, T. A., Guiver, C., Albertson, D. G., Pinkel, D., Eis, P. S., Schwartz, S., Knight, S. J. L., and Eichler, E. E. Discovery of previously unidentified genomic disorders from the duplication architecture of the human genome. *Nature Genetics 38*, 9 (Sept. 2006), 1038–1042.

[171] Shen, Y., Gu, Y., and Pe'er, I. A hidden Markov model for copy number variant prediction from whole genome resequencing data. *BMC Bioinformatics 12 Suppl 6* (2011), S4.

[172] Shendure, J., and Ji, H. Next-generation DNA sequencing. *Nature Biotechnology 26*, 10 (Oct. 2008), 1135–1145.

[173] Sindi, S., Helman, E., Bashir, A., and Raphael, B. J. A geometric approach for classification and comparison of structural variants. *Bioinformatics 25*, 12 (June 2009), i222–30.

[174] Sindi, S. S., Onal, S., Peng, L., Wu, H.-T., and Raphael, B. J. An integrative probabilistic model for identification of structural variation in sequencing data. *Genome Biology 13*, 3 (Mar. 2012), R22.

[175] Smit, A., Hubley, R., and Green, P. Repeatmasker open-3.0. `http://www.repeatmasker.org`, 1996-2010.

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

[177] Stein, L. D. Towards a cyberinfrastructure for the biological sciences: progress, visions and challenges. *Nature Reviews Genetics 9*, 9 (Sept. 2008), 678–688.

[178] Stein, L. D. The case for cloud computing in genome informatics. *Genome Biology 11*, 5 (2010), 207.

[179] Stephens, P. J., Greenman, C. D., Fu, B., Yang, F., Bignell, G. R., Mudie, L. J., Pleasance, E. D., Lau, K. W., Beare, D., Stebbings, L. A., McLaren, S., Lin, M.-L., Mcbride, D. J., Varela, I., Nik-Zainal, S., Leroy, C., Jia, M., Menzies, A., Butler, A. P., Teague, J. W., Quail, M. A., Burton, J., Swerdlow, H., Carter, N. P., Morsberger, L. A., Iacobuzio-Donahue, C., Follows, G. A., Green, A. R., Flanagan, A. M., Stratton, M. R., Futreal, P. A., and Campbell, P. J. Massive genomic rearrangement acquired in a single catastrophic event during cancer development. *Cell 144*, 1 (Jan. 2011), 27–40.

[180] Talwalkar, A., Liptrap, J., Newcomb, J., Hartl, C., Terhorst, J., Curtis, K., Bresler, M., Song, Y. S., Jordan, M. I., and Patterson, D. SMASH: A Benchmarking Toolkit for Variant Calling. *ArXiv e-prints* (Oct. 2013).

[181] Thain, D., Tannenbaum, T., and Livny, M. Distributed computing in practice: the condor experience. *Concurrency - Practice and Experience 17*, 2-4 (2005), 323–356.

[182] Treangen, T. J., and Salzberg, S. L. Repetitive DNA and next-generation sequencing: computational challenges and solutions. *Nature Reviews Genetics 13*, 1 (Jan. 2012), 36–46.

[183] Trinh, Q. M., Jen, F.-Y. A., Zhou, Z., Chu, K. M., Perry, M. D., Kephart, E. T., Contrino, S., Ruzanov, P., and Stein, L. D. Cloud-based uniform ChIP-Seq processing tools for modENCODE and ENCODE. *BMC Genomics 14* (2013), 494.

[184] Tüzün, E., Sharp, A. J., Bailey, J. A., Kaul, R., Morrison, V. A., Pertz, L. M., Haugen, E., Hayden, H., Albertson, D., Pinkel, D., Olson, M. V., and Eichler, E. E. Fine-scale structural variation of the human genome. *Nature Genetics 37*, 7 (July 2005), 727–732.

[185] Untergasser, A., Cutcutache, I., Koressaar, T., Ye, J., Faircloth, B. C., Remm, M., and Rozen, S. G. Primer3 - new capabilities and interfaces. *Nucleic Acids Research 40*, 15 (2012), e115.

[186] Vilella, A. J., Severin, J., Ureta-Vidal, A., Heng, L., Durbin, R., and Birney, E. EnsemblCompara GeneTrees: Complete, duplication-aware phylogenetic trees in vertebrates. *Genome Research 19*, 2 (Feb. 2009), 327–335.

[187] Volik, S., Zhao, S., Chin, K., Brebner, J. H., Herndon, D. R., Tao, Q., Kowbel, D., Huang, G., Lapuk, A., Kuo, W.-L., Magrane, G., de Jong, P., Gray, J. W., and Collins, C. End-sequence profiling: sequence-based analysis of aberrant genomes. *Proceedings of the National Academy of Sciences of the United States of America 100*, 13 (June 2003), 7696–7701.

[188] Walsh, T., McClellan, J. M., McCarthy, S. E., Addington, A. M., Pierce, S. B., Cooper, G. M., Nord, A. S., Kusenda, M., Malhotra, D., Bhandari, A., Stray, S. M., Rippey, C. F., Roccanova, P., Makarov, V., Lakshmi, B., Findling, R. L., Sikich, L., Stromberg, T., Merriman, B., Gogtay, N., Butler, P., Eckstrand, K., Noory, L., Gochman, P., Long, R., Chen, Z., Davis, S., Baker, C., Eichler, E. E., Meltzer, P. S., Nelson, S. F., Singleton, A. B., Lee, M. K., Rapoport, J. L., King, M.-C., and Sebat, J. Rare structural variants disrupt multiple genes in neurodevelopmental pathways in schizophrenia. *Science 320*, 5875 (Apr. 2008), 539–543.

[189] Wang, J., Mullighan, C. G., Easton, J., Roberts, S., Heatley, S. L., Ma, J., Rusch, M. C., Chen, K., Harris, C. C., Ding, L., Holmfeldt, L., Payne-Turner, D., Fan, X., Wei, L., Zhao, D., Obenauer, J. C., Naeve, C., Mardis, E. R., Wilson, R. K., Downing, J. R., and Zhang, J. CREST maps somatic structural variation in cancer genomes with base-pair resolution. *Nature Methods 8*, 8 (Aug. 2011), 652–654.

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

[191] Weischenfeldt, J., Symmons, O., Spitz, F., and Korbel, J. O. Phenotypic impact of genomic structural variation: insights from and for human disease. *Nature Reviews Genetics 14*, 2 (Feb. 2013), 125–138.

[192] Weston, J., Ratle, F., Mobahi, H., and Collobert, R. Deep learning via semi-supervised embedding. In *Neural Networks: Tricks of the Trade*, G. Montavon, G. Orr, and K.-R. Müller, Eds., vol. 7700 of *Lecture Notes in Computer Science.* Springer Berlin Heidelberg, 2012, pp. 639–655.

[193] Wienberg, J. The evolution of eutherian chromosomes. *Current Opinion in Genetics & Development 14*, 6 (Dec. 2004), 657–666.

[194] Wong, K., Keane, T. M., Stalker, J., and Adams, D. J. Enhanced structural variant and breakpoint detection using SVMerge by integration of multiple detection methods and local assembly. *Genome Biology 11*, 12 (2010), R128.

[195] 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.

[196] Xu, H., Handoko, L., Wei, X., Ye, C., Sheng, J., Wei, C.-L., Lin, F., and Sung, W.-K. A signal-noise model for significance analysis of ChIP-seq with negative control. *Bioinformatics 26*, 9 (May 2010), 1199–1204.

[197] Yalcin, B., Wong, K., Agam, A., Goodson, M., Keane, T. M., Gan, X., Nellåker, C., Goodstadt, L., Nicod, J., and Bhomra, A. Sequence-based characterization of structural variation in the mouse genome. *NATURE 477*, 7364 (2011), 326–329.

[198] Yang, L., Luquette, L. J., Gehlenborg, N., Xi, R., Haseley, P. S., Hsieh, C.-H., Zhang, C., Ren, X., Protopopov, A., Chin, L., Kucherlapati, R., Lee, C., and Park, P. J. Diverse mechanisms of somatic structural variations in human cancer genomes. *Cell 153*, 4 (May 2013), 919–929.

[199] Yasuda, T., Suzuki, S., Nagasaki, M., and Miyano, S. ChopSticks: High-resolution analysis of homozygous deletions by exploiting concordant read pairs. *BMC Bioinformatics 13*, 1 (Oct. 2012), 279.

[200] Ye, K., Schulz, M. H., Long, Q., Apweiler, R., and Ning, Z. Pindel: a pattern growth approach to detect break points of large deletions and medium sized insertions from paired-end short reads. *Bioinformatics 25*, 21 (Nov. 2009), 2865–2871.

[201] Yoon, S., Xuan, Z., Makarov, V., Ye, K., and Sebat, J. Sensitive and accurate detection of copy number variants using read depth of coverage. *Genome Research 19*, 9 (Sept. 2009), 1586–1592.

[202] Zeitouni, B., Boeva, V., Janoueix-Lerosey, I., Loeillet, S., Legoix-né, P., Nicolas, A., Delattre, O., and Barillot, E. SVDetect: a tool to identify genomic structural variations from paired-end and mate-pair sequencing data. *Bioinformatics 26*, 15 (Aug. 2010), 1895–1896.

[203] Zhang, J., and Wu, Y. SVseq: an approach for detecting exact breakpoints of deletions with low-coverage sequence data. *Bioinformatics 27*, 23 (Nov. 2011), 3228–3234.

[204] Zhang, L., Gu, S., Wang, B., Liu, Y., and Azuaje, F. Gene set analysis in the cloud. *Bioinformatics* (Nov. 2011).

[205] Zhao, S., Prenger, K., Smith, L., Messina, T., Fan, H., Jaeger, E., and Stephens, S. Rainbow: a tool for large-scale whole-genome sequencing data analysis using cloud computing. *BMC Genomics 14*, 1 (2013), 425.

[206] Zheng, Y.-S., Zhang, H., Zhang, X.-J., Feng, D.-D., Luo, X.-Q., Zeng, C.-W., Lin, K.-Y., Zhou, H., Qu, L.-H., Zhang, P., and Chen, Y.-Q. MiR-100 regulates cell differentiation and survival by targeting RBSP3, a phosphatase-like tumor suppressor in acute myeloid leukemia. *Oncogene 31*, 1 (Jan. 2012), 80–92.