Efficient Mapping-Free Methods for Discovery and Genotyping of Structural Variations

By

PARSOA KHORSAND
DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Fereydoun Hormozdiari, Chair

C. Titus Brown

Vladimir Filkov

Committee in Charge

2022

Contents

List of Figures

List of Tables

Abstract

Efficient Mapping-Free Methods for Discovery and Genotyping of Structural Variations

Structural variants (SVs) account for a large amount of sequence variability across genomes and play an important role in human evolution and diseases. Despite massive efforts over the years, the discovery of SVs in individuals remains challenging due to the highly repetitive nature of the human genome and the existence of complex SVs. The dominant mapping-based framework for SV discovery has several drawbacks including dependence on resource intensive mapping algorithms and an increased possibility of error in repetitive regions of the genome due to ambiguous read mappings. As a result, new computational methods are needed that can genotype different types of SVs in both short and long read data with high accuracy.

In this thesis, we first propose an ultra-efficient mapping-free approach for genotyping common SVs on short Illumina reads in Chapter 2. Our method Nebula generates databases of *k*-mers for catalogs of common SVs and counts these *k*-mers in unmapped samples to predict SV genotypes using a likelihood model of the *k*-mer counts. Nebula is the first method known to us that's capable of directly mapping-free SV genotyping from raw FASTQ files. We show that Nebula is not only an order of magnitude faster than mapping-based approaches for genotyping SVs, but it also has comparable accuracy to state-of-the-art approaches. Furthermore, Nebula is a generic framework that is not limited to specific types of SVs.

Next we introduce the concept of substring-free sample-specific strings (SFS) as an effective tool for comparative variant discovery between pairs of accurate long-read sequencing samples (e.g., PacBio HiFi) in Chapter 3. The SFS are sequences specific to a genome (or equivalently its sequencing reads) with regards to another genome that also do not occur as substrings of one another. We then introduce the Ping-Pong algorithm for theoretically and practically efficient extraction of SFS between a pair of target and reference samples by building an FMD index of the reference sample and querying the reads of the target sample against this index. Ping-Pong is a mapping-free method and is therefore not hindered by the shortcomings of the reference genome and mapping algorithms. We show that Ping-Pong is capable of accurately finding SFS representing nearly all variation (> 98%) reported across pairs or trios of WGS samples using PacBio HiFi data.

Finally in Chapter 4 we introduce `SVDSS`, a novel hybrid method for discovery of SVs from PacBio HiFi reads that combines the SFS concept with partial-order alignment (POA) and local assembly to yield highly accurate SV predictions. With experiments on three human samples, we show that `SVDSS` outperforms state-of-the-art methods for SV discovery on long-read data and achieves significant improvements in recall and precision particularly when discovering SVs in repetitive and traditionally difficult regions of the genome.

Acknowledgments

I would like to thank my principal advisor Professor Fereydoun Hormozdiari for giving me the opportunity to be part of his research group at UC Davis Genome Center and for being supportive and caring during every stage of my Ph.D. This thesis wouldn't have been possible without his wise advice and deep insight.

I owe my gratitude to Professor Titus Brown and Professor Vladimir Filkov for their insightful feedback on my thesis and for serving on my degree committee. I would also like to thank Professor Dan Gusfield and Professor Mohammad Sadoghi for serving on my Qualifying Exam committee.

I would like to thank my colleagues Dr. Luca Denti, Dr. Rayan Chikhi, Dr. Paola Bonizonni, Dr. Can Alkan and Dr. Daniel Standage for fruitful discussions and collaborations over the years.

I would like to thank Dr. Ali Sharifi Zarchi for being the original inspiration for me to pursue a Ph.D in bioinformatics. I'll always be grateful for his mentorship and kind support.

I would also like to thank my family and friends for their support along the way. To my parents, Hamid and Homeyra, thank you for your ever-present support in every step of my life and to my friends Sajjad, Mahyar and Kouroush for their companionship and support during the past three years.

Chapter 1

Introduction and Background

1.1 Structural Variants

Structural variants (SVs) are defined as medium and large size (> 50 bp) genomic alterations. SVs have many different types, e.g. deletion, insertions, duplication, and inversions. Complex SVs can occur as combinations of basic types. Although SVs are not the most ubiquitous type of genetic variants, the total volume of base-pairs impacted by SVs is far exceeds that of other type of genetic variant including single-nucleic variants (SNVs) [1]. However, efficient and accurate genotyping of all types of SVs using whole-genome sequencing (WGS) data is not a trivial task.

It has become clear that SVs are a major contributing factor to human diseases [2–4], population genomics [1, 5] and evolution [6]. Somatic SVs are one of the major causative variants in different types of cancer [7–10]. Furthermore, study of rare and *de novo* SVs in disease such as autism and epilepsy has proven the significant contribution of these variants in such diseases [4, 11–14].

Recent orthogonal studies of genomic variants have shown that SVs are the least well-characterized type of variants with many basic questions still not completely resolved, such as the average number of SVs per sample or sequence biases influencing their formation [15–17]. In addition, the homology-driven mechanisms behind SV formation (e.g., non-allelic homologous recombination) has contributed to the complexity of their systematic study [18]. It is believed that a large fraction of polymorphic SVs are still not fully characterized [19, 20].

High-throughput short-read sequencing (i.e., Illumina) has been the driving force behind most of the WGS studies in the past decade. Short-read sequencing is cheap, provides high-throughput data, and has a low error rate [21]. In the 1000 Genomes Project (1KG) over 42,000 SVs were discovered and genotyped in over 2,500 Illumina samples [1]. However, in many of the large scale genomic studies, SVs are being ignored or are merely an afterthought. The main reason behind SVs not being as thoroughly studied as other types of variants such as SNPs, is due to the complexity of efficient and accurate discovery and genotyping of these types of variants. It is hypothesized that lack of comprehensive studies of SVs is one of the contributing factors to the missing heritability gap observed in complex disorders [22, 23].

Today, WGS data continues to provide invaluable insight into every aspect of biology. In particular, comparative analysis of multiple samples using WGS data is fundamental in understanding the genetics of disorders, traits, and evolution. The comparison of differences found between the exome and genome of affected cases and unaffected controls has been successfully used for finding genetic variants associated with disorders and for guiding the prediction of genes contributing to such disorders [24]. Population genomics studies benefit from WGS data by finding shared or discriminating sequences and genomic variants between different populations [25, 26]. Furthermore, evolutionary studies also benefit from such comparative studies in a multiple species setting [27, 28]. However, short-read sequencing also has several major drawbacks. First, the assembly of the eukaryotic genomes using short-read sequencing data is non-trivial and computationally resource-intensive [29]. Second, the short length of the reads (generally below 250bp) produced by these technologies has caused significant complexity and ambiguity in studying repeat regions of the genome [15, 30, 31]. Third, the quality of SVs and other complex variant calls predicted using short-read data has remained low despite significant bioinformatics efforts, and still requires orthogonal validations [32, 33]. Finally, several types of genetic variations are hard to predict using short-read sequencing technologies due to their repeat nature (e.g., VNTR expansions [34]).

A comparison of the SV predictions from state-of-the-art computational methods (e.g., LUMPY [35], DELLY [36], TARDIS [32], and Pindel [37]) using short-read Illuimna WGS data against the calls produced using long-read data indicated that many SVs (> 50%) are missed by our best practices using short-read sequencing data [15]. Thus, there is still a need for approaches which can efficiently and accurately genotype SVs in short-sequencing samples.

With the introduction of long-read sequencing technologies (e.g., PacBio or Oxford Nanopore) we have access to much longer reads (> 10 kbp) that can be used to overcome the above-mentioned shortcomings of short-read sequencing [33, 38, 39]. WGS data from long-read sequencing technologies makes it possible to discover and further study variants that were either hidden or unreliably predicted from short-read data. Recent studies show that more than 50% of SVs being reported from long-read data were previously missed by short-read sequencing data [15, 31, 33].

1.2 Methods for Discovery of Structural Variants

The current approaches for SV discovery and genotyping are mainly based on first mapping the reads to the reference genome and then predicting the genotypes by analyzing the mappings for presence of certain types of signatures showing each type of SV [40], [41], [42]. This mapping-based framework has several main drawbacks. First, the mapping step is resource intensive, particularly for short-read samples, often taking upwards of a day for high coverage samples. Second, genotyping any variant close to heavily repeatitive regions in the reference genome (e.g microsatellites, segmental duplications, etc) would be less accurate due to potentially inaccurate mappings, meaning the method may fail to use all the sequencing reads that are available for a SV. Furthermore, it is established that predicting complex SVs such as inversion-duplications - that account for a significant fraction of SVs - using purely mapping-based approaches can result in increased false discovery rates compared to basic SV types [32, 33, 43]. Finally, the reference genome gaps and misasseblies could cause further complications in predicting SVs in these regions and result in an increase in false or missed calls. Delly [36], PBSV [44] and CuteSV [45] are examples of mapping-based methods for SV discovery.

Mapping-free methods that don't require read mappings are becoming popular as an alternative to traditional approaches. DISCOSNP [46] was one of the first approaches developed for predicting SNPs efficiently using *k*-mers counts. The tools LAVA [47], VarGeno [48] and MALVA [49] are examples of mapping-free methods developed for fast genotyping of common SNPs using *k*-mer counts. Merfin [50] is a mapping-free variant polishing tool that filters a provided VCF file of variants calls by building the haplotypes and checking for presence of erroneous *k*-mers not present in sequencing reads. Most current mapping-free methods for variation discovery and genotyping are limited to SNPs and small INDELs. BayesTyper [51] is a recent method that can genotype common SNPs, indels and SVs by performing exact alignment of *k*-mers to a haplotype graph of its input variants.

The growing list of such mapping-free methods relying on *k*-mers for variant detection has led to development of various tools for fast and accurate *k*-mer quantification. Some of the tools used for fast *k*-mer quantification include JellyFish [52], Khmer [53], DSK [54] and KMC [55].

Finally, assembly-based methods aim to find variations in a sample by building a complete or local assembly of the reads. However, these methods are very computationally resource intensive and often require integration of data from multiple different technologies (i.e., long-reads, short-reads, and Hi-C) [56] and extensive polishing and post-processing to yield a high-quality *de novo* assembly suitable for variant prediction, thus making them impractical for large-scale SV discovery across many samples.

1.3 Methods for Comparative Study of Structural Variants

One of the most common use cases of WGS sequencing is comparative analysis between different genomes [57], often from different populations of the same species. Comparative analysis is usually carried out by mapping reads of the different individuals under study to a common reference genome, calling variants on each sample and detecting the differences between the observed variants [15, 42, 58–60]. This strategy is effective for comparing SNPs, however for many SVs the exact breakpoint position is hard to establish and ambiguities can negatively affect accuracy. There are several heuristics used for comparing SVs in multiple samples by considering that the exact breakpoint for the SV might not be known or ambiguous[1]. These are based on merging SVs with approximately close breakpoints and considering reciprocal overlaps as a match [33]. Such

[1]`https://simpsonlab.github.io/2015/06/15/merging-sv-calls/`

heuristics tend to work for SVs in simple regions of the genome. However, for more complex scenarios such as STR/VNTR expansions [61, 62], SVs with adjacent SNP variants [63], or SVs with breakpoints in repeats (e.g., segmental duplications) this will result in reduction of accuracy as these heuristics tend to fail [32, 33, 64].

An alternative approach for comparative genome analysis is not to directly compare the predicted variants among multiple samples but rather to find the **sequences containing breakpoints** that are different between samples. This approach can be implemented without the need to map the reads to the reference genome and predict variants per sample. NovoBreak [65] is one such tool that utilizes *k*-mer counts to predict different types of somatic variants between tumor and normal samples using whole-genome sequencing data. DISCOSNP++ is another method that predicts indels between multiple sequenced samples using raw unassembled reads [66]. DE-Kupl [67] is an example of a mapping-free method for detecting RNA-Seq variations.

Discovery of *de novo* variants in families - variants in the child that were not inherited from either of the parents - is another critical application of comparative analysis. The tools Scalpel [68], COBASI [69], and Kevlar [70] are mapping free approaches for accurate discovery of *de novo* variants using whole-exome sequenced or whole-genome sequenced samples.

Finally, mapping-free approaches have also been utilized in improving association studies using whole-genome sequencing data. The tool HAWK [71] is capable of fast and accurate discovery of variants associated with phenotypes of interest by comparing the *k*-mer frequencies between cases and controls. HAWK works by first finding "significant *k*-mers" which are associated with the phenotype of interest in cases versus controls, and then builds assemblies of these *k*-mers for predicting the significant variants.

1.4 Our Contribution

During this dissertation, we present three novel approaches for study of SVs in both short and long-read data. Chapter 2 introduces Nebula, an extremely fast mapping-free method for genotyping of common SVs in unmapped Illumina short-read data using *k*-mer counts. Chapter 3 introduces the concept of substring-free sample-specific strings (SFS) and the Ping-Pong algorithm for comparative discovery of all genomic variants between pairs or trios of unmapped PacBio HiFi samples. Finally, in Chapter 4 we present `SVDSS`, a hybrid SV discovery method based on the Ping-Pong algorithm that combines mapping-free signature detection with partial-order alignment (POA) and local assembly to achieve significantly improved SV calling performance on PacBio HiFi samples.

Chapter 2

Ultra-efficient Mapping-free Structural Variation Genotyping Using *k*-mer Counts [1]

2.1 Motivation

Current approaches for genotyping SVs using WGS data are mainly based on first mapping the reads to the reference genome and then predicting the genotype [40], [41]. Mapping-free methods have also been explored as an alternative to mapping-based approaches for different genomic and transcriptome applications. These mapping-free approaches are not limited by the shortcomings of the mapping algorithms and tend to be much more computationally efficient. However, most such approaches are currently limited to genotyping small variants such as SNPs and indels. Here we are proposing a novel mapping-free approach, Nebula, that utilizes *k*-mer counts for efficient and accurate genotyping of (common) SVs in unmapped whole-genome sequenced sample.

2.2 Methods

Nebula is a two-stage approach and consists of a ***k*-mer extraction phase** and a **genotyping phase** (Figure 2.1). Given as input a set of SV coordinates (BED/VCF), the reference assembly (FASTA), and a set of mapped samples on which the genotype of the input SVs is already known (BAM), Nebula extracts a collection of *k*-mers that represent the input SVs (*k*-mer extraction phase). These

[1]This chapter was published as *Parsoa Khorsand, Fereydoun Hormozdiari. Nebula: Ultra-efficient mapping-free structural variant genotyper. Nucleic Acids Research, 49(8):e47-e47.*

extracted *k*-mers will then be used to genotype the same set of SVs on any new WGS sample(s) without the need to map the reads to the reference genome (genotyping phase). This is done by counting the *k*-mers in the WGS reads of the new sample(s) and predicting genotypes using a likelihood model. A graphical representation of the Nebula pipeline can be seen in Figure 2.1.

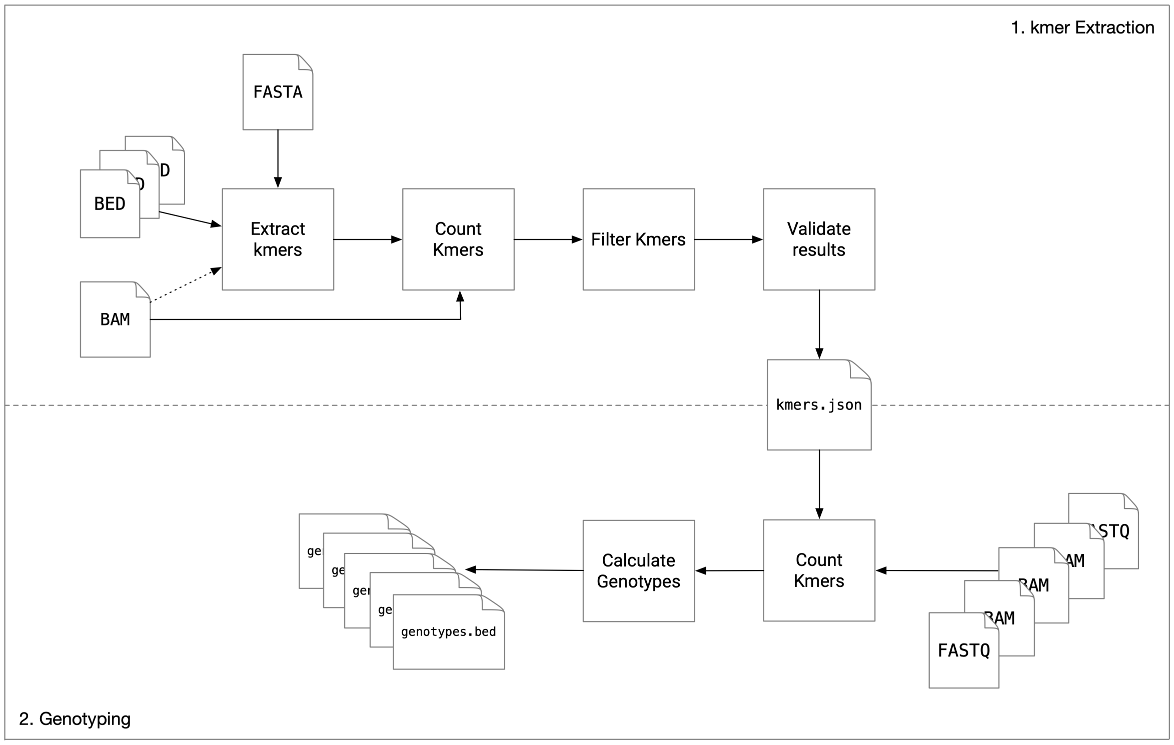


Figure 2.1: An overview of the entire Nebula pipeline. The upper half shows the *k*-mer extraction stage which takes as input a set of SV coordinates, the reference assembly, and a set of samples on which the genotypes of these SVs is known. The *k*-mer extraction stage selects a collection of *k*-mers to be used for genotyping. The bottom half shows the SV genotyping phase, which uses the *k*-mers extracted earlier to genotype the input SVs on any number of newly sequenced samples without mapping the reads.

Note that the preprocessing stage is essentially not mapping-free, however, we envision that the end user will not need to run the preprocessing stage themselves: a common scenario for using Nebula would be genotyping catalogues of SVs found in large-scale studies such as HGSV ([15]) on new samples. *k*-mers can be extracted for these SVs using the samples in the study and made publicly available for download. Users now only need to download the *k*-mers and use them to run the genotyping step, avoiding the need to rebuild the *k*-mer database locally.

For large-scale studies where it's not practical to map all samples, an alternative approach would be to map a small subset of the samples and use them for novel SV discovery and *k*-mer extraction (potentially with a mapping-based method), and use the resulting *k*-mers to genotype the remaining samples efficiently without the need for mapping. However, we believe this would be a less common scenario in practice.

2.2.1 Likelihood Model

The key assumption in Nebula is that each SV will increase and/or decrease the copy number of a specific set of *k*-mers in the genome. Note that the count of each *k*-mer in the WGS reads of a sample is directly correlated with the copy number of the *k*-mer in the corresponding genome. We develop a likelihood model to calculate the probability of different genotypes ($\{0/0, 0/1, 1/1\}$) per SV based on the counts of *k*-mers.

We define a unique *k*-mer as one that appears in exactly one loci in the sample's genome. For a given sample, we assume the number of reads containing a unique *k*-mer that are coming from each haplotype to follow a normal distribution $\mathcal{N}(\mu_h, \sigma_h^2)$. We also model the total number of reads containing that *k*-mer (i.e., the *k*-mer's count) in a diploid genome as the summation of two the normal distributions representing the number of reads containing the *k*-mer in each haplotype as $\mathcal{N}(\mu, \sigma^2) = \mathcal{N}(\mu_1, \sigma_1^2) + \mathcal{N}(\mu_2, \sigma_2^2)$ where μ_i and σ_i^2 are mean and variance for the corresponding haplotype.

However as we generally don't know which haplotype a sequencing read is from, we will directly estimate the sample-wide parameters μ and σ^2 rather than the haplotype-specific ones. For this, we select a large number of unique *k*-mers from conserved regions of the genome (e.g., exons) and count them in the sequencing reads of the sample. By further assuming that the sequencing coverage is equal for both haplotypes, the count of a unique *k*-mer present on only one haploid can be approximated using the normal distribution $\mathcal{N}(\mu/2, \sigma^2/2)$. Finally, the count of a *k*-mer not expected to be present in the genome is estimated by setting μ to zero and using a small fixed number for the variance. This provides the basis of the model that we use to calculate likelihood of SVs genotypes based on the *k*-mer counts.

2.2.2 GC Coverage Normalization

Illumina reads tend to have coverage bias depending on the GC content of the region being sequenced [72]. As a result, assuming uniform coverage across the whole genome could result in errors in calculating genotype likelihoods.

To correct for this, we select a large number of unique *k*-mers (500,000) from regions with different levels of GC content across the genome and count them to derive a separate estimate of μ for each GC content level. We define the GC content for a *k*-mer as the percentage of G or C bases in a 1000bp window centered on the *k*-mer. The GC content will have rounded integer values from 0 to 100. μ_{gc} is defined as the average of the coverage of all selected *k*-mers with GC content level of gc. Figure 2.2 shows the variations in *k*-mer coverage across the genome based on GC-content level.

Genotyping *k*-mers selected from a region with GC level of gc will use the corresponding $\mu_g c$ value in all likelihood calculations. For the genotyping *k*-mers selected from the reference genome, gc is defined as the GC content of the 1000bp window around them. For those *k*-mers selected from reads, the GC content of a 1000bp window around the read's mapping location is used.

The set of *k*-mers used for deriving coverage estimates is constant and does not need to be recalculated for each run. These *k*-mers are provided in `JSON` format on our Github repository and can be passed to Nebula as arguments during invocation.

2.2.3 *k*-mer Extraction

Nebula uses the coordinates of the input SVs, the reference genome, and mapped reads of WGS sample(s) on which the genotype of the SVs of interest are known to extract *k*-mers whose copy number is affected by the SVs. These *k*-mers either cross the SV's breakpoint or fall inside the region that is affected by the SV.

Sequencing reads that cross a SV's breakpoint are usually soft-clipped when mapped to the reference genome. For each SV, Nebula looks at soft-clipped reads mapping near its breakpoints and selects *k*-mers that overlap the clipped part of the read (Figure 2.3).

Nebula also uses the reference genome to extract additional *k*-mers from within the region that

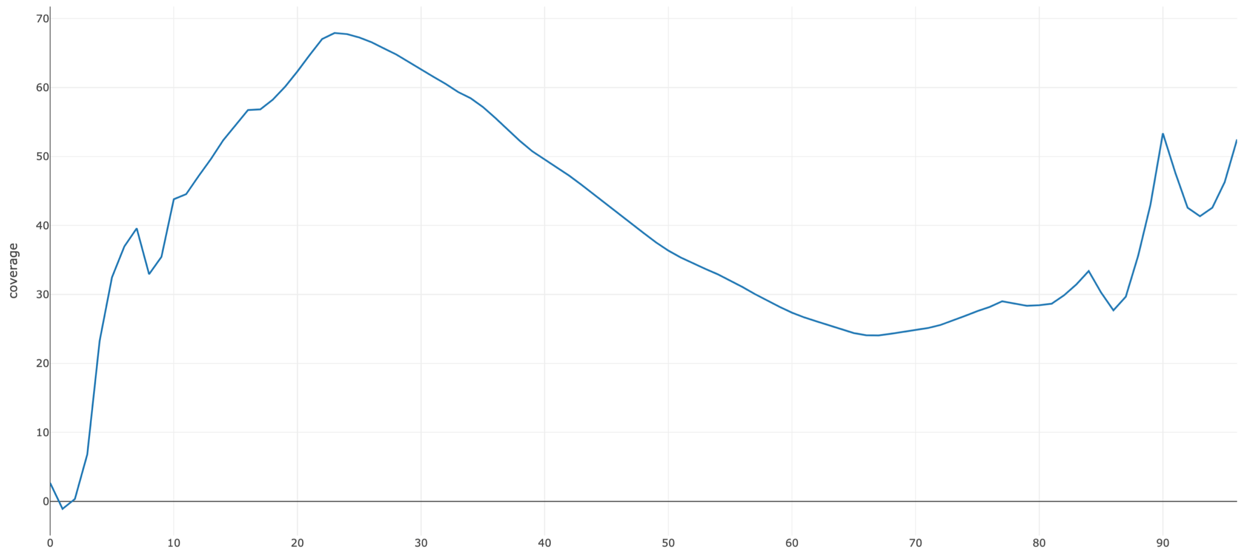
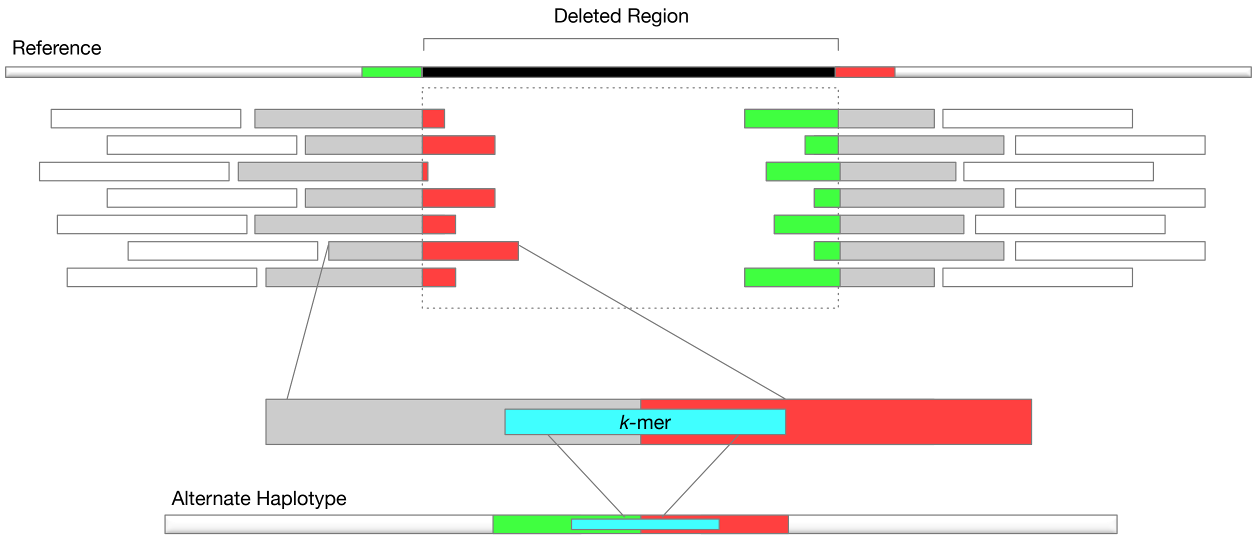


Figure 2.2: k-mer coverage variation with different GC-content levels in the NA19240 sample (40x sequencing coverage). The x-axis is GC content (0 to 100) and the y-axis shows the estimate k-mer coverage $\mu_g c$.

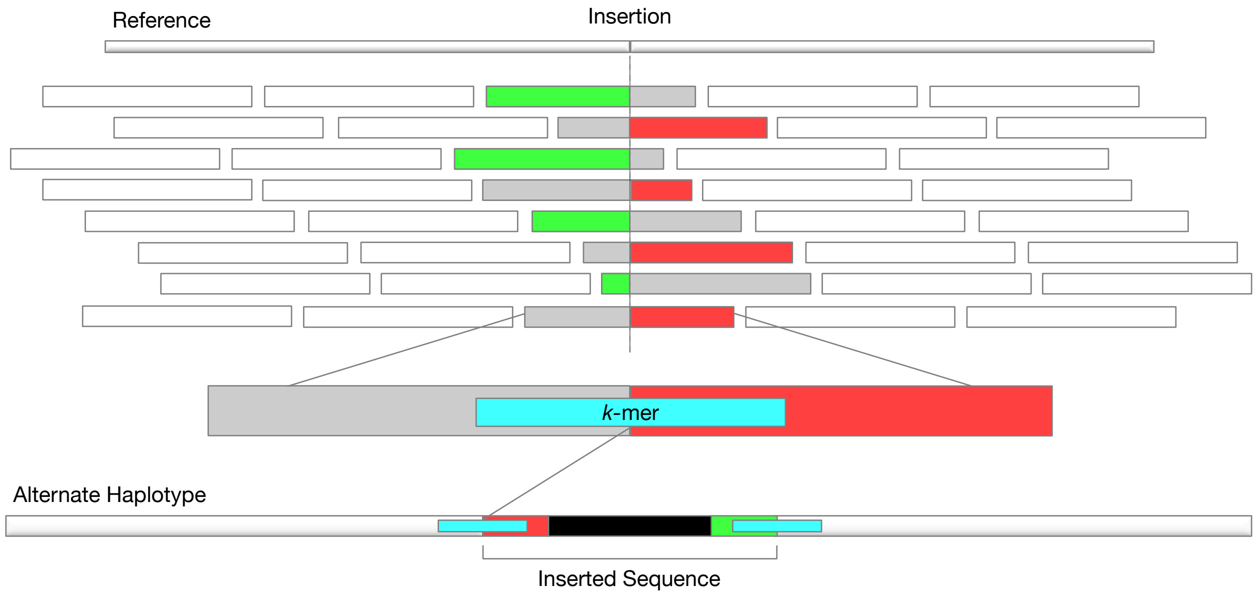
is affected by a SV (e.g inside the deleted region for a deletion or from the sequence that would be inserted into the genome for an insertion). We also extract unique k-mers that cross the breakpoints from the reference genome.

With the k-mers selected, Nebula scans the reference genome to filter any k-mer that also occurs in loci not impacted by the input SVs. Finally, the remaining k-mers are counted on each of the input sample with known SV genotypes. We use each k-mer independently to genotype its corresponding SV and filter those k-mers that do not predict the correct genotype. After filtering, the remaining k-mers are exported as the output of the k-mer extraction phase.

The likelihood of genotype g based on k-mer k with count c_k is calculated using the normal distribution as $\mathcal{L}(g|k) = p(k|g) = p(c_k|\mathcal{N}(\mu_{k,g}, \sigma^2_{k,g}))$ where $\mu_{g,k}$ and $\sigma^2_{g,k}$ are derived from the sample-wide mean μ and variance σ^2 depending on the expected copy number of the k-mer k for genotype g. For example, for an insertion SV, a k-mer selected from the inserted sequence is expected to be present on both haploids for a 1/1 genotype with $\mu_{1/1,k} = \mu$ and $\sigma^2_{1/1,k} = \sigma^2$ and on only one haploid for a 1/0 genotype with $\mu_{1/0,k} = \mu/2$ and $\sigma^2_{1/0,k} = \sigma^2/2$. We calculate the



(a) Deletion



(b) Insertion

Figure 2.3: *k*-mer extraction from clipped reads for a deletion (a) and an insertion (b). Red and green segments of the reads are soft-clipped by the aligner and correspond to the similarly colored regions of the alternate and reference haplotypes.

likelihood of all three possible genotypes using the above formulation and choose the one with the maximum likelihood as the genotype prediction.

2.2.4 Genotyping

Once k-mers have been extracted for a set of SVs, the same set of SVs can be genotyped on any new WGS sample(s) without the need to map the reads. The k-mers are counted on the sample and genotypes are predicted using an extension of the likelihood model. For a SV supported by multiple k-mers, the likelihood of each possible genotype $g \in \{0/0, 0/1, 1/1\}$ can be calculated as $\mathcal{L}(g|k_1, k_2, k_3, ...) = p(k_1, k_2, k_3, ...|g)$ where each k_i represents a different k-mer.

Note that the counts of k-mers corresponding to the same SV might not be independent as the k-mers may overlap one another. However, if we assume independence between k-mer counts, we can approximate the above likelihood by calculating the probability as the multiplication of probabilities of individual k-mers given the genotype (i.e., $\prod_i p(k_i|g)$). Note that $p(k_i|g)$ is calculated as $p(c_{k_i}|\mathcal{N}(\mu_{g,k_i}, \sigma^2_{g,k_i}))$ where the random variable c_{k_i} is the count of k-mer k_i in that sample. Furthermore, the values μ_{g,k_i} and σ^2_{g,k_i} are derived from sample-wide μ and σ according to the genotype g. We calculate the likelihood for all three possible genotypes $1/1, 1/0$ and $0/0$ for each SV and choose the one with the maximum likelihood as our prediction.

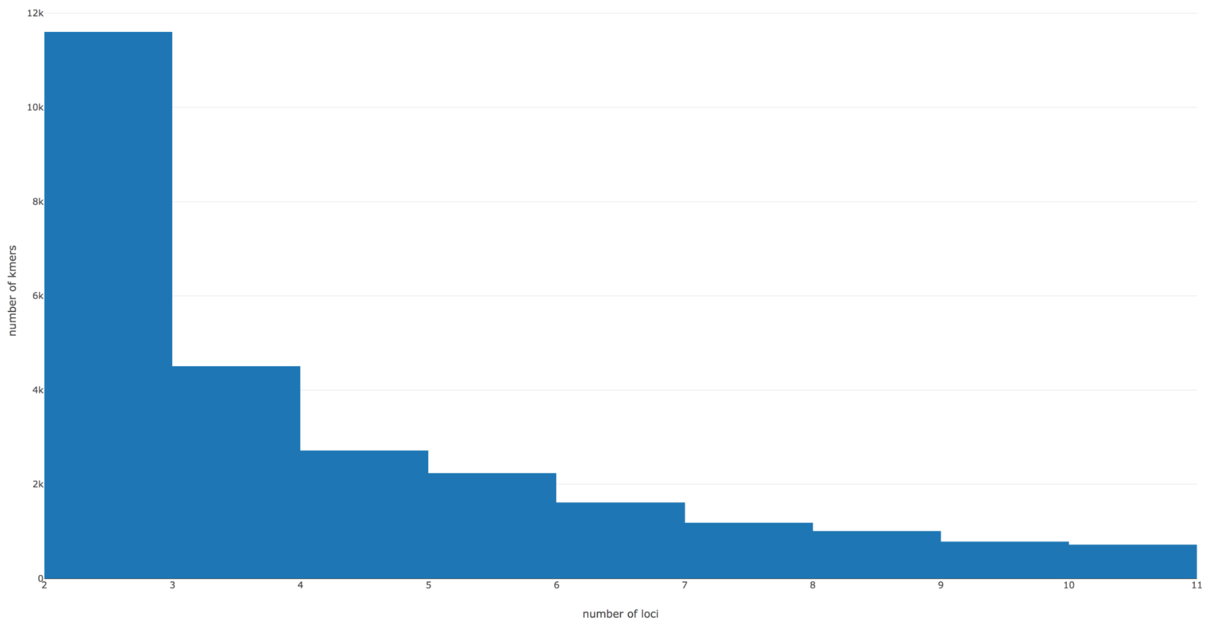
2.2.5 *k*-mer Masks and Loci Reduction

Nebula only works with unique k-mers, e.g k-mers that are only seen in one loci in the genome and are as a result associated with a single SV. However to increase the number of available k-mers and improve the accuracy of counting, we use a broader definition of a unique k-mer in our implementation which also takes into account the context surrounding the k-mer.

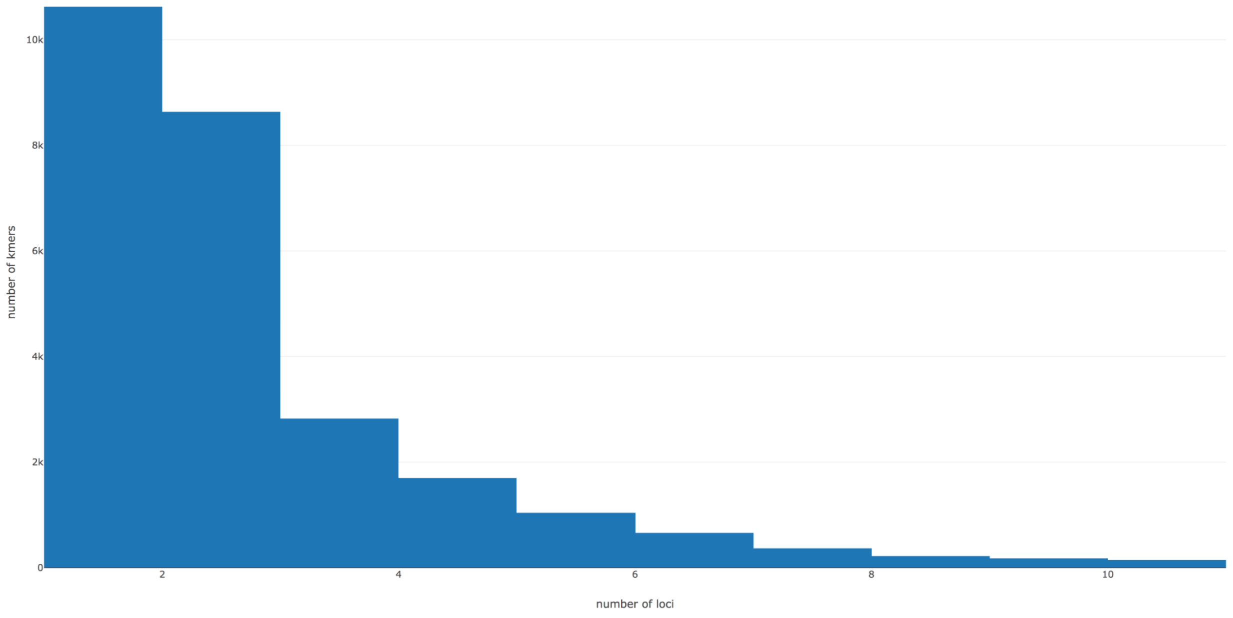
When extracting k-mers, Nebula stores the immediate left and right 32bp sequences surrounding a k-mer as “masks”. A k-mer is considered unique if it is associated with only one SV and the combination of the k-mer and its masks, 96bp in total, is only seen in one loci in the genome. In this sense, a k-mer could be present in multiple loci outside of the SV in the genome, but as long as the SV locus has unique masks that are not seen in any other non-SV loci, the k-mer is still considered unique. k-mers whose SV locus cannot be uniquely identified using the masks are discarded. This results in nearly half of the non-unique SV-associated k-mers to become effectively unique, significantly increasing the number of available k-mers and consequently the method’s recall. We refer to this use of masks as “loci-reduction”. Figure 2.4 better shows the effect of loci reduction.

When counting *k*-mers in the reads, Nebula checks for exact matches for the *k*-mers, however once a *k*-mer is found in a read, only approximate matches are required for the masks, i.e only 28bp of each mask needs to match. With short sequencing reads being typically shorter than 120bp, it’s unlikely to see a *k*-mer and both masks in a read, instead the presence of a single masks is enough, as long as the mask uniquely identifies the SV locus among all loci of the *k*-mer.

It is possible that a SNP or sequencing error would result in a SV-associated unique *k*-mer appearing in different regions in the genome, however, the use of masks prevents counting of such instances, eliminating the possibility of overestimating *k*-mer counts. However, as *k*-mers need to exactly match to be counted, Nebula may miss correct instances of *k*-mers that were affected by errors or SNPs and hence it is possible for a *k*-mer’s count to be underestimated. As it would be computationally expensive to consider the possibility of mismatch for *k*-mer, the current scheme is a compromise between speed and accuracy.



(a) Before reduction



(b) After reduction

Figure 2.4: *k*-mer loci reduction. Top figure shows the distribution of the number of loci of seemingly non-unique *k*-mer before reduction (x-axis begins at 2). Bottom figures shows more than half of *k*-mers being reduced to one loci considering masks (x-axis now begins at zero).

2.2.6 Implementation Details

Nebula is implemented entirely in C++ and is heavily parallelized using OpenMP [73]. To improve speed and reduce memory usage, the 32bp *k*-mers are hashed into 64-bit integer values in form of `unit64_t` types. Similarly, sequencing reads are stored in memory as arrays of 8-bit integers. This allows kmers to be extracted from reads by moving forward pointer to a `unit64_t` value and avoids the overhead of allocating a new `string` instance for every *k*-mer.

2.3 Experimental Results

We utilized both simulations and real data to quantify and evaluate the performance of Nebula using high quality SV predictions from long-read sequencing data on 1KG samples HG00514 (CHS trio, child), HG00733 (PUR trio, child) and NA19240 (YRI trio, child) [33].

2.3.1 Results on Simulated Illumina WGS Data

An extensive WGS simulation was performed to evaluate Nebula's performance for accurately genotyping SVs. The simulation consists of two stages: first we mutated a genome with the set of SVs from the 1KG dataset and used it for *k*-mer extraction. Second, we simulated a subset of these SVs on a new sample and used the extracted *k*-mers to genotype the simulated SVs.

During the first step, a diploid GRCh38 genome was mutated with the union of all insertions and deletions reported for samples HG00514 and HG00733 (11551 total SVs) with random genotype assignments of 1/0 or 1/1. Short paired-end sequencing reads were simulated from this diploid simulated genome using wgsim (`https://github.com/lh3/wgsim`) at 30x coverage and mapped using BWA-mem [74]. After running the *k*-mer extraction phase, Nebula found *k*-mers to genotype 11330 (98%) of the simulated SVs.

During the second stage of the simulation, another diploid genome was constructed from GRCh38 and was randomly mutated with the same set of SVs but with all three possible genotypes (0/0, 0/1 and 1/1) allowed. Paired-end short reads were generated from this genome at 30x coverage in FASTQ format and the extracted *k*-mers were used to genotype the sample.

The entire procedure was also repeated at 10x coverage to measure Nebula's resilience to low coverage. For the 10x simulation, *k*-mers could be extracted for 11304 (97.8%) SVs.

We compared Nebula's predictions against those of the mapping-based approaches SVTyper [41] and Delly [36], the graph-based approach Paragraph [75] and the *k*-mer-based approach BayesTyper [51]. Due to limitation of SVtyper and Delly on genotyping long insertions [76], we have excluded these tools from the comparison for insertions. Note that none of the mentioned methods except BayesTyper can genotype unmapped samples in FASTQ format directly and instead require mapped reads as input.

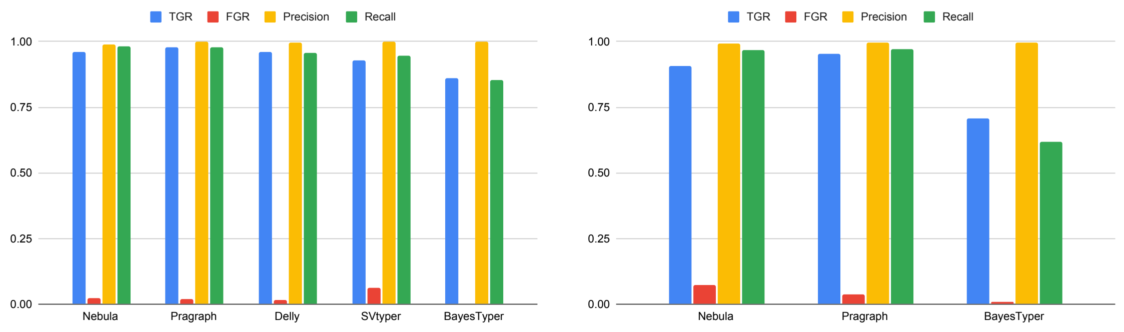
We calculated four different measures of accuracy for each method: The *true genotyping rate* (TGR) is defined as the number of correct genotype calls for each tool divided by the total number of input events. The *false genotyping rate* (FGR) is similarly defined as the number of false genotype calls made by a tool divided by the total number of calls made by that tool. *Precision* is defined as the number of true positive calls divided by all the positive calls ($1/1$ and $1/0$) made by a tool and finally *recall* is defined as the number of true positive calls produced by a tool divided by the total number of $1/1$ or $1/0$ SVs present on the sample.

In both simulations Nebula produces comparable results to state-of-the-art genotyping approaches without requiring the mapping of the reads to the reference genome. The detailed results for each simulation, separated by event type are presented in Figure 2.5 and 2.6.

Figure 2.6 shows the comparison of different accuracy metrics between Nebula and other tools when genotyping the 10x simulation. The lower coverage in the 10x sample results in a higher FGR for Nebula compared to the 30x simulation, as the separation of 1/1 and 1/0 genotypes based on *k*-mer counts has a smaller margin. However Nebula's accuracy metrics remain on-par with other methods, showing that the method is robust to low sequencing depth.

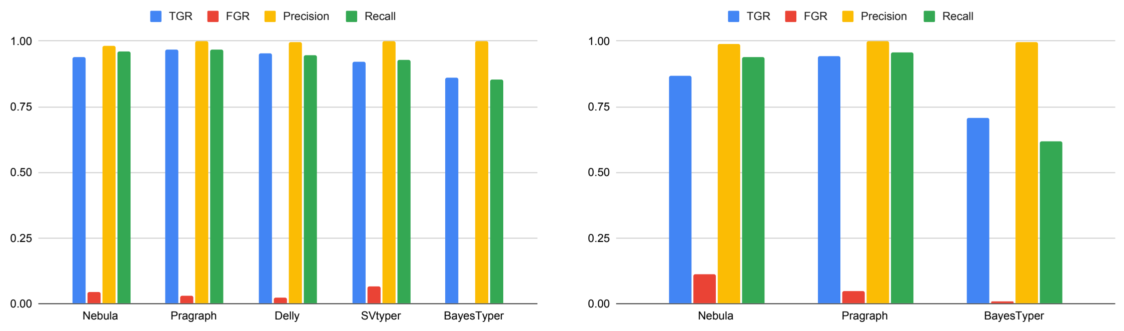
2.3.2 Results on 1KG Illumina WGS Data

We also used real WGS data for experimental evaluation of Nebula. We considered the union of all insertions, deletions and inversions reported from non-repeat regions of the HG00514 and HG00733 genomes as the set of input SVs [33]. We used these two samples to extract *k*-mers for the SVs and used the *k*-mers to genotype a third sample NA19240 with Nebula. We also used Delly, SVTyper, Paragraph and BayesTyper to genotype the selected set of SVs on NA19240 and validated their predictions against the 1KG callset. Figure 2.7 below provides an illustration of



(a) Deletions (b) Insertions

Figure 2.5: Comparison of different accuracy metrics between Nebula and other methods when genotyping SVs on the 30x simulation.



(a) Deletions (b) Insertions

Figure 2.6: Comparison of different accuracy metrics between Nebula and other methods when genotyping SVs on the 10x simulation.

how *k*-mers extracted from HG00514 and HG00733 can be used to genotype SVs on NA19240 using Nebula.

For evaluation, we only considered SVs that could be correctly genotyped on HG00514 and HG00733 using at least one of the four methods (Delly, SVTyper, Paragraph and BayesTyper) in the comparison. For consistency in validating genotypes, we have merged overlapping deletions and insertions (less than 10bp apart) in different samples into a single event. A total of 4810 deletions, 7511 insertions, and 81 inversions were considered for our evaluation.

We use the same metrics introduced earlier for comparing the performance of different methods and we observe that Nebula consistently performs equal to or better than the other state-of-the-art

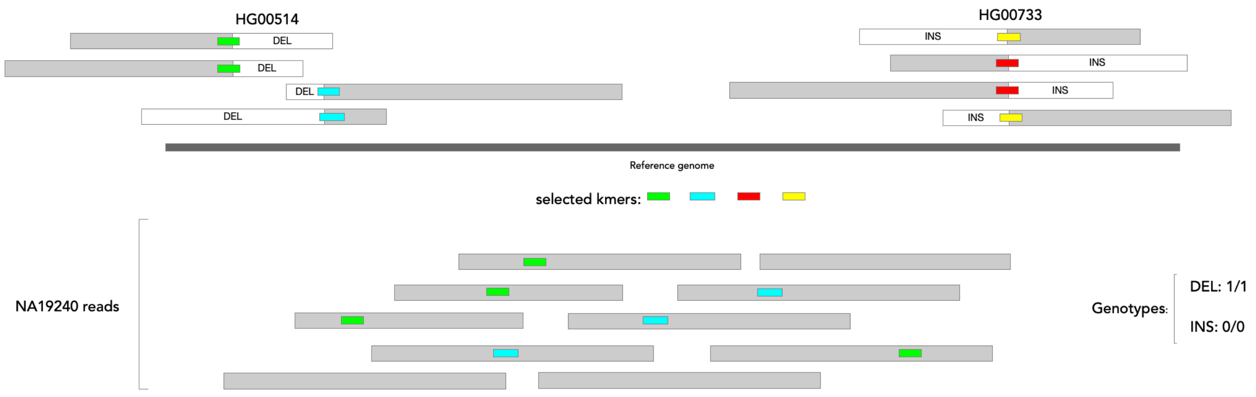
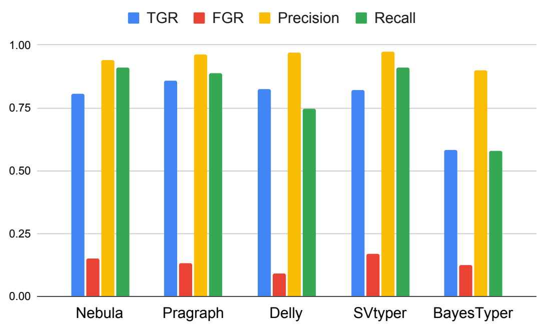


Figure 2.7: Example of genotyping SV on NA19240 using *k*-mers from HG00514 and HG00733. The green and blue *k*-mers shows breakpoints of a deletion in HG00514 while the red and yellow *k*-mers show breakpoints of an insertion present in HG00733. Only the green and blue *k*-mers are observed in NA19240 reads, which result in a genotype of 1/1 for the deletion and a 0/0 call for the insertion.

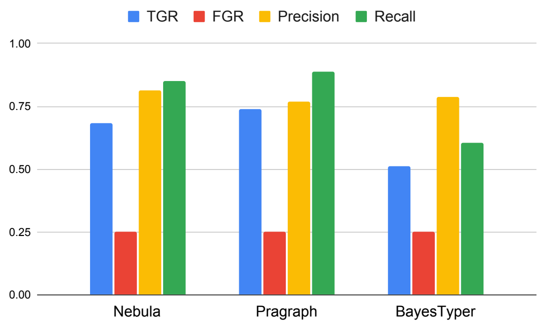
methods (Figure 2.8). As the input callset does not include genotypes for inversions and only marks them as present or not, we have only reported precision and recall for inversions. We couldn't genotype the inversions using Delly or BayesTyper and we have thus removed them from the comparison for inversions. Note that BayesTyper requires exact SV breakpoints for optimal performance; as a result, its performance for insertions and deletions may have been negatively affected due to inexact breakpoints for some of the SVs in the dataset.

2.3.2.1 Comparison Results for Mobile Element Polymorphisms on NA19240

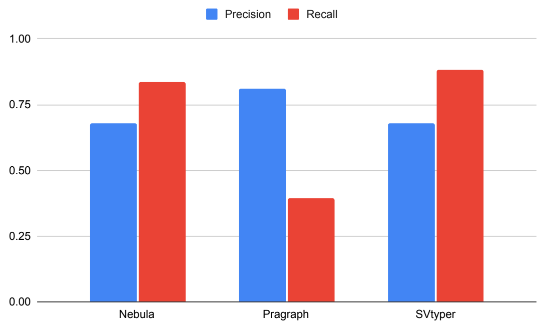
Figure 2.9 shows the performance of Nebula and other considered approaches when genotyping SVs categorized as transposable elements (`SVCLASS=ALU,L1,LTR,HERV` in the VCF files) on NA19240. Mobile element polymorphisms can manifest both as deletions and insertions. The SVs considered in this figure are a subset of those shown in Figure 2.8. Delly and SVTyper are not included in the comparison due to their limitation in genotyping insertions. Nebula and other methods perform relatively well on SVs involving mobile elements (e.g., SINE or LINE elements) and all methods achieve a precision of over 90%.



(a) Deletions



(b) Insertions



(c) Inversions

Figure 2.8: Comparison of different accuracy metrics between Nebula and other methods when genotyping SVs on NA19240.

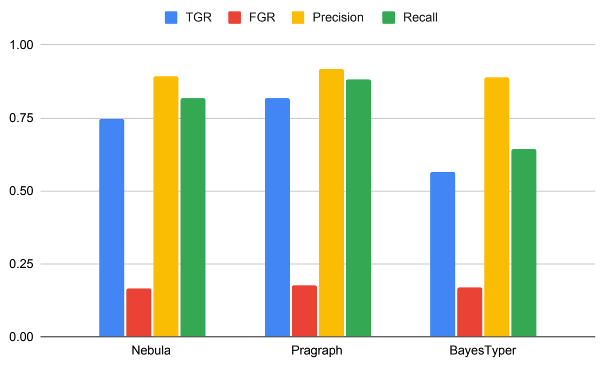
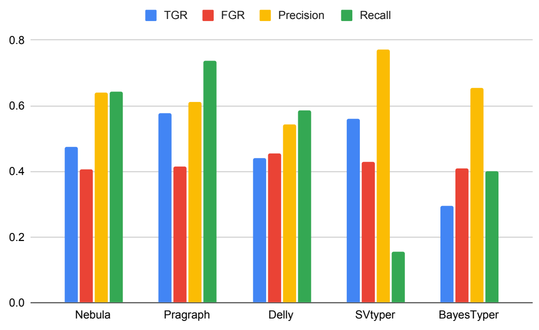
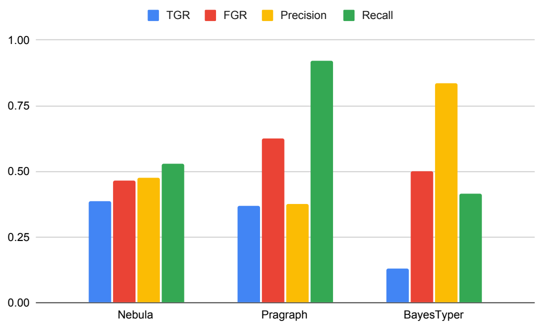


Figure 2.9: Comparison of different accuracy metrics between Nebula and other genotyping tools for genotyping MEIs on NA19240. Includes both repeat and non-repeat regions.



(a) Deletions



(b) Insertions

Figure 2.10: Comparison of different accuracy metrics between Nebula and other genotyping tools for repeat events on NA19240.

2.3.2.2 Comparison Results for SV Calls in Tandem Repeat and Satellite Regions on NA19240

We repeated the experiment on NA19240 using the SVs reported on tandem repeat and satellite regions of the HG00514 and HG00733 genomes, i.e those with `IS_TRF=TRUE` in the VCF file from 1KG. The comparison can be seen in Figure 2.10. All tools perform relatively poorly; no method achieves a TGR above 60% on deletions and all methods have a less than 50% TGR on insertions. Still Nebula achieves the highest TGR on insertions and the lowest FGR on deletions. This result shows the difficulty of genotyping SV in repetitive regions with short reads.

2.3.3 Results on Simons Genome Diversity Project Data (SGDP)

We used *k*-mers extracted for a total of 14103 insertion and deletion selected from the three 1KG samples HG00514, HG00733 and NA19240 to genotype the entire set of 279 samples from the Simons Genome Diversity Project [26] stored in BAM format on the cloud computation platform Cancer Genomics Cloud (CGC) [77].

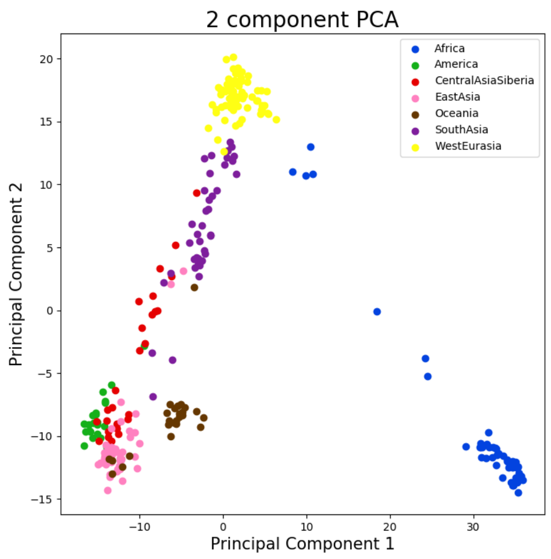
On average, we see about 19% homozygous and 24% heterozygous genotype predictions among all samples. We expect the genotypes to cluster samples based on geographical origin. For this we preformed a principal component analysis (PCA) on the SV genotypes and plotted the two most significant components (Figure 2.11). The PCA clearly separates populations of different continents with a greater level of separation between Africa and the rest (Figure 2.11a). We repeated the PCA analysis using one million randomly selected SNP calls from the Simons Genome Diversity Project [26] and plotted the two most significant components (Figure 2.11b). The plot from Nebula's genotypes captures the same structure as SGDP's SNP calls, showing the accuracy of our method for population studies.

We further checked our genotypes for concordance with Hardy-Weinberg Equilibrium and compared our result to those made by Paragraph. We used the `HWExact` function with p-value of 0.05 from `HardyWeinberg` R package ([78],[79]). Figure 2.12 shows the results of the test on insertion SVs calculated on SGDP samples. Paragraph fails on 31% of insertion calls while Nebula fails on only 26%. Similarly Paragraph fails on 33% of deletion calls while Nebula fails on 27%.

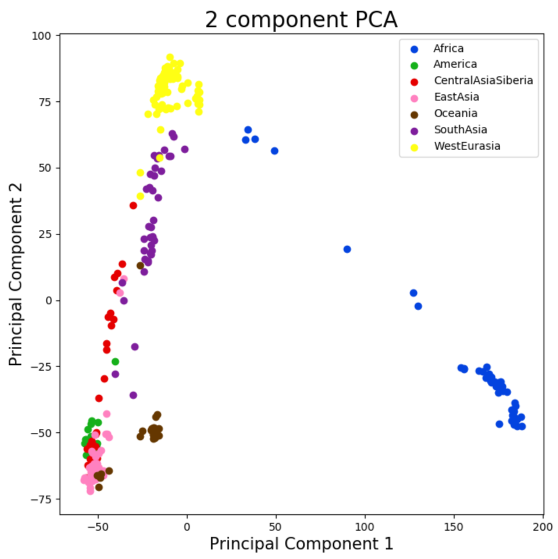
We need to note that although the 1KG SVs are based on GRCh38 coordinates, the SGDP samples are mapped against GRCh37. With Nebula's modest resource requirements and independence from mapping, each sample was genotyped accurately in under an hour and at a cost of $0.30 per sample without the need to remap the reads to GRCh38.

2.3.4 Time and Memory Performance

Nebula's main advantage is its ability to directly genotype unmapped samples with high efficiency and comparable accuracy to the state-of-the-art mapping-based genotypers. Furthermore, Nebula is not limited to specific types of SVs and can genotype deletions, insertions, inversions, or other



(a) SV genotypes (Nebula)



(b) SNP genotypes (SGDP)

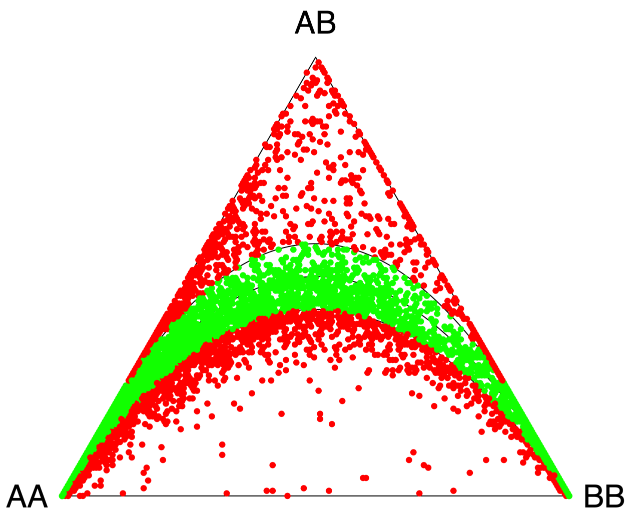
Figure 2.11: Population clustering of SGDP samples

types of SVs using a universal algorithm.

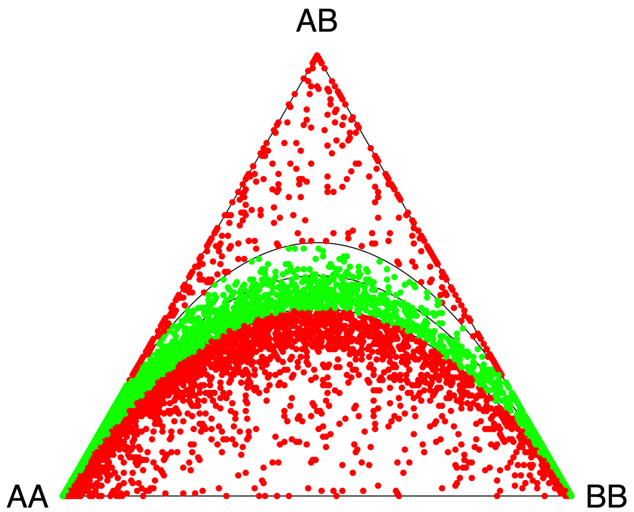
Figure 2.13 shows the runtime and peak memory usage of Nebula and other tools for genotyping NA19240. Nebula's genotyping stage has comparable runtime and often lower memory usage than most other tools with the exception of SVTyper which has the lowest memory usage and runtime among all the considered methods (excluding mapping time). Nebula is nearly 5 times faster than BayesTyper, another *k*-mer-based approach and uses a tenth of the memory while achieving higher recall and overall accuracy.

Nebula can be as much as 40 times faster than mapping-based methods in genotyping newly sequenced samples. This is particularly useful in large studies with hundreds to thousands of samples, where Nebula can be efficiently used to genotype common SVs on the entire cohort an order of magnitude faster than other approaches.

Although Nebula is meant to genotype unmapped FASTQ files, *k*-mers can also be counted in SAM, BAM and CRAM files with slight differences in performance between the different formats due to parsing and decoding. Unlike many mapping-based tools that require certain fields in input



(a) (Nebula)



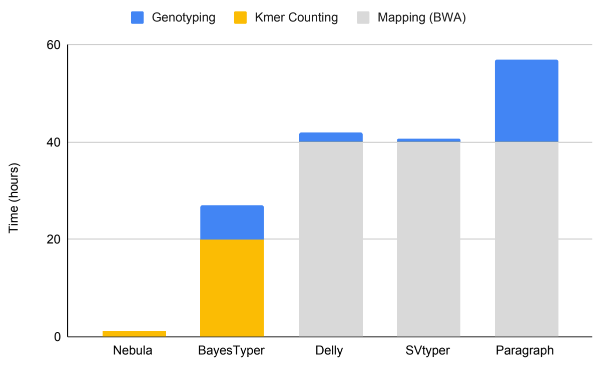
(b) Paragraph

Figure 2.12: Comparison of Hardy-Weinberg Equilibrium concordance on insertion SVs on SGDP data between Nebula and Paragraph.

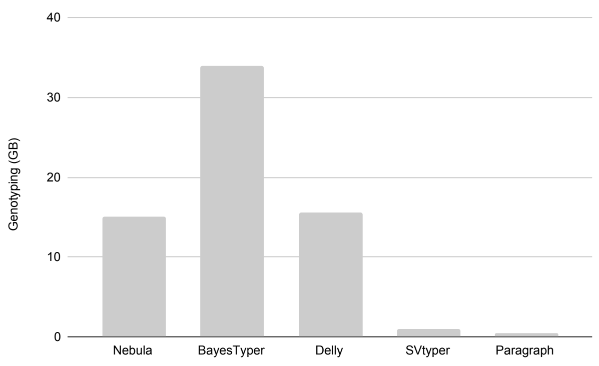
VCF files, Nebula only requires the SV coordinates (and optionally the inserted sequence for insertions). We also provide a Docker version of Nebula that can be easily deployed to various cloud computing platforms such as Cancer Genomics Cloud [77].

Nebula also has advantages when genotyping mapped samples. For a mapping-based genotyper, the sequencing reads should be mapped against the same reference genome version that the SV coordinates are from; however, once Nebula has extracted *k*-mers for a set of SVs reported against a certain reference genome (e.g GRCh38), it can genotype them on samples mapped to other reference genome versions (e.g GRCh37) directly and without the need to remap the samples or lift SV coordinates.

Due to a heavily optimized implementation and the fact that Nebula only counts *k*-mers that are associated with SVs, our *k*-mer counting stage is an order of magnitude faster than that of BayesTyper and uses less than half of the memory. For our experiments with 1KG data in Section 2.3.2, Nebula counts a total of 6.7M genotyping *k*-mers on NA19240 plus an additional



(a) Runtime



(b) Peak Memory Usage

Figure 2.13: Comparison of single-thread runtime (a) and peak memory usage (b) of Nebula and other genotyping tools while genotyping 12321 insertion and deletion SVs on unmapped NA19240 reads. Nebula and BayesTyper are *k*-mer-based methods and don't require read-mappings. Delly and SVtyper mainly parallelize over the number of input samples and don't benefit from multiple threads when genotyping a single sample. Peak memory usage excludes the memory usage of BWA-mem (peak memory usage of BWA-mem mapping was 16GB).

600,000 *k*-mers for estimating GC-corrected coverage. In our experiments Nebula can count *k*-mers at upwards of 500,000 reads per second using a single Xeon processor core.

2.4 Discussion

We have presented Nebula, a novel approach for ultra-efficient and accurate genotyping of any type of SV without the need to map the reads to the reference genome. We have demonstrated that *k*-mers can act as a lightweight and simple alternative for expensive mapping-based methods to genotype polymorphic SVs. Although several tools have already achieved similar conclusions for other types of variants such as SNVs [47, 48] and indels [49, 66], Nebula was the first mapping-free method for genotyping SVs known to us at the time of its publication.

Our proposed approach can easily be modified to genotype other types of variants (i.e., SNVs and indels) by selecting more *k*-mers. Thus, we believe that utilizing a combination of these mapping-free methods can provide a framework for accurate and efficient genotyping all types of variation using *k*-mer counts. This would significantly reduce the computational resources needed to analyze new WGS samples and will speed-up large scale studies.

Note that Nebula does not require exact SV breakpoints for genotyping SVs and can work with approximate breakpoints. This is an advantage over approaches that require exact breakpoints or assembled haplotypes to guide the *k*-mers selection and accurate variant genotyping. Furthermore Nebula only counts the *k*-mers directly associated with the SVs, significantly reducing the runtime and memory usage compared to other *k*-mer based approaches.

Furthermore, genotype imputation algorithms [80] can be incorporated into Nebula's pipeline to improve the method's accuracy and ability to genotype variants that are difficult to genotype solely using *k*-mers, e.g. SVs with breakpoints in repeat regions of the genome.

Finally, extending Nebula to utilize non-unique *k*-mers that are shared between different SVs may help us improve our performance when genotyping SVs in repeat regions of the genome (e.g. tandem repeats).

2.5 Code and Data Availability

The code and data used in these experiments along with detailed usage documentation and instructions for reproducing the 1KG experiments is publicly available at https://github.com/Parsoa/Nebula.

Nebula was initially made available as a BioRxiv draft[2] in March 2019 and was later presented at Recomb-Seq 2019. A significantly improved version was published under the title "Nebula: Ultra-efficient Mapping-free Structural Variant Genotyping" in Nucleic Acid Research in January 2021, available at `https://doi.org/10.1093/nar/gkab025`.

2.6 Funding

This work is funded in part by the Sloan Research Fellowship FG-2017-9159 and UC-Davis funding to Fereydoun Hormozdiari.

[2]`https://www.biorxiv.org/content/10.1101/566620v1.full.pdf+html`

Chapter 3

Comparative Genome Analysis Using Sample-Specific String Detection in Accurate Long Reads[1]

3.1 Motivation

One of the main objectives of performing WGS is the comparison of two or more genomes. Comparative genomic studies are concerned with multiple individuals from the same or closely-related species, either in a case versus control setting or within the context of population genomics and evolution [57]. Discovery of variants between multiple samples using WGS is at the core of most such analysis.

The mapping-free approaches developed for comparative study of short-read sequencing data (e.g [81], [49], [71], [67], [68]) are mostly based on finding k-mers that distinguish one sample from other samples. The idea of computing k-mer that are unique to a target with regards to a background set of genomes was also proposed in [82]. In general, the length of k-mers(i.e., k) is a fixed constant and usually short. However, for long and accurate reads we are not limited by the length of the short reads and can select arbitrarily long k-mers if needed. This flexibility on length of sequences selected can be advantageous for comparative studying of repeat regions of the

[1]This chapter was published as *Parsoa Khorsand, Luca Denti, Human Genome Structural Variant Consortium, Paola Bonizzoni, Rayan Chikhi, Fereydoun Hormozdiari. Comparative genome analysis using sample-specific string detection in accurate long reads. Bioinformatics Advances, no. 1, 2021.*

genome. The tools mentioned above are fundamentally unable to deal with variable-length *k*-mers and therefore novel developments are needed to fully explore this direction.

We propose a novel method for comparative analysis of multiple WGS samples using accurate long-read sequencing data (e.g., HiFi reads from PacBio [83]), without the need to map the reads to a reference genome or choose a fixed *k* value. The advantages of utilizing flexible length strings (e.g., adaptive seeds) in pattern matching has been previously demonstrated [84].

The main novelty is the formulation and the resolution of a new computational problem concerned with enumerating sample-specific strings, while avoiding a combinatorial explosion due to the quadratic size of the set of potential candidates. We show that this approach enables identifying nearly all sequences spanning variants between two human genomes on actual PacBio HiFi data. Some of the applications of the proposed comparative genome analysis framework include finding *de novo* variants, sequences segregating in a pedigree, or markers distinguishing between populations (e.g., cases and controls).

3.2 Problem Definition

Consider two sets of strings: T (targets) and R (references). Here by *references* we mean either 1) a reference genome, or 2) a set of unassembled reads that are coming from an unknown reference genome, or 3) a heterogeneous set of reads and genomes that are taken together to be the reference pangenome of some population of interest. We are interested in enumerating substrings of the targets that do not appear as exact substrings of the references.

As a motivating example consider two individuals and their respective sets of sequencing reads T and R. We define a **variant** as a genomic event that can be described by a single line in the VCF format, such as a single nucleotide polymorphism (SNP), an insertion or deletion, or a SV such as a duplication, or a translocation. We define a ***de novo* variant** as a variant in the child genome, defined relative to some reference genome (e.g. hg38), that is not present in either the mother or the father genomes. *de novo* variants represent variants in the child genome that are not inherited from either parents.

More complex forms of genomic variation, e.g. inversion-duplications, can be seen as combinations of variants and therefore are not further considered here. The intuition is that for each variant, there should exist at least one substring of the genome of T spanning this variant that is not found within the genome of R. Indeed, the whole genome of T would be one such substring, but there also likely exist shorter strings than that. Translating this observation to reads, there should exist for each variant at least one substring of T that is not found in R.

We postulate, and will later experimentally verify, that with long and accurate enough reads virtually all variants can be found in substrings of T that do not appear in R.

We are now returning to the abstract formulation of our initial problem of finding substrings of the targets not found in the references. For two strings s and t, we will use the notation $s \sqsubset t$ to indicate that s is a substring of t (and $s \not\sqsubset t$ for s is not a substring of t). Formally we want to enumerate the set $\widetilde{S_T}$ of all strings s such that

1. there exists $t \in T$ where $s \sqsubset t$, and
2. for all $r \in R$, $s \not\sqsubset r$.

In the worst case, the size of $\widetilde{S_T}$ can be quadratic in the total length of strings in T, which is too large to be stored or even enumerated. Therefore we will instead seek a reduced set of strings S_T that can be seen as a minimal representation of $\widetilde{S_T}$ that do not consider strings having proper substrings in $\widetilde{S_T}$ (the **substring-free** property). This is formalized as the following problem:

Problem 1 (Substring-Free Sample-specific strings (SFS) extraction problem) *Let T and R be two sets of strings, targets and references, respectively. Let $\widetilde{S_T}$ be the set of all strings satisfying conditions 1 and 2 above. Return the largest subset $S_T \subset \widetilde{S_T}$ such that for all $s \in S_T$, there does not exist $s' \in \widetilde{S_T}$, $s' \neq s$, where $s' \sqsubset s$; i.e., S_T is the set of all strings from $\widetilde{S_T}$ for which no shorter string of $\widetilde{S_T}$ is substring of them.*

A string $s \in S_T$ is then called a T*-specific string* w.r.t. references R, or simply *specific string* when T and R are clear from the context. Furthermore, a T-specific string s that is a substring of $t \in T$, will be also called a t-specific string. In the following, we will sometimes omit recalling that T-specific (and t-specific) strings are substring-free.

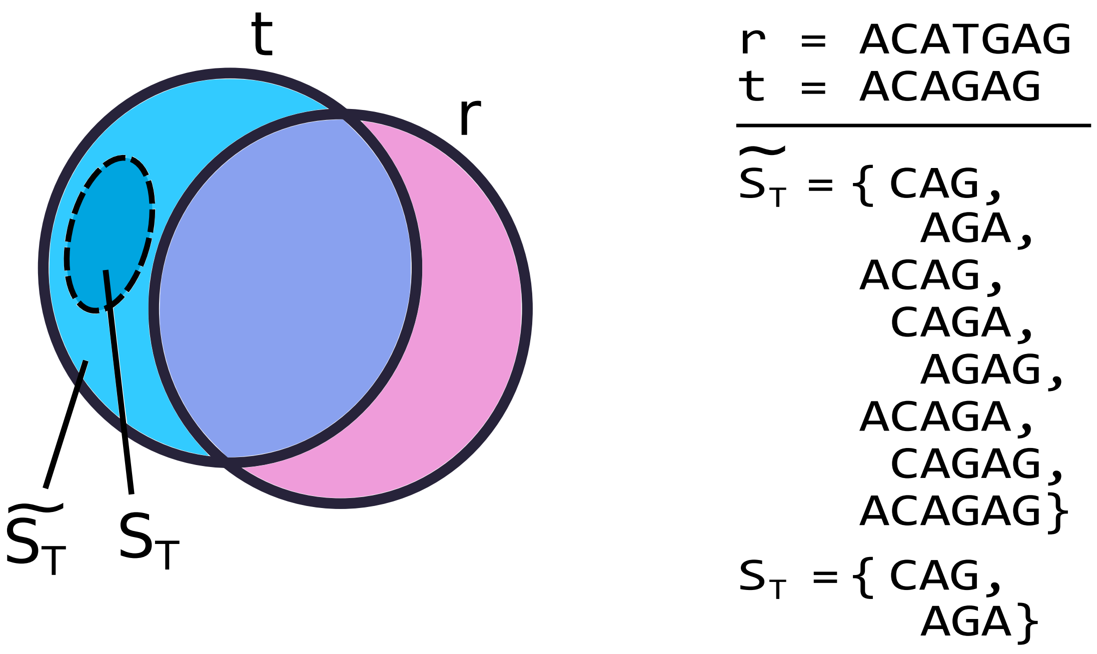

t
r
$\widetilde{S_T}$
S_T
r = ACATGAG
t = ACAGAG
$\widetilde{S_T}$ = { CAG, AGA, ACAG, CAGA, AGAG, ACAGA, CAGAG, ACAGAG}
S_T = { CAG, AGA}

Figure 3.1: **Illustration of the SFS framework.** Consider a target string t and a reference string r, each represented by a circle symbolizing all substrings. Blue area: substrings of t not in r; pink: substrings of r not in t; purple: substrings common to both t and r. We start by enumerating $\widetilde{S_T}$, consisting of all strings s that satisfy conditions 1 and 2 of Section 2 (i.e., s is a substring of t and not a substring of r). Then, the set S_T (result of SFS) is the largest substring-free subset of $\widetilde{S_T}$.

We will refer to Problem 1 as the "SFS problem", and an instance is illustrated in Figure 3.1. It is easy to see that SFS can be (inefficiently) solved in $O(n^3)$ worst-case time and $O(m^3)$ memory, where n and m are the total lengths of strings in T and R respectively. The set $\widetilde{S_T}$ can be constructed by enumerating all substrings of T and checking their membership in a hash table containing all substrings in R; then another pass over $\widetilde{S_T}$ constructs S_T in linear time and space over the total length of strings in $\widetilde{S_T}$, e.g., through indexing $\widetilde{S_T}$ using a FM-index. In this paper, we will propose a novel and more efficient quadratic-time $O(n^2)$ algorithm (Algorithm 1 in Section 3.3) using linear space $O(m)$ for solving the SFS problem. We will also propose a heuristic version of the algorithm that solves a relaxed variant of Problem 1 in linear time $O(n)$. All these complexities are on top of the FMD-index construction [85], which in our case can be done in $O(m)$ time and space [86].

The following property shows that it is sufficient to consider instances of the SFS problem where T is reduced to a single string.

Property 1 (Local substring-free property) *Let T and R be two sets of strings (targets and references, respectively). The set S_T of T-specific strings w.r.t. R, i.e., the solution of SFS problem, can be computed as the union of the sets S_t with $t \in T$, where S_t is the set of t-specific strings.*

For the sake of simplicity, assume that $T = \{t_1, t_2\}$. Let S_{t_1} be the set of t_1-specific strings obtained as a solution of the SFS problem on instance $(\{t_1\}, R)$ and similarly let S_{t_2} be the set of t_2-specific strings on instance $(\{t_2\}, R)$. We need to prove that given S_T the solution of the SFS problem on instance $(\{t_1, t_2\}, R)$, then $S_T = S_{t_1} \cup S_{t_2}$. Let us first observe that $S_T \subseteq S_{t_1} \cup S_{t_2}$ as indeed each string s in S_T must be a substring of t_1 or of t_2 and thus s is a t_1-specific string or is a t_2-specific string. Hence let us now prove that $S_{t_1} \cup S_{t_2} \subseteq S_T$. By construction, any t_1-specific string (as well as any t_2-specific string) is a substring of a string in T (condition 1) and it is not a substring of any string in R (condition 2). Moreover, strings in S_{t_1} (S_{t_2}, respectively) are substring-free in the sense that each string is not a substring of another one in the same set. We have to prove that any t_1-specific string x cannot be a substring of any t_2-specific string y, and vice versa (substring-free property). We will prove this by contradiction. Let us assume that x is a substring of y. By definition y is not a substring of R which implies that x is a substring of R: indeed y being substring-free, it holds that any substring of y is a substring of R. But x being a t_2-specific string, we obtain a contradiction. At this point, the vice versa is trivial to prove.

3.3 Algorithm for Sample-specific String Detection

3.3.1 Preliminary Concepts

The FMD index [85] is a data structure based on the FM-index [87] which indexes a set of strings and their reverse complements at the same time, allowing to perform search operations on the index.

The FM-index of the collection $\{r_1, \ldots, r_n\}$ of strings of sample R is essentially made of the BWT (Burrows Wheeler Transform) of R which is itself a permutation B of the symbols of R obtained from the Generalized Suffix Array (GSA) SA of R. Indeed, recalling that $SA[i]$ is equal to (k, j) if and only if the k-suffix of string r_j is the i-th smallest element in the lexicographic ordered set of all suffixes of the strings in R, then $B[i] = r_j[|r_j| - k]$, if $SA[i] = (k, j)$ and $k < |r_j|$, or

$B[i] = \$$ otherwise.

Given a string Q, all suffixes that have Q as a prefix appear consecutively in GSA, where they induce an interval $[b, e)$ which is called *Q-interval*. Note that the difference $e - b$, also called the width of the Q-interval is equal to the number of occurrences of Q as a substring of some string $r \in R$. The backward extension operation of an arbitrary character σ applied to the Q-interval of a string Q allows to determine the σQ-interval in the index. In particular, iteratively performing the backward operation on a pattern by searching the pattern backwards from its last symbol to its first symbol, allows to find all occurrences of the pattern inside the strings of the reference sample R in linear time in the size of the pattern.

The FMD index also allows to apply a forward extension operation of an arbitrary character σ to a Q-interval of a string Q to determine the $Q\sigma$-interval in the index. The implementation of both forward and backward operations in the FMD index is realized by constructing a FM-index for the collection R concatenated with the reverse-complement of each string in R. Differently from the bidirectional BWT [88] which builds two FM-indices, the FMD index builds a single FM-index for both strands.

By adopting the same notations as in [85], we keep a triple $[i, j, l]$ (called *bi-interval*) that encodes for the Q-interval $[i, i + l]$ and the $\bar{Q}$-interval $[j, j + l]$, where $\bar{Q}$ is the reverse complement of string Q. Whenever $l = 0$ the Q-interval (respectively $\bar{Q}$-interval) is empty and string Q (respectively $\bar{Q}$) does not occur in I_R. We will use notation $t[b : e]$ to denote an interval on string t, i.e., $t[b : e]$ is a substring of t, whereas $[i_b, j_b, l_b]$ to denote the corresponding $t[b : e]$-interval on the index I_R.

3.3.2 Ping-Pong Algorithm

We present Ping-Pong search (Algorithm 1), a novel algorithm to solve the SFS problem between a set of reference strings and a single target string $t \in T$. Our algorithm computes substring-free t-specific strings with respect to the reference sample R using the FMD index of R, from now on denoted I_R.

Note that based on Property 1, it is straightforward to extend the proposed algorithm to solve the SFS problem between a set of reference strings and a set of target strings (i.e., T). We will give

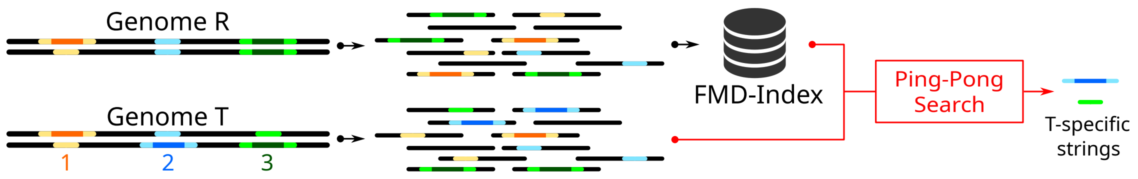


Figure 3.2: Two genomes R and T are depicted. With respect to genome R, site 1 has no variation in T, site 2 is an heterozygous insertion in T, and site 3 is an heterozygous deletion in T. Our pipeline aims to detect T-specific strings by (a) indexing the reads sequenced from R with a FMD index and (b) analyzing the reads sequenced from T with our novel *Ping-Pong search* algorithm. We note that, for ease of presentation, we depict at the end of the pipeline a single T-specific string per site even though multiple T-specific strings may actually be reported for each site.

more details on this at the end of this section.

The following main property which is a direct consequence of the substring-free property of specific strings is used to define the generic iteration step of the Ping-Pong algorithm.

Lemma 1 *Let R be a collection of strings with FMD index I_R and let t be a string that does not exist in R. Let x be the rightmost t-specific string currently found in t, where $x = t[b_x : e_x]$. It must then be the case that any other t-specific string will begin before b_x. Assume such a specific string y exists and starts at b_y, it must then be the case that y is the shortest prefix of $t[b_y : e_x - 1]$ that does not occur in the index.*

By definition, two specific strings cannot start at the same position as one cannot be a substring of the other. Thus given x the rightmost occurrence of a specific string in t, the second rightmost occurrence y of a specific string must start to the left of b_x, i.e., given $y = t[b_y : e_y]$ it must be that $b_y < b_x$. By the substring-free property $t[b_y : e_x]$ will not occur in the index as it contains the substring x which does not occur in the index. On the other hand it must be that $e_y < e_x$ otherwise y includes x as $b_y < b_x$ which is not possible by definition of x and y as substring-free specific strings. Thus $e_y < e_x$ which implies that y is a prefix of $t[b_y : e_x - 1]$. Now, y must be the shortest such prefix not in the index, otherwise it includes another specific string contradicting the substring-free property, thus concluding the proof of the Lemma.

Based on the previous Lemma, given the interval $[b_x : e_x]$ of the last detected specific string, the algorithm will start looking for a new occurrence of a specific-string from the end position

Algorithm 1: The Ping-Pong algorithm for computing *t*-specific strings

```
1  Function PingPongSearch($t$, $I_R$)
2      $b \leftarrow |t| - 1$
3      $[i, j, l] \leftarrow init(I_R, t[b])$
           // init function initializes a FMD index bi-interval representing a single
        character
4      while $b \geq 0$ do
5          while $l \neq 0 \wedge b > 0$ do                    // Step 1 - Backward extension
6              $b \leftarrow b - 1$
7              $[i, j, l] \leftarrow backwardExtension(I_R, [i, j, l], t[b])$
8          if $l \neq 0 \wedge b = 0$ then return
9          $e \leftarrow b$
10         $[i, j, l] \leftarrow init(I_R, t[e])$
11         while $l \neq 0$ do                                 // Step 2 - Forward extension
12             $[i_b, j_b, l_b] \leftarrow [i, j, l]$
13             $e \leftarrow e + 1$
14             $[i, j, l] \leftarrow forwardExtension(I_R, [i, j, l], t[e])$
15         Output $t[b : e]$
16         $[i, j, l] \leftarrow [i_b, j_b, l_b]$
```

$e_x - 1$. More precisely, the algorithm keeps track of two search positions b and e inside t which respectively represent the start and end of a substring of t that may or may not exist in I_R and uses the constant-time forward and backward extension operations defined on the FMD index [85].

Given the index I_R and a triple $[i, j, l]$ encoding a Q-interval and $\bar{Q}$-interval, the algorithm alternates between extending the Q-interval backward (step 1, lines 5 to 7) and forward (step 2, lines from 11 to 14) to find t-specific strings. Figure 3.3 illustrates how the algorithm iterates over an input string t.

During each iteration of step 1, the algorithm backward extends the $t[b : e]$-interval of I_R with $t[b - 1]$ until the backward extension in the index I_R with $t[b - 1]$ is not possible. In other words, this is equivalent to finding the left maximal match ending at position e and extending it one base on the left. Now $t[b - 1 : e]$ is a substring of t that is specific to t. However, such a substring is not necessarily the shortest, since one of its prefixes may also be specific.

Step 2 initializes e to $b - 1$ and then keeps incrementing e by one position at a time, and performs a forward extension in I_R for the prefix $t[b - 1 : e]$ for each increment. If the forward extension with $t[e + 1]$ is not possible in I_R, the algorithm stops and returns $t[b - 1 : e + 1]$ as the shortest string beginning from position $b - 1$ that's not in I_R. In other words, we are looking for the longest right maximal match starting at position $b - 1$ and then we are extending it one base to the right. We note that Algorithm 1 outputs substring $t[b : e]$ since b (resp. e) has been already decremented (resp. incremented) previously in the corresponding `while` (i.e., step 1 for b and step 2 for e).

Finally, since substring $t[b - 1 : e]$ is not t-specific and is in the index, it could be extended to the left to compute a new t-specific which will eventually overlap the last computed t-specific $t[b - 1 : e + 1]$. Line 16 initializes this process. Observe that the Ping-Pong algorithm may compute the same SFS multiple times when processing a string t, however, the output is still a set of t-specific strings without duplicates.

Theorem 1 *Ping-Pong algorithm solves the SFS problem for a string t w.r.t. a reference set R in time* $\sum_{s \in S_t} O(|s| \times occ_s + |t|)$*, where* occ_s *is the number of times a string s is output by Algorithm 1 when processing t.*

We start by proving correctness and then time complexity. Based on Lemma 1, the algorithm searches for a new specific string starting from the end position e_x of the last detected specific string x. The correctness relies on the fact that the Ping-Pong algorithm visits from right to left each position b of the prefix of length e_x of the input string t maintaining the following invariant property: the algorithm outputs the shortest prefix $t[b : e]$ of $t[b : e_x - 1]$ which does not occur in the index I_R (if such a string exists).

Based on Lemma 1, this invariant property allows us to state that the algorithm for any position b outputs the t-specific string starting at that position (which is unique by the substring-free property) if any; since all positions of the input string are processed by the algorithm, all possible specific strings are output in the end. We now show the invariant by analyzing a single iteration.

Assume that b is a position such that $t[b : e]$ is a t-specific string computed when the algorithm visits such a position of t. Now, let k be the smallest integer (with $k < b$) such that $t[b-k : e-1]$ is the next string x not in the index. This is easily detected by backward extension, i.e., by iterating k times the loop from line 5 to 7 of the algorithm. After finding k, the algorithm sets $k' = 0$ and computes whether $t[b - k : b - k + k']$ is in the index for increasing values of k' and stops as soon as $t[b - k : b - k + k']$ is not in the index thus computing the shortest prefix of $t[b - k : e - 1]$ not in the index. This concludes the proof of the invariant.

To prove Ping-Pong algorithm's time complexity, observe that it performs a number of backward extensions which is equal to the length of the string t, while it performs a number of forward extensions that is $O(l_b)$ for l_b being the length of the specific string retrieved from position b of t. Thus the time complexity easily follows from the above observation.

3.3.3 Relaxed Ping-Pong Search: A Faster Heuristic Search Algorithm

Observe that by Theorem 1, the worst case time required to solve the SFS problem on a single string t is $O(n^2)$ for n being the length of the string t, assuming that the index I is already available. Note that in the formula $\sum_{s \in S_t} O(|s| \times occ_s + |t|)$, $|s|$ can be $O(n)$ in the worst case and $\sum_{s \in S_t} O(|s|)$ can achieve the bound of $O(n^2)$ since the strings in S_t span positions of the string t that are overlapping and we can have $O(n)$ strings in S_t each of length $O(n)$. See Section 3.3.5 for an example. This clearly implies a quadratic time for solving the SFS problem when the input is no longer a single string t but a collection T of strings of total length n.

In order to speed-up the computation of the SFS problem we consider a simple variant of Ping-Pong algorithm that leads to a linear time complexity by avoiding the computation of specific strings that occur in overlapping positions of the original string t. The variation is simply obtained from the pseudo-code of Algorithm 1 by deleting instruction 12 and replacing line 16 with the instruction $[i, j, l] \leftarrow init(I_R, t[b-1])$. This implies that the search procedure of t-specific strings starts from one position to the left of the beginning of the last detected string in t. We call this procedure the *relaxed Ping-Pong Search*.

Observe that the relaxed version may compute t-specific strings which have non-empty sequence overlaps, they however occur in non-overlapping positions of the read t.

It is easy to verify that the relaxed version of Ping-Pong algorithm is linear in the size of string t. Indeed, in the worst case it performs two index queries per symbol of the input string: each character is searched in the index one time during the backward extension and one time during the forward extension (see Figure 3.3). Formally, when estimating the formula $\sum_{s \in S_t} O(|s| \times occ_s + |t|)$ of Theorem 1 in this variant, strings in S_t occur in positions of t that are disjoint and thus in the worst case the sum of the sizes of strings in S_t is $\sum_{s \in S_t} O(|s| \times occ_s) = O(|t|)$, thus proving that the time complexity of the algorithm is linear in the size of the input string.

A more detailed visual comparison of the relaxed and exact versions of the algorithm can be seem in Figure 3.3.

3.3.4 Relationship Between Edit-distance and the Relaxed Algorithm

The edit-distance is a well known measure in the comparison of two genome sequences. By counting the minimum number of nucleotide insertions, deletions and changes that transform a genome t into r, the edit distance between t and r, denoted by $D(t, r)$ is clearly an upper bound for the number of positions with variations in t w.r.t. to r. In the following we show that for a pair of strings t and r, each t-specific string returned by the relaxed version of Ping-Pong algorithm corresponds to at least one edit operation that changes t into r, thus showing that $D(t, r)$ is an upper bound on the size of its output set. Observe that the relaxed version of the algorithm computes a subset of the T-specific strings w.r.t. R that has the substring-free property.

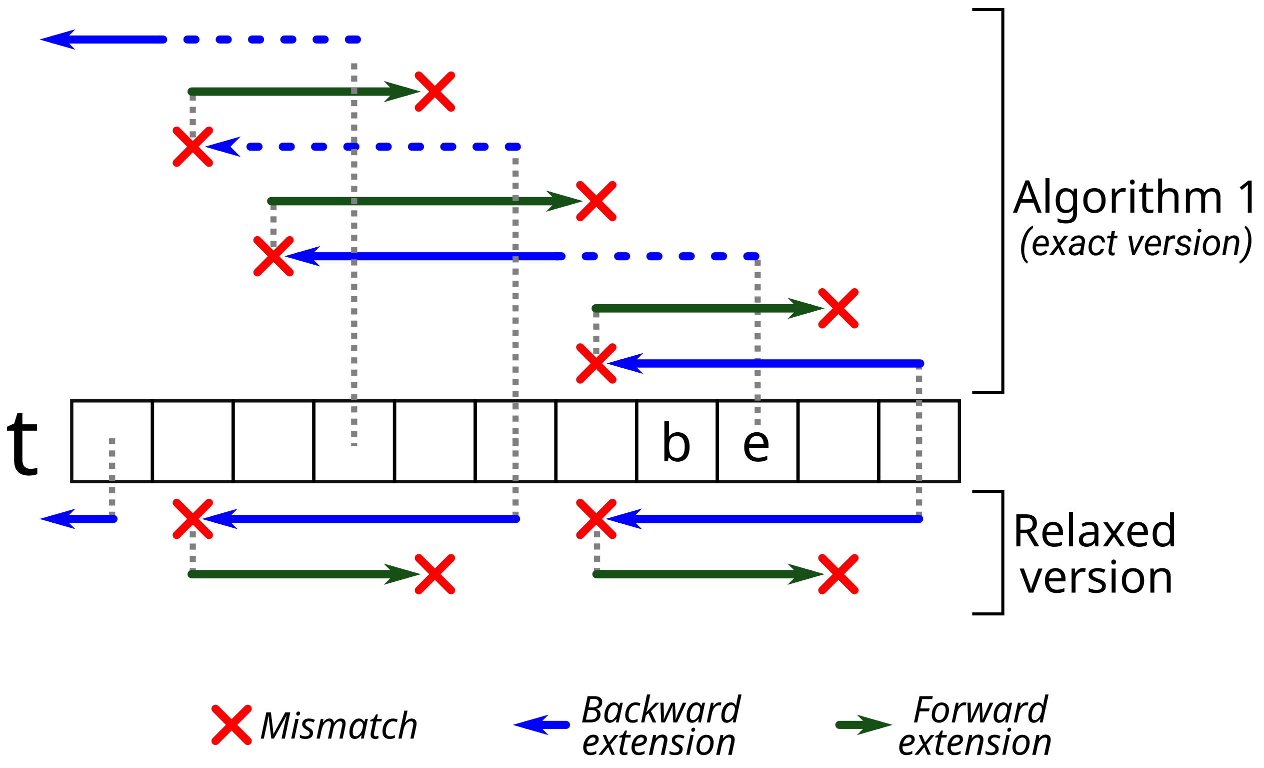


Figure 3.3: **The Ping-Pong search algorithm** (top) starts from the end of the input string *t* and alternates between backward and forward extensions. When the backward extension (blue arrows) ends due to a mismatch (red cross), the algorithm starts a forward extension (green arrows) until another mismatch is found. After a single iteration (outer `while` loop of the pseudocode), a *t*-specific string $t[b-1 : e+1]$ is found and the algorithm restarts the search from position e, allowing solutions to "overlap" on *t*. A dashed blue line represents bi-intervals that were already computed during a forward search (and therefore not recomputed in the next iteration). In the relaxed version of the algorithm (bottom side), solutions cannot overlap and the search restarts from position $b-2$ instead of e. We note that Algorithm 1 outputs substring $t[b : e]$ since b (resp. e) has been already decremented (resp. incremented).

Theorem 2 *Given two strings t and r, $|S_t|$ the size of the set of strings S_t returned by the relaxed Ping-Pong search with respect to r, then $|S_t| \leq D(t, r)$.*

Since the set S_t consists of strings induced by non-overlapping intervals of sequence t, any edit operation changes $|S_t|$ by at most 1. The minimum set of edit operations to convert t to r (i.e., $D(t, r)$) will transform $t = t_0$ into successive strings $t_1, t_2, \ldots$ and eventually $t_{D(t,r)} = r$. For each operation, the successive sets of relaxed Ping-Pong strings for $t_2, \ldots$ change in cardinality by at most 1, i.e. $||S_{t_i}| - |S_{t_{i+1}}|| \leq 1$ for $0 \leq i < D(t, r)$. Observe that $D(t, r) = 0$ implies $|S_t| = 0$ thus $|S_{t_{D(t,r)}}| = 0$. Thus, total size of $|S_t|$ could not have been more than $D(t, r)$ to start.

3.3.5 Example of a Ping-Pong Search in Quadratic Time

An example of string where the Ping-Pong algorithm works in quadratic time in the length of the input target t is given by the string $TT\{ACG\}^n$ while the reference R consists of the unique string $TT\{ACG\}^{\frac{n}{2}}$. Observe that $CG\{ACG\}^{\frac{n}{2}-1}A$, $G\{ACG\}^{\frac{n}{2}-1}AC$ are specific strings and each of them occurs $\frac{n}{2} - 2$ times in the target sequence. Each of these strings will be extracted $\frac{n}{2} - 2$ times when running the exact Ping-Pong search whereas they will be only selected once with the relaxed version of the algorithm.

3.3.6 Implementation Notes

We implemented Algorithm 1 in C++ based on code from `ropeBWT2` [89]. The implementation is deeply parallelized using Open-MP [73]. Parallelization is critical for achieving reasonable runtimes as the Ping-Pong algorithm is indeed very computationally intensive, in particular for the exact case.

After creating the index of the reference set R, our code executes the Ping-Pong algorithm on each target string $t \in T$ while also keeping track of the number of times each specific string is seen (Figure 3.2). As each target string can be processed independently, our code is embarrassingly parallel. Once all target strings have been analyzed, a post-processing step combines the smaller solutions into the final solution of the SFS problem. In order to remove specific strings produced by sequencing errors when our method is run on WGS data, the post-processing step can filter out all the specific strings occurring less than τ times, with τ being a user-defined cutoff. To efficiently perform this step, the implementation stores the specific strings along with the number of times they occur in the input sample in a hashtable.

3.4 Experimental Results

We use experiments on both simulated and real data to show the effectiveness of the Ping-Pong algorithm and the SFS concept as a tool for comparative SV discovery. The following sections detail these experiments.

3.4.1 Specific String Detection in Simulated Human HiFi Trio

We used simulations to test the performance of our proposed method in detecting *de novo* SVs in WGS trios (i.e., proband, mother and child). We mutated the GRCh38 genome randomly with 6115 insertions and deletions from the 1KG project [33] to produce two haplotypes for each parent. We limited the simulations to chromosomes 1-5. We then simulated the child genome by inheriting variants from the parents and considering recombination inside each chromosome. Finally, we introduced an additional 17,595 randomly-generated *de novo* SVs equally divided between insertions, deletions and inversions into the child genome impacting 7,913,593 base-pairs.

We simulated reads from the father, mother and child genomes at different coverage levels (5x, 10x, 20x and 30x) for each haplotype using `PBSIM` [90] with sequencing error rate and read length distribution similar to real HiFi data. Specifically, we sampled these parameters from the HGSVC2 PacBio HiFi reads for the HG00733 sample [91] with the error rate averaging at 0.1%. All three samples were error corrected using `ntEdit` [92] to remove sequencing errors. The combined reads of the father and mother were indexed using FMD-index and we searched for child-specific strings using Ping-Pong algorithm (exact version).

We measured the accuracy of the method using two metrics of **recall** and **precision**. Recall is defined as the percentage of *de novo* variants that are covered with child-specific strings and precision is defined as the percentage of child-specific strings that cover a *de novo* variant. We test the performance of the method for different τ cutoff values ($2 \leq \tau \leq 6$) to study the relationship between this parameter and sequencing coverage levels and to measure our method's sensitivity (Figure 3.5). While the high coverage simulations (30x, 20x and 10x) have constantly high recall rates regardless of τ, the low-coverage 5x sample's recall drops significantly with larger cutoff values.

We analyzed the child-specific strings from the 30x simulation using $\tau = 5$ in more detail. A total of 14,381,350 child-specific strings were retrieved with 2,052,144 remaining after filtering low-abundance strings. The selected child-specific strings achieved > 98% recall and 82% precision at recovering simulated *de novo* SVs. To better demonstrate the usefulness of child-specific strings, we compared the performance of the strings generated using both the exact and relaxed versions of Ping-Pong algorithm on the 30x simulation against child-specific *k*-mers of fixed sizes 32bp and 101bp with abundance of at least 5. Child specific *k*-mers were calculated using `KMC3` [93] by subtracting the set of parent *k*-mers from the set of child *k*-mers. We calculated precision and recall by mapping the *k*-mers and SFS strings to the child haplotypes with `BBMap` [94]. We observe that SFS consistently performs better than fixed-length *k*-mers. The results can be seen in Table 3.1.

We further analyzed the qualities of the alignments of child-specific strings against all three genomes in the trio (Figure 3.4). Alignment quality is evaluated based on the number of bases that do not match. More than 83% of child-specific strings map perfectly to the child genome, and no (zero) string has a mismatch-free mapping to either parent genomes, indicating that the strings are truly child-specific.

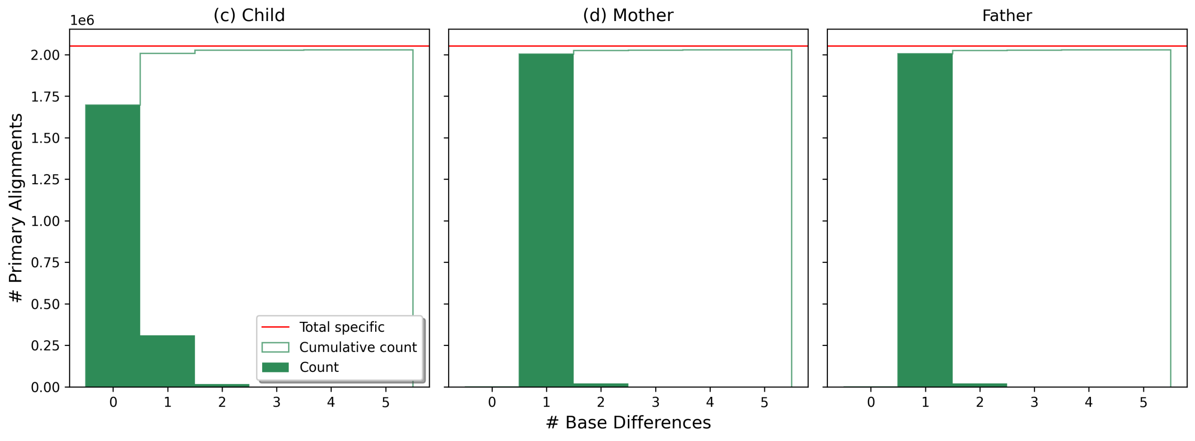


Figure 3.4: Comparison of the quality of SFS alignments in the 30x simulated trio with $\tau = 5$.

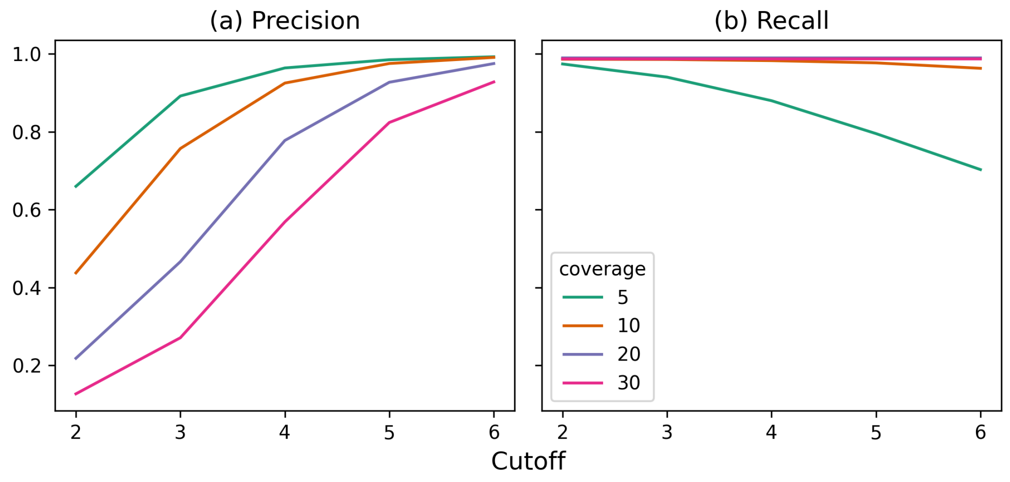


Figure 3.5: Precision (a) and recall (b) calculated for different coverage levels (5x, 10x, 20x and 30x per haplotype) and cutoff values $2 \leq \tau \leq 6$ in simulation.

		Overlapp(ing/ed)	Total	%	Metric
SVType	INV	5648	5768	97.91%	**Recall**
	DEL	5774	5879	98.21%	
	INS	5945	5947	99.96%	
Child-specific strings		1,690,675	2,052,144	82.38%	**Precision**

Table 3.1: Summary of simulation results.

Finally, we re-ran the simulation at 30x coverage without incorporating any sequencing errors in the trio. In this scenario, the simulated SVs are the sole source of novel sequences in the child compared to the parents and therefore we expect every recovered SFS to cover a variant. Analyzing the 1,720,395 child-specific strings retrieved in this scenario indeed yields a precision of 100.0%. However, the recall remains the same as in the case with sequencing errors, at 98.70%. This is because some variants don't produce novel sequences and thus cannot be captured with our approach.

3.4.2 Specific String Detection in Real Human HiFi Data

We performed an extensive evaluation of sample-specific strings using real HiFi data to assess their ability to compare two individuals of different populations. We considered the HG00733 child (Puerto Rican trio) and the NA19240 child (Yoruba trio). For both these individuals, the HGSVC2 [91] provides a PacBio HiFi 30x sample. Figure 3.6 reports the length distribution of the considered samples.

After correcting both samples with `ntEdit` [92], we indexed the NA19240 sample and we searched for HG00733-specific strings (from now on we will refer to these strings simply as ‘specific’) using both the exact and the relaxed version of our algorithm.

Table 3.2 reports the running times and the peak memory usage of our pipeline; the creation of the FMD-Index was the most time-consuming step.

Based on the results on simulated data (Figure 3.5) and the coverage of the two samples (30x), we considered all specific strings occurring more than 5 times. The main goal of this post-filtering is to remove from downstream analyses specific strings that are with high probability the result of sequencing errors. Using the exact (relaxed, respectively) version of our algorithm we retrieved 34,219,149 (7,125,436, respectively) strings. Figure 3.9 reports information on the lengths and the abundances of these strings. As expected, the exact version of our algorithm is slower and retrieves more strings than the relaxed one.

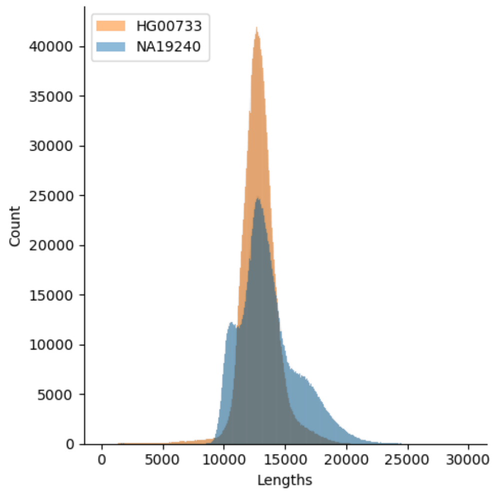


Figure 3.6: Read length distribution for HG00733 and NA19240 samples.

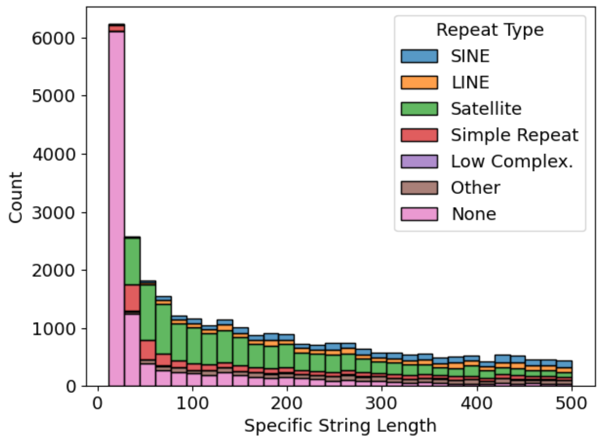


Figure 3.7: RepeatMasker classification of HG00733-specific strings mapping perfectly to NA19240 contigs.

3.4.3 Contigs-based Analysis

We first analyzed the quality of HG00733-specific strings by checking whether they are effectively specific to the HG00733 child. To do so, we aligned the strings to the contigs provided by the HGSVC2 consortium of the two individuals and we counted base differences (substitutions, insertions, deletions, and clips) within alignments. We mapped strings shorter than 500bp with `BBMap` [94] and longer ones with `minimap2` [94]. We used two different aligners since `BBMap` showed higher sensitivity in mapping short (< 50bp) strings. Figure 3.8 (a/b) shows the results of this analysis for the exact version of our algorithm (see Figure 3.10 for the relaxed results).

A total of 33,964,009 specific strings were mapped to the HG00733 contigs and 27,326,747 (80%) of these were aligned perfectly, i.e., without any base difference. On the other hand, 33,932,307 specific strings were mapped to the NA19240 contigs but only 158,094 (0.4%) of these were aligned perfectly.

Note that in principle we do not expect HG00733-specific strings to align perfectly to the NA19240 contigs. To investigate the small percentage of strings that nevertheless do, we screened them with RepeatMasker [95] for interspersed repeats and low complexity DNA sequences. In both scenarios, RepeatMasker masked ~70% of the considered bases (Figure 3.7), hinting that they may be the product of repeat-induced incorrect assembly or alignment.

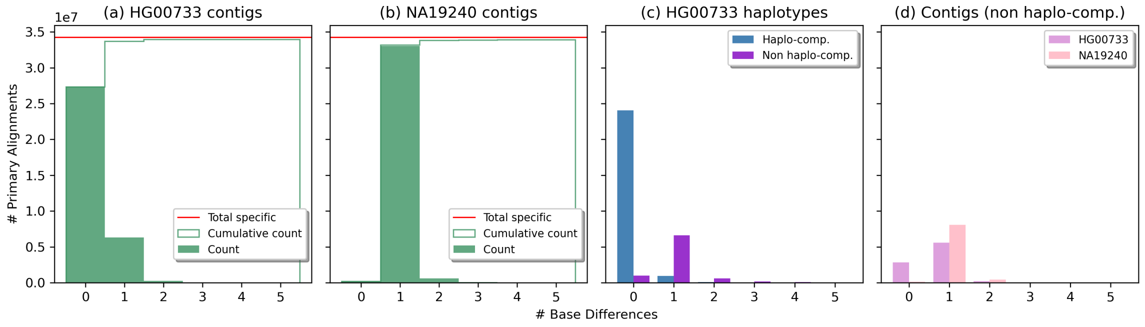


Figure 3.8: Results on exact HG00733-specific strings. Panels (a, b): comparison of the quality of specific strings alignments computed against the HG00733 contigs (a) and the NA19240 contigs (b). Panel (c): comparison of the qualities of the specific string alignments representing (Haplo-compatible) and not representing (Non haplo-compatible) a specific portion of HG00733 haplotypes. Panel (d): comparison of the qualities of non haplo-compatible specific string alignments computed against the HG00733 contigs and the NA19240 contigs. Quality is expressed as number of base differences (mismatches, insertions, deletions, and clips).

To summarize the results of this contigs-based analysis, we introduced the **C-precision** (contigs-based precision) metric. Based on the alignments to the contigs, it computes the fraction of HG00733-specific strings that align perfectly to HG00733 contigs and not perfectly to NA19240 contigs. Out of 27,326,747 specific strings aligned perfectly to HG00733 contigs, 132,031 aligned perfectly also to NA19240 contigs. The exact version of our algorithm therefore achieved a C-precision of 79.47%. On the other hand, the relaxed version achieved a C-precision of 90.61%. This was expected since the relaxed version of our algorithm retrieves a lower number of specific strings easily achieving a higher precision at the expense of, as we will see in the next section, a lower recall (see Table 3.3).

This analysis shows that the strings output by our algorithm are effectively specific to the HG00733 and may be effectively used to characterize differences between the two individuals.

3.4.4 Haplotypes-based Analysis

We evaluated the effectiveness of HG00733-specific strings in covering variant alleles that are specific to the considered individual. To do so, we considered the phased callset provided by the HGSVC2 consortium [96] and, after filtering out overlapping variations, we extracted for each variation and for each haplotype the set of alleles that are present in the HG00733 child but not

		Error Correction	NA19240 Indexing	HG00733-specific Retrieval	Total Time	Peak RAM
Ping-Pong	Exact	05:03	20:30	11:59	37:32	242
	Relaxed			03:01	28:34	32
k-mers	31-mers		-	03:25	08:28	12
	101-mers			03:10	08:13	24

Table 3.2: Running time (hh:mm) and peak memory (GB) of our pipeline and the *k*-mers pipeline (on real data). We used 16 threads where possible.

in the NA19240 (we will refer to these alleles as *HG00733-specific* or simply *specific alleles*). Therefore, each variation may have 0, 1 or 2 specific alleles. For instance, if a variation has genotype 0|2 in the HG00733 child and 1|1 in the other child, we considered alleles 0 and 2 as specific to the HG00733.

Table 3.3 (column *Total*) reports the number of specific alleles we considered in our analysis. We classified each allele with respect to the type of its originating variant (following the classification in [96]): SNPs, indels (insertions and deletions of 1-49 bp), and SVs (insertions and deletions of $\geq$ 50bp), which include copy number variants and balanced inversion polymorphisms.

Considering the entire set of known variations, we built the haplotypes of the HG00733 individual using `BCFtools` and then we aligned the HG00733-specific strings (occurring more than 5 times) to them using `BBMap` (strings $\leq$ 500bp) and `minimap2` (strings $>$ 500bp). Finally, we used `BEDtools` [97] (*intersect* sub-command) to find the overlaps between the alignments and the considered alleles.

We evaluated the quality of our specific strings in terms of **recall**, i.e., number of specific alleles effectively intersected by at least one alignment, and **H-precision** (Haplotype-compatible precision), i.e., the number of specific strings representing a specific portion of a haplotype of the HG00733 child. By “specific portion” we mean a subsequence of a HG00733 haplotype induced by a set of variations that is different from the subsequence of any NA19240 haplotype induced by the same set.

Table 3.3 reports the results of this analysis. We introduced the H-precision measure since close alleles (especially SNPs) on a haplotype of one individual may result in a specific string even when neither alleles are specific.

Indeed, a set of close alleles may be shared between two individuals but in one individual they may be on the same haplotype whereas in the other one on different haplotypes. Consider for instance two nearby variants with genotypes 0|1 and 0|1 in one individual and 1|0 and 0|1 in the other. In this case, the haplotype containing both 1 alleles on the first individual is specific to the first individual even though the single alleles themselves are not.

Remarkably, the set of specific strings computed by our method (exact version) intersect most of the HG00733-specific alleles ($> 98\%$), covering nearly all alleles coming from SNPs and indels ($> 98\%$ and $> 95\%$, respectively) and most of alleles coming from SVs ($> 92\%$).

We observed that a majority of the variants not covered by the sample-specific strings were indels in stretches of A or T sequences, likely addressable through improvements in homopolymer error correction.

Out of the 34,219,149 specific strings retrieved by the exact version, 73.43% of them represent a specific portion of the HG00733 haplotypes (H-precision). Figure 3.8 (c) reports the comparison in terms of base differences between the alignments representing specific portions of the haplotypes (denoted as "haplo-compatible") and those that do not (denoted as "non haplo-compatible"). As expected, the vast majority of the haplo-compatible strings are aligned perfectly to the haplotypes whereas the vast majority of non haplo-compatible strings are aligned with errors.

To better investigate why $\sim 27\%$ of the specific strings align well to the HG00733 haplotypes but do not represent a specific portion of them (accordingly to the considered VCF), we aligned those strings to the contigs of the two individuals. Figure 3.8 (d) reports the results of this analysis. 2,885,356 strings were aligned perfectly to the HG00733 contigs whereas only 208,330 were mapped perfectly to the NA19240. Moreover, $\sim$ 1.8 million specific strings align perfectly to the HG00733 contigs but not to its haplotypes.

This shows that even though the specific strings we computed do not represent a specific portion of the HG00733 haplotypes (accordingly to the VCF), they are effectively specific to that individual, confirming their capability in characterizing an individual even in those regions hard to call and analyze. This leads us to conjecture that a portion of those strings correspond to true variants missing from the VCF.

Results on strings retrieved by the relaxed algorithm follow the same trend (Figure 3.10). They however achieve higher H-precision and lower recall than the exact version (see Table 3.3), likely due to a lower number of strings returned.

Moreover, the relaxed algorithm may fail in covering close variations: if two variations are too close to each other, strings retrieved by our relaxed algorithm may cover only the right-most variation (due to its right-to-left traverse of input strings). See Figure 3.10 for an example.

To put our results in perspective, we compared them with a *k*-mer method. Similarly to a HG00733-specific string, a *HG00733-specific k-mer* is a *k*-mer occurring in the HG00733 sample and not in the NA19240. To compute the set of specific *k*-mers we first counted all *k*-mers occurring more than 5 times in the two samples independently with `KMC3` [93] and then we retrieved the *k*-mers present only in the HG00733 sample by subtracting the two sets (`kmc_tools` *kmers_subtract* operation). A total of 97,975,734 HG00733-specific *k*-mers ($k = 31$) were retrieved. We then mapped those to HG00733 haplotypes with `BBMap` and evaluated their recall and H-precision similarly to HG00733-specific strings.

Table 3.3 reports the results of this analysis. HG00733-specific 31-mers achieved lower recall and H-precision than HG00733-specific strings, although their computation is faster (8h for *k*-mers vs 28–37h for Ping-Pong, see Table 3.2). The poor performance of 31-mers can be explained by their length: a 31-mer located at a variant position might occur elsewhere in the genome, whereas a longer string would be unique.

We note that long ($> 500bp$) HG00733-specific strings retrieved by the exact algorithm cover $\sim 1.5\%$ of indels and SVs not covered by shorter ones, proving that longer strings are sometimes needed to effectively cover a variation.

For this reason, we also performed an analysis using longer *k*-mers ($k = 101$). A total of 387,221,925 101-mers were retrieved. However `BBMap` failed to align that many *k*-mers in reasonable time. We therefore aligned them with `BWA-MEM` [98] and computed their recall and H-precision. Results of this analysis can be found in Table 3.3.

Thanks to their length, 101-mers are able to cover more variations than 31-mers but not as many as our (exact) specific strings which are of variable length, sometimes longer than 101bp. For instance, Figures 3.11 and 3.12 show two examples of variants covered by specific strings and not by specific 101mers, highlighting the biological usefulness of our method.

Moreover, 101-mers are less precise than (exact) HG00733-specific strings: indeed, due to their overlapping nature, a false variant (e.g., a sequencing error) will in the worst case yield 101 false specific 101-mers.

We therefore mapped the specific *k*-mers to the contigs of the two individuals and we computed their C-precision (fraction of specific *k*-mers mapping perfectly only to HG00733 contigs). Similarly to specific strings, C-precision of 31-mers and 101-mers is higher then their H-precision (see Table 3.3), proving one more time that the considered VCF may be incomplete.

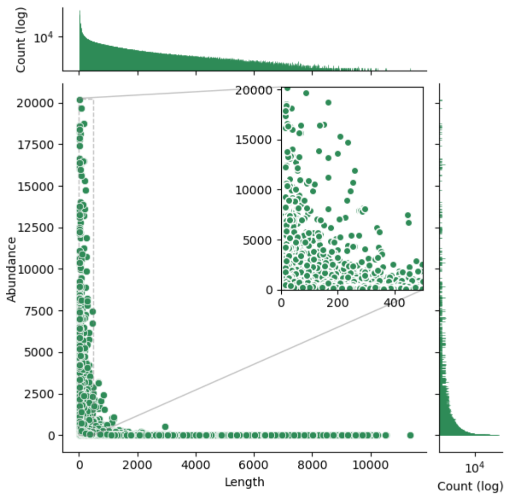
Finally, in an attempt to reduce the number of strings obtained using the *k*-mer method, we assembled the 31-mers and the 101-mers into unitigs (which correspond to maximally extending *k*-mers using their $(k-1)$-overlaps and stopping at any variation) using `BCALM2` [99] and we computed their recall and H-precision. Results of this analysis, can be found in Table 3.3. Surprisingly, assembling the *k*-mers into unitigs did not improve their overall accuracy.

The explanation behind these results must be sought in the incompleteness of the considered VCF. Indeed, the considered VCF takes into account only 75% of the entire genome: variants are called only in those regions in which all the haplotypes of the three considered trios were properly covered by contig-alignments.

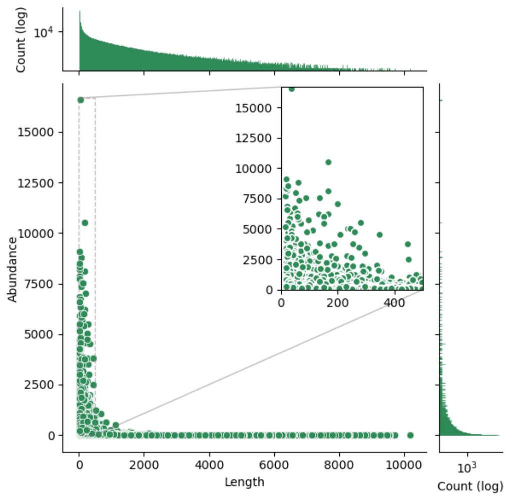
3.5 Discussion

We have presented a novel algorithm called Ping-Pong for finding substring-free samples-specific (SFS) strings with the primary objective of performing comparative genome analysis between two groups of whole-genome sequenced samples. We have shown that these SFS strings capture a comprehensive representation of genomic variation between samples of interest. In practice the proposed approach is capable of finding sequences that span the breakpoints of most variants specific to each sample.

The proposed approach improves upon fixed-length sequences (i.e., *k*-mers) for comparative



(a) Exact Ping-Pong Search



(b) Relaxed Ping-Pong Search

Figure 3.9: Correlation between length and abundance of HG00733-specific strings occurring at least 5 times: 34,219,149 (7,125,436) strings for the Exact (Relaxed) version. For ease of presentation, we zoomed the region concerning strings of length ≤ 500bp: 33,600,277 (6,750,216) strings for the Exact (Relaxed) version.

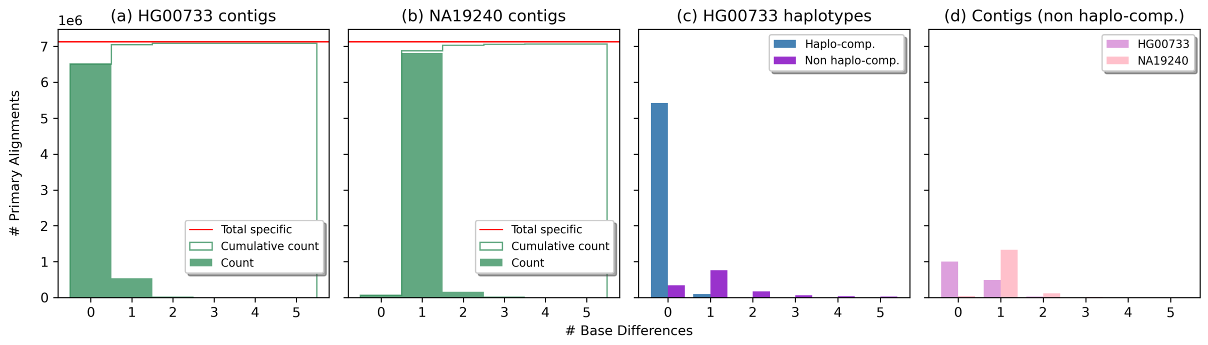
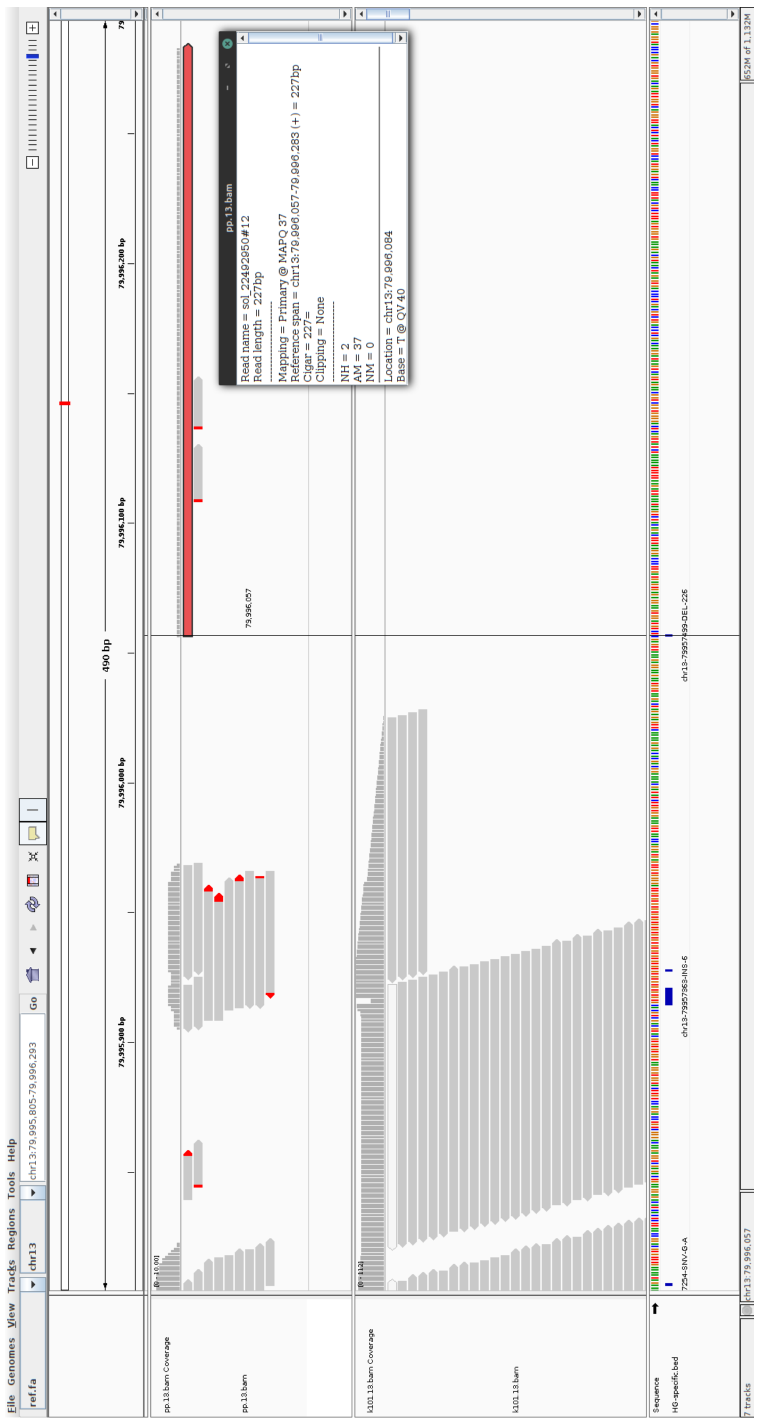


Figure 3.10: Results on relaxed specific strings. Panels (a, b): comparison of the quality of HG00733-specific strings alignments computed against the HG00733 contigs (a) and the NA19240 contigs (b). Panel (c): comparison of the qualities of the HG00733-specific string alignments representing (Haplo-compatible) and not representing (Non haplo-compatible) a specific portion of HG00733 haplotypes. Panel (d): comparison of the qualities of non haplo-compatible HG00733-specific string alignments computed against the HG00733 contigs and the NA19240 contigs. Quality is expressed as number of base differences (mismatches, insertions, deletions, and clips).

Metric		Method	Missed	Total	Hits (%)
Recall	SNPs	Alg. 1 (exact)	39,354	3,147,410	**98.75**
		Alg. 1 (relaxed)	112,940		96.41
		31-mers	243,363		92.27
		101-mers	46,143		98.53
		31-tigs	267,078		91.51
		101-tigs	55,627		98.23
	indels	Alg. 1 (exact)	31,426	716,226	**95.61**
		Alg. 1 (relaxed)	120,313		83.20
		31-mers	131,591		81.63
		101-mers	32,944		95.40
		31-tigs	175,892		75.44
		101-tigs	31,705		95.57
	SVs	Alg. 1 (exact)	1,521	20,775	**92.68**
		Alg. 1 (relaxed)	2,948		85.81
		31-mers	4,912		76.36
		101-mers	1,978		90.48
		31-tigs	6,383		69.27
		101-tigs	2,698		87.01
	All	Alg. 1 (exact)	72,301	3,884,411	**98.14**
		Alg. 1 (relaxed)	236,201		93.92
		31-mers	379,866		90.22
		101-mers	81,065		97.91
		31-tigs	449,353		88.43
		101-tigs	90,030		97.68
H-precision		Alg. 1 (exact)	9,093,407	34,219,149	73.43
		Alg. 1 (relaxed)	1,583,684	7,125,436	**77.77**
		31-mers	28,561,768	97,975,734	70.85
		101-mers	120,109,600	387,221,925	68.98
		31-tigs	2,764,640	5,839,695	52.66
		101-tigs	2,167,395	5,281,605	58.96
C-precision		Alg. 1 (exact)	7,024,433	34,219,149	79.47
		Alg. 1 (relaxed)	669,324	7,125,436	**90.61**
		31-mers	23,170,031	97,975,734	76.35
		101-mers	84,211,940	387,221,925	78.25
		31-tigs	2,563,021	5,839,695	56.11
		101-tigs	-	-	-

Table 3.3: Variant analysis on real human HiFi data. Recall is the fraction of known alleles specific to HG00733 (w.r.t. NA19240) overlapped by at least one HG00733-specific string (or specific *k*-meror specific unitigs). For the sake of completeness, we reported the recall values for alleles coming from SNPs, indels (2-49bp), and SVs ($\geq 50bp$), as well as all the considered specific alleles. H-precision (Haplotype-aware precision) is the fraction of HG00733-specific strings (or HG00733-specific *k*-mersor HG00733-specific unitigs) representing a portion of its haplotypes that is specific w.r.t. the NA19240 haplotypes. C-precision (Contig-based precision) is the fraction of HG00733-specific strings (or *k*-mersor unitigs) aligning perfectly only to HG00733 contigs (and with errors to NA19240 contigs). C-precision for 101-tigs is not present since `minimap2` crashed while mapping long unitigs to the contigs.



pp.13.bam
Read name = sol_22492950#12
Read length = 227bp

Mapping = Primary @ MAPQ 37
Reference span = chr13:79,996,057-79,996,283 (+) = 227bp
Cigar = 227=
Clipping = None

NH = 2
AM = 37
NM = 0

Location = chr13:79,996,084
Base = T @ QV 40

Figure 3.11: Example of allele from a long deletion (bottom track, vertical line) covered only by a HG00733-specific string (top track) and not by specific 101-mers (mid track). The string of length 227bp occurs 12 times in the sample and starts exactly at the breakpoint position (vertical line middle of the image). No 101-mers cover such a breakpoint since at least 227bp are needed to make any string covering the breakpoint specific to the HG00733 sample. On the left, two variations (a small insertion and a snp) are covered both by specific strings and specific 101-mers (note the "overlapping nature" of *k*-mers).

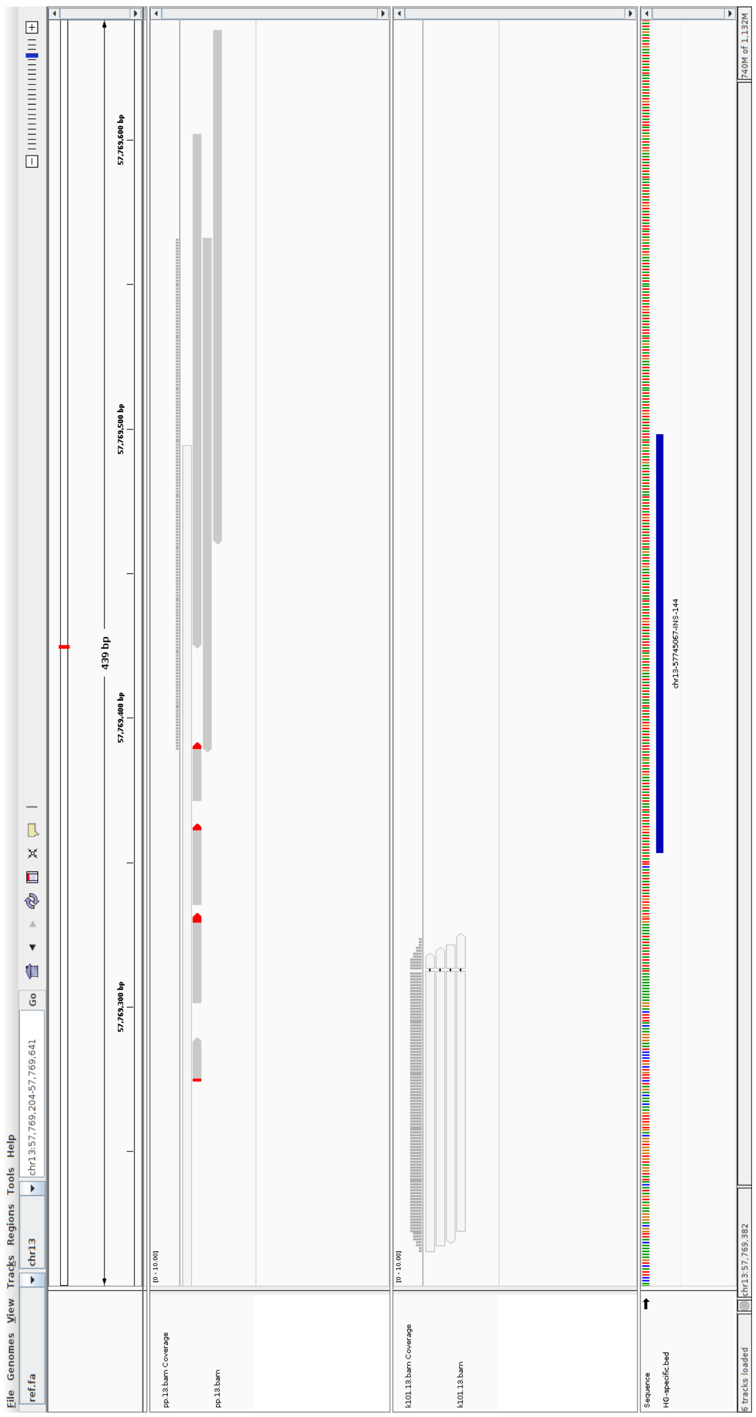


Figure 3.12: Example of allele from a long insertion (bottom track) covered only by HG00733-specific strings (top track) and not by specific 101-mers (middle track). Although the insertion comes from a tandem repeat region of the genome, specific strings - thanks to their variable-length nature - are able to cover it.

genome analysis in three aspects: 1) higher recall: SFS sequences cover a higher fraction of true difference between two genomes than fixed-length k-mers. This is mainly due to the variable-length nature of SFS which increases their power in finding strings representative of differences between genomes in repetitive regions (e.g., segmental duplications). 2) higher precision: our experiments have indicated that SFS sequences have a higher precision than fixed length k-mers($k = 31$ or 101). 3) specificity: our exact algorithm returned between 3x-10x less strings than k-mers, making results more amenable to further analysis. As a motivating example, we could not exhaustively map the results of the 101-mer analysis in reasonable time (< 1 week).

Our method also has several major advantages over traditional mapping based approaches for comparative genome analysis. First, it is not dependent on a prior knowledge of variants in each sample and thus, its performance is not impacted by the biases in variant prediction methods. Second, the proposed approach does not require mappings of the reads, hence, ambiguities in read mappings or biases in mapping algorithms will not impact the results of the proposed method.

One of the main limitations of the proposed method is its reliance on reads with low sequencing error (e.g., HiFi reads). To be able to accurately predict SFS strings from reads with sequencing errors we need to utilize an error correction tool such as `ntEdit`. This method is not expected to translate well to higher error-rate long reads, unless correction yields nearly perfect reads.

Another downside is the 3x longer running time of the relaxed algorithm compared to k-mers. This longer runtime is mainly due to the overhead of building the FMD index. We note that the FMD index can be replaced by a more efficient implementation that offers the same backwards and forward extension operations, if such a data structure or implementation becomes available, thus improving the performance of the method.

We believe there are many applications and possible future research directions for SFS. An obvious application of the experiments presented in this chapter would be the discovery of *de novo* variants in the child sample in genomic trios (Section 3.4.1). Another potential application would be discovery of somatic variants between whole-genome sequences of tumor and normal tissues. Furthermore, as SFS strings will capture any variant as long as it produces a genomic sequence not present in the FMD index, our method could be used as an orthogonal approach to catalogue all variants in a given sample against the reference genome. Note that this mapping-free variant calling approach against a reference genome would be significantly faster than the comparative analysis scenario as only the reference genome needs to be FMD-indexed. We will explore this idea more deeply in the following chapter.

3.6 Code and Data Availability

The current implementation of the Ping-Pong algorithm along with detailed usage manual and instructions to reproduce the results and experiments presented in this chapter is publicly available at https://github.com/Parsoa/PingPong.

The results in this chapter where published under the title "Comparative Genome Analysis using Sample-specific String Detection in Accurate Long Reads" in Bioinformatics Advances in May 2021, available at `https://doi.org/10.1093/bioadv/vbab005`. Ping-Pong was also presented as part of the HIT-Seq track of the ISMB conference in July 2021.

3.7 Contributions

This chapter is a result of collaboration with colleagues Dr. Luca Denti [2], Dr. Rayan Chikhi[3] and Dr. Paola Bonizonni[4].

My contributions were to development of the main ideas behind the Ping-Pong algorithm, implementation of the algorithm in C++, performing and analyzing the results of the simulation experiment and analysis of the results of the real-data experiments. Both first authors equally contributed to the work.

3.8 Funding

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grants agreements numbers [872539] and

[2]luca.denti@pasteur.fr, Department of Computational Biology, Institut Pasteur, Paris, France

[3]rayan.chikhi@pasteur.fr, Department of Computational Biology, Institut Pasteur, Paris, France

[4]paola.bonizzoni@unimib.it, Department of Informatics, Systems and Communication, University of Milano-Bicocca, Milano, Italy

[956229], ANR Inception (ANR-16-CONV-0005), and ANR Prairie (ANR-19-P3IA-0001). This work has also been supported in part by NSF award DBI-2042518 to Fereydoun Hormozdiari.

Chapter 4

Structural Variation Discovery Using Sample-Specific Strings in Accurate Long Reads

4.1 Motivation

There are many methods developed for prediction of SVs using whole-genome sequencing (WGS) data produced from different sequencing technologies. The majority of these methods are mapping-based and try to predict variants by detecting certain SV signatures (i.e., read-depth, read-pair, or split-read) in mappings of the reads to the reference genome. Mapping-free methods are a more recent group of approaches that try to predict SVs without mapping the reads to the reference genome and instead by comparing sequence data between different genomes. Finally, assembly-based approaches first assemble the sequenced reads into longer contigs and use the assembled contigs to predict variants.

There are several limiting factors for predicting SVs using each of these frameworks. Predicting SVs in highly repeated regions of the genome (e.g., segmental duplications) can be particularly challenging for mapping-based methods due to increased error-rate in read alignments. On the other hand, purely mapping-free approaches are mostly limited to detecting the presence of a known variants (e.g provided in a VCF or BED file) as the lack of mapping information means the method is not able to find new variants. Assembly-based approaches are also very computationally intensive and often require integration of data from multiple different technologies (i.e., long-reads, short-reads, and Hi-C). Such approaches are mostly used for building catalogs of variants in large studies rather than for individual genotyping.

Here we propose a novel method called `SVDSS` that combines ideas from all three mapping-based, mapping-free, and assembly-based frameworks for predicting SVs. Our method utilizes mapping-free SFS signatures introduced in Chapter 3 and mapping information to cluster reads potentially including SVs and then performs local assembly and alignment of the clusters for accurate SV prediction. With the combination of different analysis methods, our algorithm is able to improve SV calling performance compared to other contemporary approaches particularly in repetitive areas of the genome.

4.2 Methods

We present `SVDSS` (Structural Variant Discovery with Sample-specific Strings), a novel method for the discovery of SVs from accurate long reads (e.g., PacBio HiFi) using sample-specific strings (SFS). `SVDSS` takes as input a reference genome and a mapped BAM file and produces SV calls in VCF format along with assembly contigs for SV sites in SAM format.

SFS were defined in Chapter 3 as the shortest substrings that are unique to one genome (or equivalently its sequencing reads) with regards to another genome. Here our method utilizes SFS extracted using the Ping-Pong algorithm for coarse-grained identification of potential SV sites and performs partial-order-alignment (POA) [100] of clusters of SFS from such sites to produce assembly contigs that are then locally aligned to the reference genome to detect SVs. The main advantage of using SFS is that they are not limited to a fixed length and the algorithm can dynamically find the shortest string for covering the breakpoints of each variant, thus, making SFS ideal for anchoring potential SV breakpoints.

`SVDSS` has three main steps as depicted in Figure 4.1:

1. **Read smoothing:** reads are smoothed to remove sequencing errors, SNPs and small indels (Figure 4.2).

2. **SFS computation and superstring assembly:** SFS are computed from the smoothed reads (Figure 4.1, 2A) and then assembled into superstrings (2B).

3. **SV prediction from superstrings:** SFS are clustered based on position and length (Figure 4.1, 3A) and are assembled using POA (3B). The assembly contigs are mapped back to the reference and SVs are called from the superstrings (3C).

Before any SFS are extracted, we preprocess the input reads to remove SNPs, short indels ($<$ 20bp) and sequencing errors that may interfere with SV calling (Figure 4.1, step 1). This preprocessing step is called *smoothing* and is described in detail in Section 4.2.2. Smoothing significantly reduces the number of extracted SFS while increasing their specificity for the purpose of SV calling. SFS are then extracted from the smoothed reads using the optimal Ping-Pong algorithm [101] (step 2A). To reduce sequence redundancy, overlapping SFS on each read are further assembled into superstrings (step 2B).

Next, nearby superstrings are clustered and extended to include unique anchoring sequences from the reference genome (step 3A). The position of the superstrings with respect to the reference genome is inferred based on the mapping of the reads they originate from. Each cluster of superstrings is further divided based on length into up to two subclusters and each subcluster is assembled with POA (step 3B) to generate haplotype candidates. The `abPOA` [102] library is used for POA computation. Finally the resulting consensus sequences are aligned back to the reference genome to make SV calls (step 3C). We will now explain the different parts of the method in more detail:

4.2.1 Sample-specific string computation and assembly

`SVDSS` uses the Ping-Pong algorithm from Chapter 3 to compute SFS. Ping-Pong works by building the FMD index [85] of the reference genome and querying the reads of a PacBio HiFi sample against this index and reporting substrings that are not present in the index. The FMD index is a bidirectional text index with constant-time forward and backward search operations and allows for extremely fast computation of SFS.

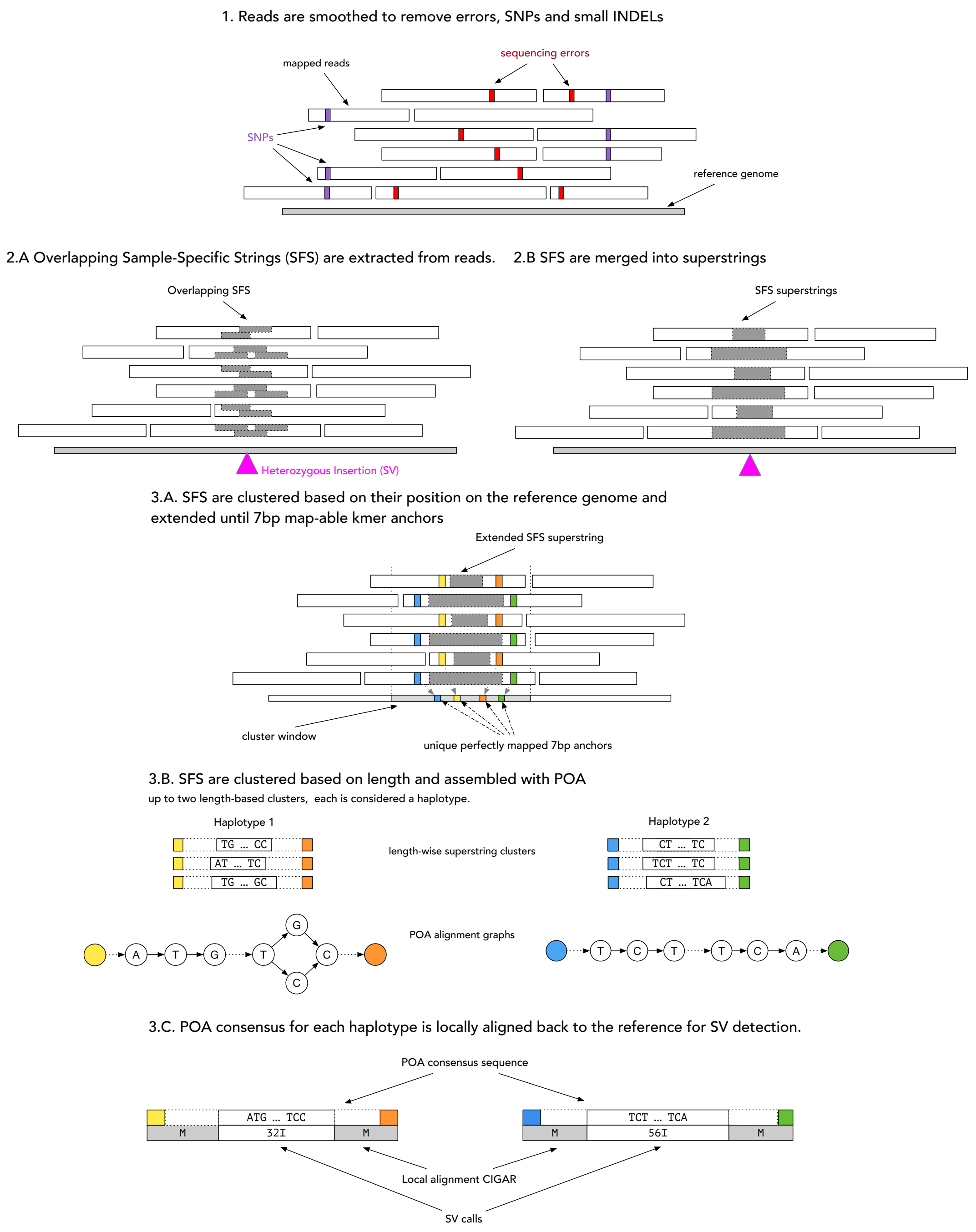

1. Reads are smoothed to remove errors, SNPs and small INDELs
mapped reads
sequencing errors
SNPs
reference genome
2.A Overlapping Sample-Specific Strings (SFS) are extracted from reads.
2.B SFS are merged into superstrings
Overlapping SFS
SFS superstrings
Heterozygous Insertion (SV)
3.A. SFS are clustered based on their position on the reference genome and extended until 7bp map-able kmer anchors
Extended SFS superstring
cluster window
unique perfectly mapped 7bp anchors
3.B. SFS are clustered based on length and assembled with POA
up to two length-based clusters, each is considered a haplotype.
Haplotype 1
TG ... CC
AT ... TC
TG ... GC
length-wise superstring clusters
Haplotype 2
CT ... TC
TCT ... TC
CT ... TCA
POA alignment graphs
A T G T G C C
T C T T C A
3.C. POA consensus for each haplotype is locally aligned back to the reference for SV detection.
POA consensus sequence
ATG ... TCC
M
32I
M
TCT ... TCA
M
56I
M
Local alignment CIGAR
SV calls

Figure 4.1: Overview of the SVDSS SV prediction pipeline. 1) Reads are smoothed to remove SNPs and sequencing errors. 2) SFS are extracted from reads (A) and assembled into superstrings (B). 3. Superstrings (grey) are clustered based on their placements on the reference genome and extended until uniquely mapable 7bp anchors on each side (colored) (A). Each cluster is further clustered into up to two subclusters based on length of the superstring. Each subcluster signifies a potential haplotype. The subclusters are assembled with POA to generate a consensus sequence (B). The POA consensus for each cluster is locally aligned to the reference genome and SVs are called from the mapping information.

SFS capture nearly all variation in the sample genome with regards to the reference genome. Each sequencing read including a variant will produce the same SFS for this variant. Indeed, each sequencing read including a variant produces at least one SFS supporting the variant, hence a variant will be supported by at least one SFS per read covering it. SV breakpoints usually result in novel sequences that are captured as SFSs. However, due to the "shortest" property of SFS, the entire SV sequence is not necessarily covered by a single SFS: a read may produce several overlapping SFS for long variations.

To remove unnecessary redundancy in the information captured by overlapping SFS, we assemble all such overlapping SFS into longer strings called "superstrings". We note that assembling SFS into superstrings also reduces the number of SFS by an order of magnitude, making any downstream analysis more efficient. As SFS on each read are naturally sorted based on start position, the assembly stage can be implemented as a single pass over the SFS on each read, merging each SFS with the next one if they overlap. The resulting superstring can further be merged with the next SFS if they overlap. More formally, on a read R where k consecutive SFS are overlapping such that $R[i_1, j_1]$ overlaps with $R[i_2, j_2]$ and $R[i_2, j_2]$ overlaps with $R[i_3, j_3]$ and $\ldots R[i_{k-1}, j_{k-1}]$ overlaps with $R[i_k, j_k]$, we merge the strings into the single superstring $R[i_1, j_k]$.

Assembling SFS into superstrings reduces the number of SFS by an order of magnitude and removes unnecessary redundancy in the information captured by overlapping SFS. The assembly procedure effectively merges all the SFS belonging to the same variant into a single long superstring. This results in superstrings from the same variant to have similar length, sequence and position with respect to the reference genome which allows them to be easily clustered for SV prediction.

4.2.2 Read smoothing

The SFS extraction step (Ping-Pong algorithm) requires reads with low error-rates for optimal performance as sequencing errors can result in millions of erroneous SFS to be extracted. While most such SFS can be filtered later on, they can still affect the accuracy negatively and will increase runtime by adding excess computational load. For the experiments in Chapter 3 we had used `ntEdit` [92] to further error-correct the HiFi reads. However, `ntEdit` is computationally expensive and increases the runtime of the method significantly. Furthermore, the presence of millions of SNPs and small INDELs in a sample results in tens of millions of additional SFS being extracted that are not directly useful for genotyping SVs. To solve both of the above problems, we introduce a preprocessing step called *“read smoothing”* that aims to eliminate both sequencing errors and short variants from the dataset. The smoothing algorithm uses information from the CIGAR strings of BAM alignments to remove any short mismatch between a read and the reference genome.

For segments reported as a match between a read and the reference genome (CIGAR op ‘`M`’), the algorithm will replace the read sequence with the corresponding sequence from the reference genome, automatically removing any single-base mismatches (i.e., sequencing errors or potential SNPs) in the process. For short (< 20bp) deletions (CIGAR op ‘`D`’), the algorithm will remove the deletion from the read by copying back the deleted bases from the reference sequence. Short (< 20bp) insertions (CIGAR op ‘`I`’) are similarly smoothed by removing the inserted bases from the read.

Soft-clipped regions (CIGAR op ‘`S`’) will be retained as they indicate potentially long insertions or deletions; any SNP or sequencing error inside clipped regions cannot be corrected as a result. Note that removing an INS or DEL segment results in the merging of the surrounding ‘`M`’ section in the CIGAR. A smoothed read’s CIGAR strings will have significantly fewer edit operations than that the original read and it will consist of one or more very long ‘`M`’ segments with large INDELs in between and potentially surrounded with soft-clipped regions. Figure 4.2 illustrates the smoothing procedure on an example read.

The above modifications will not change the overall mapping of the read as the mapping end and begin positions remain the same as before. As a result, the algorithm will not change the order of the reads in a sorted BAM file. This allows us to quickly smooth a sorted BAM file without the need to sort it again. However, because the size of the reads may have changed, the index of the original BAM files is no longer valid for the smoothed BAM and it has to be indexed again with `samtools index`.

Note that the Ping-Pong algorithm won’t produce any SFS that is entirely contained in a M section of a smoothed read as the corresponding sequence has been replaced base-by-base with

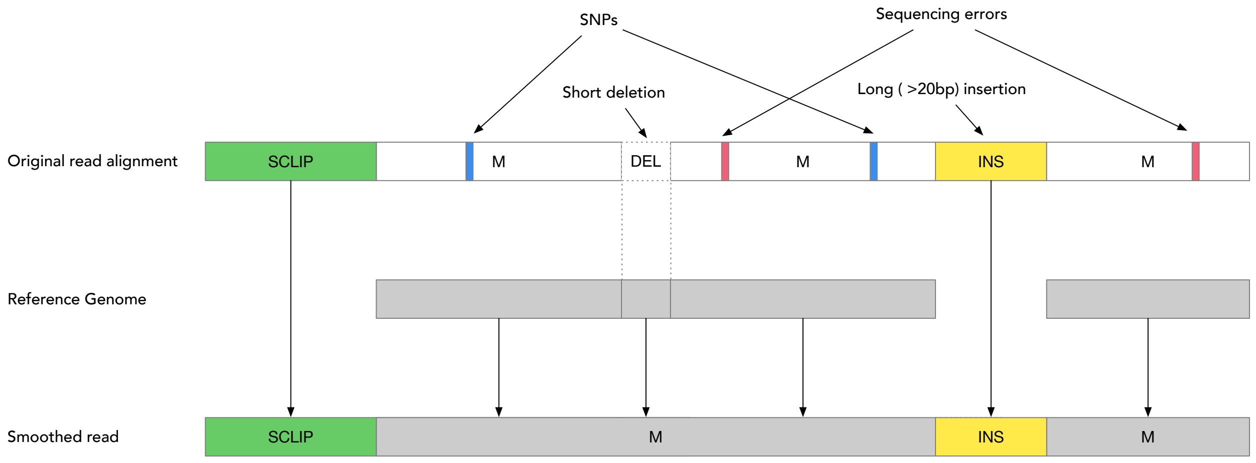


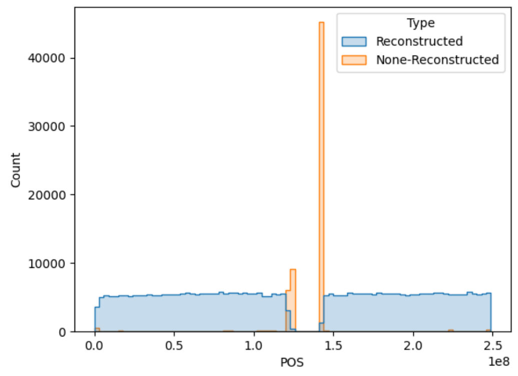
Figure 4.2: Illustration of the read smoothing algorithm. `M` alignment segments are smoothed from the reference genome, correcting SNPs (blue) and sequencing errors (red) in the process. Long indels are preserved while small ones ($<$ 20bp) are removed. The small deletion is smoothed using the reference genome sequence while the large insertion (yellow) - potentially a SV - is carried over to the read. The soft-clipped section (green) is directly copied to the read.

reference genome sequence. This significantly reduces the number of SFS that will be extracted from smoothed reads and allows for a small runtime reduction when processing smoothed reads.

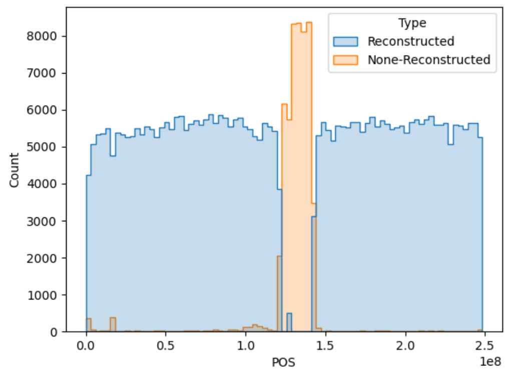
Smoothing relies on correctness of long HiFi read alignments. If an alignment is thought to be inaccurate, the algorithm won't modify its sequence. The algorithm keeps track of the average number of mismatches between M section of alignments and the corresponding reference sequence as it's processing the reads. Any read that has more than 3 times the mismatch rate will be ignored.

In our experiments, smoothing effectively reduces the number of extracted SFS by over 90%, while having effectively no impact on the SV calling pipeline's recall. Out of the 6.2M reads for the CHM13 samples, around 5M are smoothed and the rest are deemed to have unreliable mappings and are discarded. The 1.2M non-smoothed reads from CHM13 are responsible for more than 82% of all SFS extracted from that sample after smoothing. However, the SFS extracted from non-smoothed reads do not contribute to the method's recall at all. Indeed excluding the SFS extracted from non-smoothed reads increases the method's precision while leaving the recall unaffected. This leads to the heuristic of excluding non-smoothed reads from the `SVDSS` pipeline entirely.

Further analysis shows that effectively all non-smoothed reads map to centromere regions of



(a) GRCh38



(b) CHM13

Figure 4.3: Distribution of read mapping locations on chr1. Almost all non-smoothed reads originate from centromeres of CHM13, however almost none can be properly mapped the GRCh38, resulting in an alignment gap around the centromere.

the CHM13. Figure 4.3 below shows the distribution of mapping positions of reads from chr1 on both CHM13 and GRCh38. The large gap around the centromere when mapping to GRCh38 explains the poor performance of non-smoothed reads when predicting SVs against the reference genome.

In summary, read smoothing is a critical preprocessing step of the SVDSS pipeline. smoothing reduces the number of retrieved SFS and increases the specificity of the extracted SFS which results in higher precision in predicting SVs without affecting the recall. The procedure is also computationally very lightweight, as it essentially rewrites the BAM file in a single pass with minor modifications. As a result, smoothing is an effective method for increasing the specificity of SFS for SV calling and improving the computational efficiency of the pipeline.

4.2.3 SV Calling

The main SV-calling algorithm consists of three main steps as shown in Figure 4.1. We will explain each step in more detail below:

1. Superstrings constructed from the SFS strings are "placed" on the reference genome by extracting their alignments from read alignments. The superstring are then clustered based on the loci they are aligned to Each cluster represents one or more SVs that are close to each

other and may be potentially from different haplotypes.

2. Each cluster is further clustered based on length to generate up to two haplotype candidates (taking into account the diploidy of the human genome). Each haplotype cluster candidate is assembled with Partial Order Alignment (POA) to yield a consensus sequence.

3. Each haplotype candidate is locally realigned back to the reference genome region corresponding to its cluster and SVs are called based on the alignment.

4.2.3.1 Superstring placement and clustering

Aligning superstrings back to the reference genome can be time-consuming and error-prone due to their relatively short lengths, however we note that the necessary mapping information is indeed already available in the sample's read mappings. In practice, superstring are not aligned to the reference genome but instead their alignment is extracted from the alignment of the reads they originate from. Assuming $R[i, j]$ is a superstring that spans positions $i \dots j$ on read R, its alignment to the reference will be $C[i, j]$ with C being the CIGAR string of R.

It is possible that a superstring's mapping is entirely contained in an inserted or clipped part of the read and hence the mapping extracted above cannot be used to place the superstring on the reference genome. To avoid this, each superstring is extended on the read from both sides until we reach a perfectly map-able and locally unique k-mer anchor (the default value for k is 7). The superstrings that cannot be extended in this manner are ignored. Figure 4.1 3A shows this extension procedure. The k-mer anchoring idea was influenced by LongShot [103].

We observe that due to this extension and the fact that a deletion or insertion cannot directly precede or follow a soft-clip in the read alignment, superstrings cannot have both a deletion/insertion and a soft-clip. This allows us to classify the superstrings based on their alignments into 4 groups: deletions, insertions, both and soft-clips. We ignore superstrings that don't include any deletions, insertions or soft-clips.

Finally, we cluster the superstrings of each type based on their their mapping location: superstrings that have close enough mappings (by default less than 500bp apart) are placed in the same cluster. The resulting cluster's interval will be the smallest interval in the genome that completely includes all of its superstrings. Each cluster now represents either a single SV or several close or overlapping SVs possibly from different haplotypes. Based on the type of superstrings in each cluster, the type of SV being predicted by the cluster is tentatively known. However, for clusters including soft-clipped superstrings, the type of SV is not immediately known and we need further processing to determine the type.

4.2.3.2 SV calling from non-clipped clusters

Each cluster so far includes one or more close SVs. However, as the human genome is diploid, the SVs might indeed be from different haplotypes. To resolve the different haplotypes, we further split each cluster into subclusters of superstrings of similar size and sequence. This is based on the assumption that different alleles at each site have different length and sequence. The similarity of sequences is calculated using `rapidfuzz` [1]. The two largest resulting subclusters (in terms of number of superstrings) are selected as haplotype candidates since the human genome is diploid. If only one subcluster is returned, it signifies a homozygous variant. `SVDSS` then computes a consensus sequence for each subcluster using Partial Order Alignment.

Assume that a cluster c spans the interval $G[s_c, e_c]$ of the reference genome G. Most strings of the cluster only partially cover this interval (i.e., they align to positions $[s, e]$ with $s_c \leq s < e \leq e_c$) while some others span the entire interval (i.e., they align to positions $[s_c, e_c]$). In order to perform a more accurate POA, `SVDSS` requires all the strings in a cluster to be of the same length. Therefore, `SVDSS` fills the gaps preceding or following a superstring using the reference genome. For instance, if a superstring S aligns to $[s, e]$ with $s_c < s < e < e_c$, then the resulting sequence will be $G[s_G, s - 1] + S + G[e + 1, e_G]$ (where + is the string concatenation operator). The main goal of this extension is to summarize the information contained in a cluster and to minimize the difference between the superstrings coming from different reads. The extended superstrings in each subcluster are then aligned to each other using POA to generate a consensus (Figure 4.1 3B).

Finally, each POA consensus sequence is realigned locally to the reference genome window corresponding to its cluster using `parasail` [104] and the alignment's CIGAR information is analyzed to call and detect SVs (Figure 4.1 3C). A weight is assigned to each SV prediction based

[1]`https://github.com/maxbachmann/rapidfuzz-cpp`

on the number of superstrings that support it. A higher support indicates a more confident call. By default, we filter SV calls having less than four supporting superstrings however The confidence threshold can also be determined at runtime using the `--min-cluster-wight` option.

4.2.3.3 SV calling from soft-clipped clusters

Clipped reads usually include SVs that are larger than the read length and hence cannot be captured entirely in a single read. For very large insertions, reads completely contained within the inserted sequences are either not mapped at all or will have low-quality mappings and are likely discarded during smoothing.

For clusters including soft-clipped superstrings, the type and coordinates of the SV are decided by analyzing the mapping positions of the superstrings. Each clipped cluster is further divided into "left-clipped" and "right-clipped" subclusters based on which end of each superstring is clipped. The position of each subcluster is calculated by averaging over start position of all its superstrings. For a left(right)-clipped subcluster lc (rc), $lc.p$ ($rc.p$) represents the start position.

We then analyze the distance between the subclusters to detect SV types based on Figure 4.4. In case of a deletion, it is the case that $lc.p > rc.p$. Ideally all left-clipped superstring should have the same start position $lc.p$ and all right-clipped ones should have start position $rc.p$, however in practice there is some variation in positions due to mapping imperfections, in particular in repetitive regions. For an insertion it should ideally be the case that $rc.p = lc.p$, but in practice there is again some variation so we allow that $rc.p - lc.p <= 1000$.

In our experiments with the CHM13 sample, out of the 300 SV calls from clipped regions, only 40% were correct. Furthermore, nearly all SVs correctly reported from clipped SFS were already being detected with non-clipped superstrings, meaning that they were short enough compared to the read length. Due to this, `SVDSS` does not by default use clipped SFS for SV detection. However, this can be optionally enabled by passing the `--clipped` flag. We recommend manual inspection of clipped SV calls.

4.2.3.4 SV Chain Filtering

In repetitive regions of the genome such as STRs, the structure of the DNA may result in reads originating from the same locus mapping to slightly different coordinates. This will result in mul-

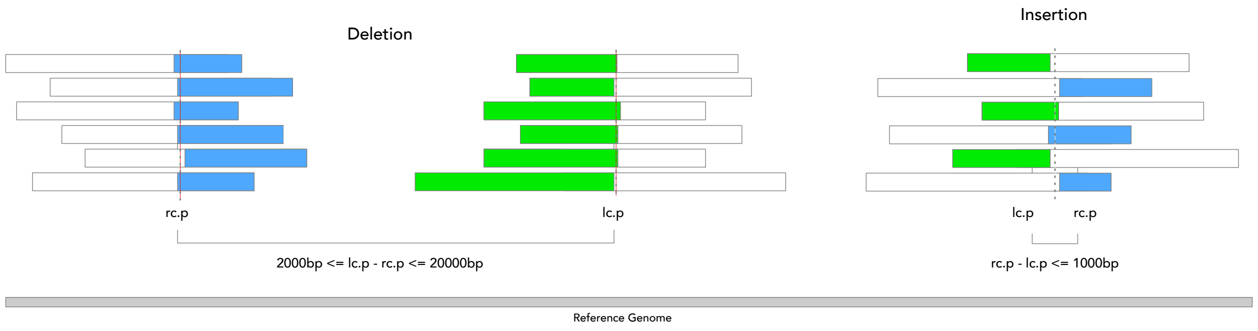


Figure 4.4: SV calling from clipped superstrings. A deletion SV is seen on the left and an insertion on the right. Green shows left clips and blue shows right clips. $lc.p$ and $rc.p$ shows the start position of the left-clipped and right-clipped subclusters. Note that the $lc.p < rc.p$ for insertion and $lc.p > rc.p$ for deletions.

tiple SV calls for the same variant but at slightly different positions. To reduce the number of false positives and eliminate such redundant calls, we perform a "chain-filtering" post-processing step. This step sorts all predicted SVs based on coordinates and filters out consecutive SVs of the same type with similar sizes, keeping only the one with the highest weight. This heuristic is indeed very effective, and on CHM13 our precision is improved by nearly 2% after chain-filtering while the recall is not affected.

4.3 Results

One complexity in comparing different tools for calling SVs is the imperfectness of available callsets. Missing variants and potentially false predictions affect almost all published callsets, and even the most high-quality callsets have been reported to have a $\sim$ 5% false discovery rate [15]. Furthermore, many callsets are constructed using state-of-the-art but imperfect tools and are thus biased towards these methods [105].

For these reasons, we have opted out of using popular callsets (e.g. GIAB) in our experimental benchmarking and instead constructed our ground truths using the recent high-quality haplotype-resolved *de novo* assemblies built for a few samples. We considered the assemblies available for three samples CHM13, HG002, and HG007 and built the callsets by comparing each assembly against the GRCh38 reference genome using the assembly-to-assembly SV calling tool `dipcall` [105]. By calling SVs directly from the high-quality assemblies, we can exhaustively benchmark our SV calling approach against state-of-the-art tools using unbiased and comprehensive SVs callsets.

4.3.1 Whole-genome SV Discovery Performance

We experimentally validated the accuracy of the `SVDSS` pipeline in calling SVs from three whole-genome samples sequenced using PacBio HiFi technology: the homozygous CHM13 sample from the telomore-to-telomere (T2T) project [56] and the HG002 and HG007 samples from the GIAB project [106] corrected using DeepConsensus [107]. These samples were chosen because of the availability of high-quality and effectively complete assemblies for them. Furthermore, the HG002 and HG007 samples show higher accuracy than standard HiFi samples corrected using only `pbccs`. The use of both homozygous (CHM13) and heterozygous (HG002 and HG007) samples allows for more comprehensive analysis of SV calling methods.

We mapped each sample against the reference genome using `pbmm2` [108] and then we called SVs on each sample using the `SVDSS` pipeline. We compared our approach to 4 state-of-the-art mapping-based SV callers: `pbsv` [44], `cuteSV` [45], `sniffles` [109], `SVIM` [110] and on a recent preprint of a POA-based method, `debreak` [111]. We ran each caller on the three considered samples mapped with `pbmm2` to call insertions and deletions.

We validated the calls of each tool against the truthset constructed with `dipcall` using the benchmarking utility `Truvari` [112]. `Truvari` reports precision, recall, and F1 score for each method. From this comparison, we further exclude calls made in haplotype gaps in the respected assemblies reported by `dipcall` (i.e., regions of the reference genome not covered by both haplotypes), as any such call would be classified as false positive regardless of correctness. Our method consistently outperforms other methods in all three measures of accuracy when considering the full genome (Table 4.1 - *Full Genome* row).

On HG002 and HG007 samples, `SVDSS` outperforms the other callers' recall by 5 to 10% while achieving the highest precision on the full genome. `SVDSS` has been able to report $\sim$ 2,400 more correct calls on HG002 and $\sim$ 1,600 more calls on HG007 without introducing many false calls. `SVDSS` also achieves the highest recall on CHM13 and reports $\sim$ 400 more true positive calls than other methods while maintaining a very high precision.

We note that the improvement achieved by SVDSS over other approaches is less significant for CHM13 compared to the other two samples (improvement of 2 to 5% in recall while achieving similar precision to other tools). We believe this is partially due to homozygous nature of the CHM13 sample that makes SV calling relatively easier for all approaches.

Figure 4.6(a) reports the length distribution of the SVs called by each tool on the HG007 sample. On HG007, the number of SVs reported by each tool ranges from 34,827 to 38,659 with SVIM reporting the lowest number of SVs and SVDSS reporting the highest number. Overall, all the tools report more insertions than deletions with shorter SVs (of length $\leq 100bp$) being more frequent than longer ones. Moreover, all the tools show a clear peak at around 300bp suggesting a good capability in calling potential *Alu* mobile elements.

We also repeated the above experiment on HG007 using different aligners to test how SV callers are influenced by how reads are aligned. We tested all 5 callers in combination with minimap2 [113] and ngmlr [114]. Overall, the choice of aligner seems to have minimal impact on the SV calling performance of all methods. Table 4.2 reports results for this analysis.

Finally, we also investigated how read coverage affects SV calling performance. To this aim, we subsampled the HG007 sample (coverage 15x) down to 5x and 10x and we ran the 5 considered approaches on these two newly-created samples. Our SVDSS approach was also able to outperform other approaches using 10x sequencing coverage in all the metrics of interest (precision, recall, and F1, see (Figure 4.6(b) and Table 4.3).

When sample coverage is low (5x), pbsv achieves the highest recall (63.2%) at the expense of lower precision (58.6%) whereas other tools achieve similar high precision (ranging from 87.4% of SVIM to 92.9% of SVDSS) but low recall (ranging from 46.2% achieved by SVDSS to 51.6% achieved by cuteSV). On the other hand, with higher coverages of 10x and 15x, SVDSS achieves the best precision and recall, outperforming other approaches.

4.3.2 SV calling performance in hard-to-analyze regions.

For further analysis, we divided the genome into two regions based on difficulty of calling variants based on GIAB's definition of tiers [106]. Tier 1 accounts for nearly 86% of the genome spanning 2.51 Gbp, includes 50% or less of the total expected number of SVs, and is likely biased towards

		HG002			HG007			CHM13		
Region	Tool	P	R	F1	P	R	F1	P	R	F1
Full Genome	SVDSS	88.4	**78.2**	**83.0**	**90.1**	**76.5**	**82.7**	87.3	**84.6**	**86.0**
	cuteSV	86.0	68.6	76.3	88.3	68.1	76.9	87.1	79.7	83.2
	pbsv	86.9	68.8	76.8	84.9	68.6	75.9	84.6	82.7	83.6
	sniffles	82.0	67.3	73.9	86.7	64.1	73.7	86.4	81.4	83.8
	SVIM	83.5	65.1	73.2	84.9	64.7	73.4	**90.1**	79.9	84.7
	debreak	**88.6**	67.5	76.6	90.1	64.2	75.0	83.7	79.6	81.6
Tier 1	SVDSS	95.2	**85.5**	**90.1**	95.2	**82.7**	**88.5**	95.3	93.4	**94.5**
	cuteSV	90.9	82.9	86.7	93.0	79.9	86.0	94.8	93.1	93.9
	pbsv	95.7	83.1	89.0	89.7	80.5	84.9	94.0	93.7	93.9
	sniffles	87.7	81.1	84.3	92.3	75.9	83.3	87.2	93.6	90.3
	SVIM	90.1	81.1	85.4	91.5	77.9	84.2	**96.6**	92.5	**94.5**
	debreak	96.8	82.5	89.1	**96.2**	76.4	85.2	93.7	93.0	93.3
Extended Tier 2	SVDSS	**82.7**	**72.3**	**77.2**	**84.6**	**70.2**	**76.7**	80.3	**77.4**	**78.8**
	cuteSV	80.9	57.0	66.9	82.3	56.0	66.6	79.9	68.1	73.6
	pbsv	78.4	57.2	66.1	78.8	56.4	65.7	76.0	73.3	74.6
	sniffles	77.8	56.1	65.2	80.3	52.1	63.2	72.7	73.2	72.9
	SVIM	76.4	52.0	61.9	76.2	51.2	61.2	**83.4**	69.3	75.7
	debreak	80.4	55.3	65.5	82.3	51.9	63.7	74.4	68.1	71.1

Table 4.1: **Comparison of performance of SVDSS and other methods on calling SVs.** Accuracy of each tool is reported in terms of Precision (P), Recall (R), and F-measure (F1). Results are further broken down by different regions of the genome.

	pbmm2			minimap2			ngmlr		
Tool	P	R	F1	P	R	F1	P	R	F1
SVDSS	**90.1**	**76.5**	**82.7**	**91.9**	**79.3**	**85.1**	**93.0**	62.8	75.0
cuteSV	88.3	68.1	76.9	89.8	68.8	77.9	90.5	64.3	75.2
debreak	90.1	64.2	75.0	91.3	64.8	75.8	86.0	60.8	71.2
pbsv	84.9	68.6	75.9	85.0	68.7	76.0	84.9	**67.9**	**75.5**
sniffles	86.7	64.1	73.7	90.8	66.1	76.5	87.7	61.3	72.2
SVIM	84.9	64.7	73.4	87.2	65.7	74.9	81.9	64.4	72.1

Table 4.2: Comparison of performance of SVDSS and other methods when calling SVs on HG007 reads mapped with different aligners. Accuracy of each tool is reported in terms of Precision (P), Recall (R), and F-measure (F1). Results are whole-genome.

	5x			10x			15x		
Tool	P	R	F1	P	R	F1	P	R	F1
SVDSS	**92.9**	46.2	61.7	**91.6**	**70.3**	**79.5**	**90.1**	**76.5**	**82.7**
cuteSV	92.0	51.6	**66.1**	90.5	65.2	75.8	88.3	68.1	76.9
pbsv	58.6	**63.2**	60.8	66.0	68.1	67.0	84.9	68.6	75.9
sniffles	91.4	46.9	62.0	89.7	60.0	71.9	86.7	64.1	73.7
SVIM	87.4	49.9	63.5	86.3	62.3	72.4	84.9	64.7	73.4

Table 4.3: Comparison of performance of SVDSS and other methods on HG007 at different coverages. Accuracy of each tool is reported in terms of Precision (P), Recall (R), and F-measure (F1). Results are genome-wide.

easy-to-call SVs [2].

On the other hand, Tier 2 accounts for nearly 0.8% of the genome and consists of ~ 6000 difficult-to-genotype sites. The remaining 13% of the genome mostly consists of centromeres, telomeres, and microsatellite regions (e.g., STRs) which are generally more difficult to genotype because of their repeat structure and due to the ambiguities of the reference genome. Because the high-quality assemblies that are the basis of our analysis include effectively complete genomes for each individual, we decided to extend Tier 2 to also include these regions (Extended Tier 2). This way, we are able to more thoroughly evaluate callers' accuracy across the entire human genome and we do not limit our analysis to easier-to-call regions (i.e., Tier 1). Figure 4.5 provides a breakdown of the tiers we considered in our analysis.

In this analysis, we considered the callsets produced by `SVDSS`, `cuteSV`, `pbsv`, `sniffles`, and `SVIM` starting from `pbmm2` alignments. Table 4.1 reports the results of this analysis. Results on both tiers follow the same trend seen on full genome, with `SVDSS` managing to call more correct SVs without introducing many false calls. As expected, all tools achieve higher accuracy on Tier 1 regions, that are easier to analyze. Furthermore, we observed that the improvement between performance of `SVDSS` and other tools widens in the Extended Tier 2 regions of the genome (Table 4.1). Remarkably, on difficult-to-analyze regions (i.e., extended Tier 2), `SVDSS` achieves the highest recall, outperforming other callers by 15%, /14% and /4% on the three considered individuals.

To further provide evidence of correctness for true positive calls in these hard regions, we analyzed how these calls are shared among the tested callers using an upset plot [115]. Upset plots are an alternative to Venn diagrams that represents more conveniently the intersections of multiple sets. Figure 4.6(c) shows that out of the 10,333 total SVs in the truth set for HG007 (i.e., the `dipcall` callset), 3,923 (38%) of these SVs are correctly called by all the tested approaches whereas 2,444 (24%) are not detected by any tool. Remarkably, 739 SVs (7%) are detected only by our pipeline, partially explaining the higher recall it is able to achieve. `SVIM` has the second-

[2]`https://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/AshkenazimTrio/analysis/NIST_SVs_Integration_v0.6/README_SV_v0.6.txt`

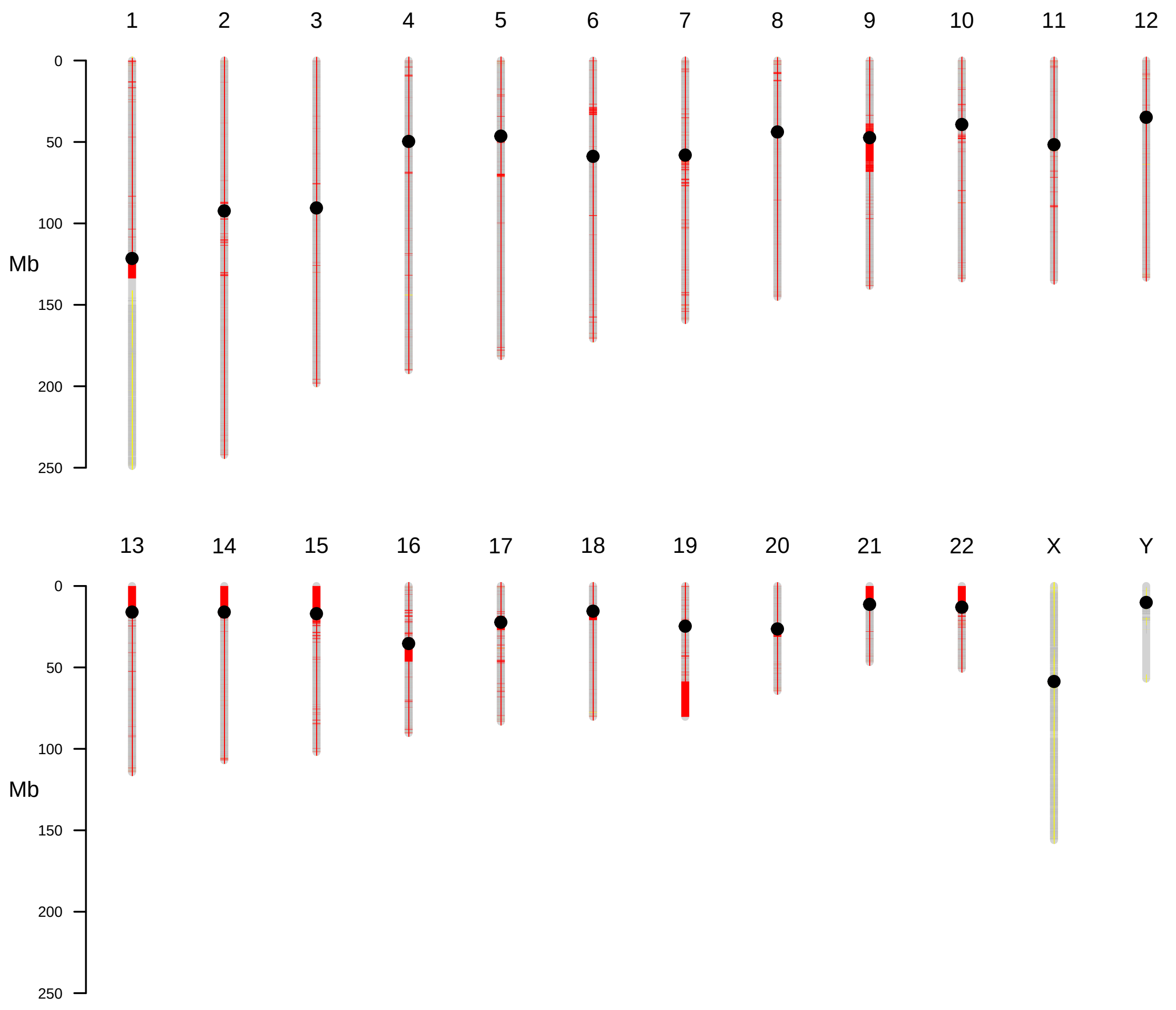


Figure 4.5: Breakdown of reference genome GRCh38 into tiers. Grey areas show Tier 1 regions and red areas correspond to extended Tier 2 regions.

highest number of specific calls at 130. Figure 4.12 shows the distribution of `SVDSS`-specific vs `SVIM`-specific calls on chr1, chr2 and chr3 of the HG007 sample. `SVDSS` also detects the highest number of SVs that would have been exclusive to other tools, i.e., 242 (2.3%) calls are shared by `SVDSS` and `pbsv`, and 215 (2%) are shared between `SVDSS` and `sniffles`.

We manually investigated some of the SVs that are exclusively called by `SVDSS` and we discovered that a some of such calls are SVs that exhibit two different alleles on the two haplotypes. These SVs account for heterozygous non-reference SVs, i.e., SVs genotyped 1/2 (see two examples in Figures 4.7 and 4.8) as well as pairs of close SVs whose alleles come from different haplotypes (see an example in Figure 4.9). Other callers cannot always discern between the two haplotypes and heuristically call only one of the two alleles, inferring the length of alleles by combining the information coming from the two haplotypes.

To further validate our claim, we considered each SV called only by our pipeline and we computed its distance to the closest SV. Out of a total of 792 SVs exclusively called by `SVDSS`, 345 (44%) are located at the exactly the same position as another called SV, hence are heterozygous non-reference SVs, while 227 (29%) SVs are close (≤ 100) to another SV, and 107 (14%) SVs are too distant to another SV to be considered heterozygous events (Figure 4.11). The `SVDSS` pipeline is able to more correctly manage these situations since it better discerns haplotypes supported by the input reads. Indeed, after clustering SV signatures by position, `SVDSS` splits each clusters in two subclusters, one per haplotype, hence allowing it to call two different alleles - if needed.

4.3.3 Baseline Error Rate Comparison

We further investigate the lower bound on baseline false discovery rate of the proposed method by comparing the HiFi reads from CHM13 against the high-quality genome assembly built on the same samples (T2T [56]). Given the almost perfect T2T CHM13 assembly produced using multiple orthogonal technologies, it is expected that an ideal SV caller would predict no SVs when comparing CHM13 reads against this assembly. Thus, we propose to use a simple experiment to establish the lower bound on the baseline false discovery rate of different methods.

Ideally, our pipeline should generate zero SVs calls as no SFS should be extracted when querying CHM13 reads against the T2T assembly of the same sample. This will not be the case in

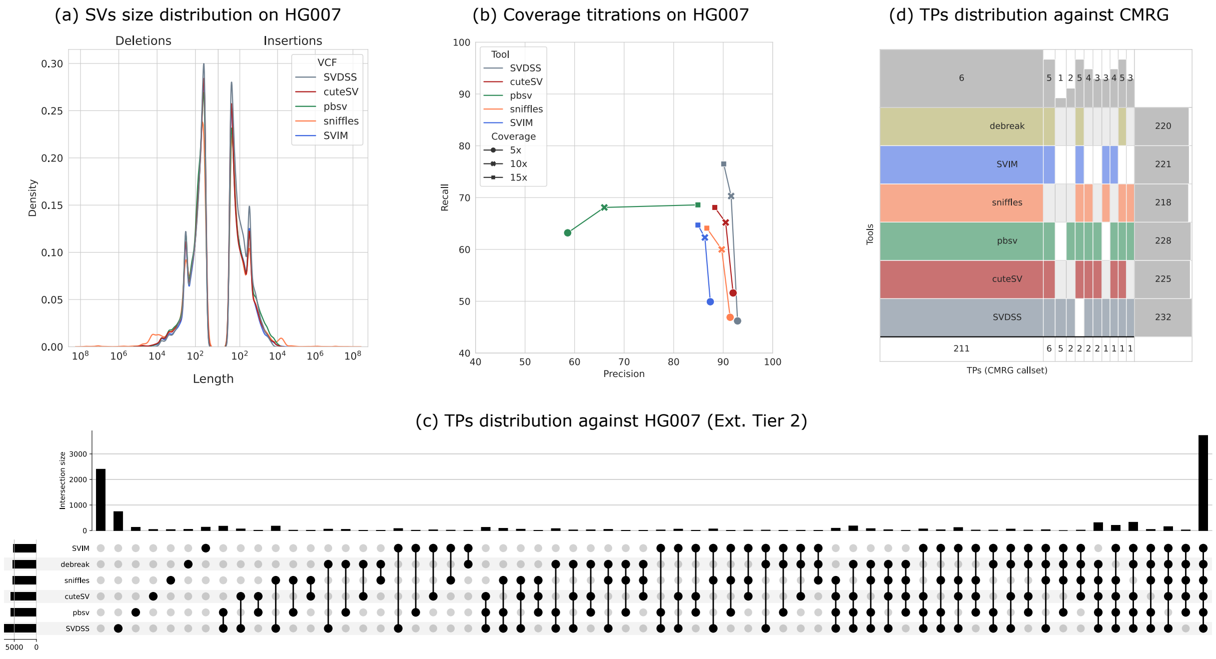


Figure 4.6: (a) Distribution of SVs lengths reported by different tools on HG007 (Full Genome). (b) Lineplot presenting results of the coverage for 5x, 10x, and 15x. (b) Analysis of shared calls (True Positives) between different tools on HG007 (extended tier 2). (d) Venn diagram showing shared calls (True Positives) between different tools on the 273 medically-relevant genes considered in the CMRG callset.

practice due to the abundance of sequencing errors in particular from repetitive regions. Still, we expect the method to produce very few variant calls in this ideal scenario. The number of variants reported in such a scenario would also establish an empirical baseline for the method's error-rate. As a side-objective, we will also investigate the resulting SV calls to find if our method has discovered any true SVs missing from the T2T assembly. Due to the effectively homozygous nature of the CHM13 genome, any true variant discovered must be homozygous. However it is possible artifacts accumulated in the cell-line and actual heterozygosities in the genome may result in heterozygous SVs being reported.

We built the FMD index for v1.1 of the CHM13 assembly and extracted SFS from smoothed CHM13 PacBio HiFi data against this index. We then passed the SFS through the `SVDSS` pipeline for SV discovery. Our pipeline discovers a total of 102 SVs. For comparison, we repeated the above experiment with the other tools `pbsv`, `cuteSV`, `SVIM` and `sniffles`. Table 4.4 below in-

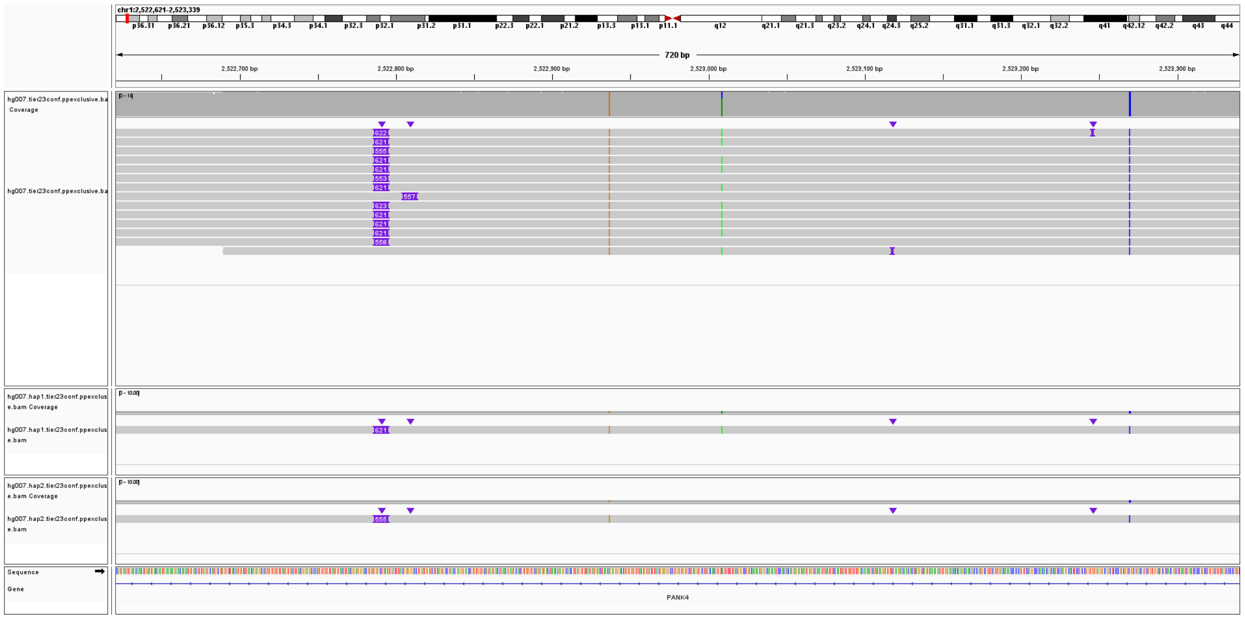


Figure 4.7: Heterozygous insertion in the HG007 sample (`chr1:2522791`). `dipcall` called two alleles of length 555 and 621. `SVDSS` agreed with `dipcall` correctly calling both alleles. `cuteSV`, `pbsv`, `sniffles`, and `SVIM`, instead, called just one allele of length 601, 621, 621, and 602, respectively.

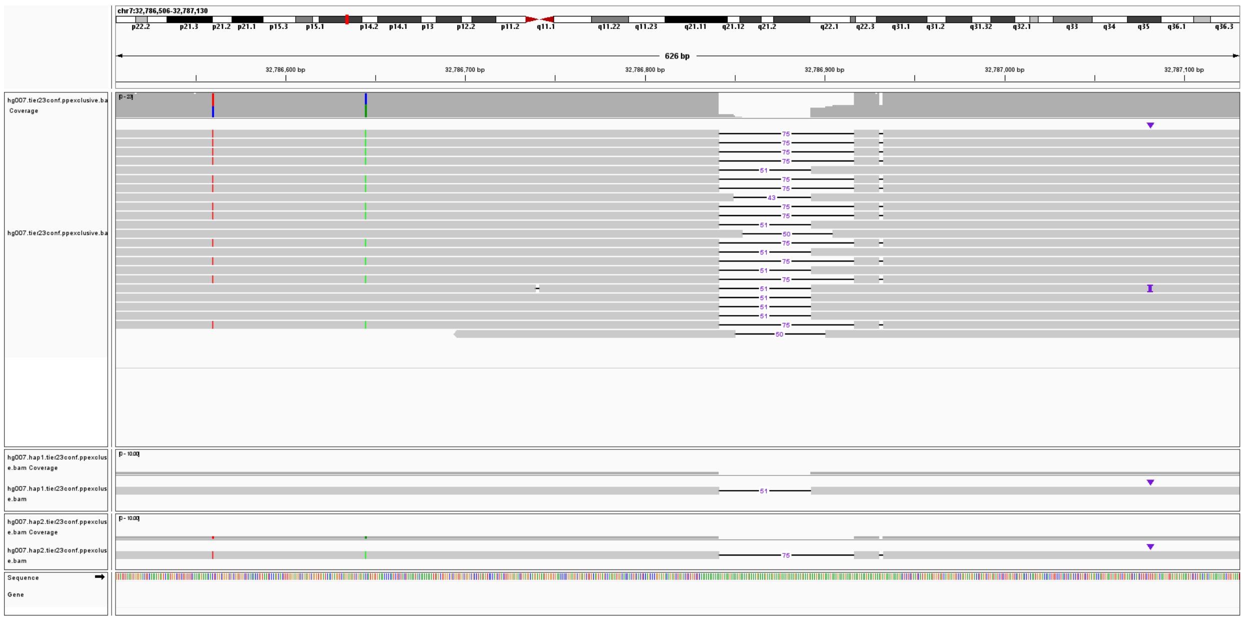
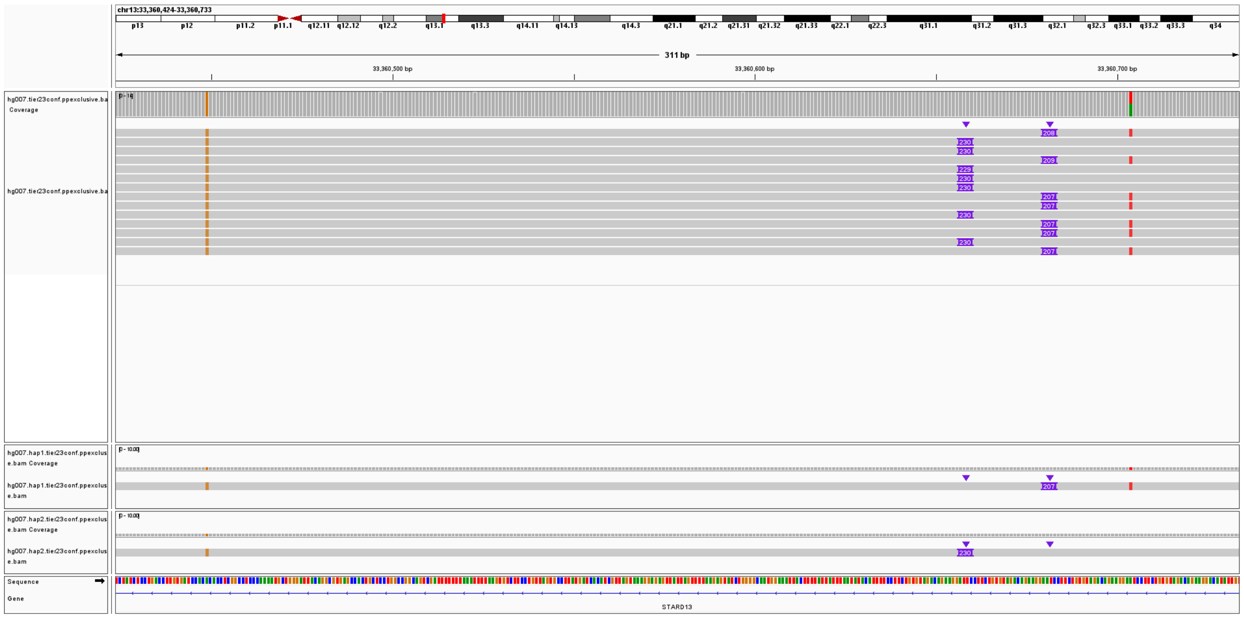


Figure 4.8: Heterozygous deletion in the HG007 sample (`chr7:32786841`). `dipcall` called two alleles of length 51 and 75. `SVDSS` agreed with `dipcall` correctly calling both alleles. `cuteSV`, `pbsv`, `sniffles`, `debreak` and `SVIM`, instead, called just one allele of length 63, 75, 75, and 63, respectively.



chr13:33,360,424-33,360,733
311 bp
33,360,500 bp
33,360,600 bp
33,360,700 bp
hg007.tier23conf.ppexclusive.bam Coverage
hg007.tier23conf.ppexclusive.bam
hg007.hap1.tier23conf.ppexclusive.bam Coverage
hg007.hap1.tier23conf.ppexclusive.bam
hg007.hap2.tier23conf.ppexclusive.bam Coverage
hg007.hap2.tier23conf.ppexclusive.bam
Sequence
Gene
STARD13

Figure 4.9: Two close insertion alleles in the HG007 sample (`chr13:33360658` and `chr13:33360681`). `dipcall` called two alleles of length 207 and 230. `SVDSS` agreed with `dipcall` correctly calling both alleles. `cuteSV`, `pbsv`, `sniffles`, `debreak` and `SVIM`, instead, called just one allele of length 218, 230, 230, and 218, respectively.

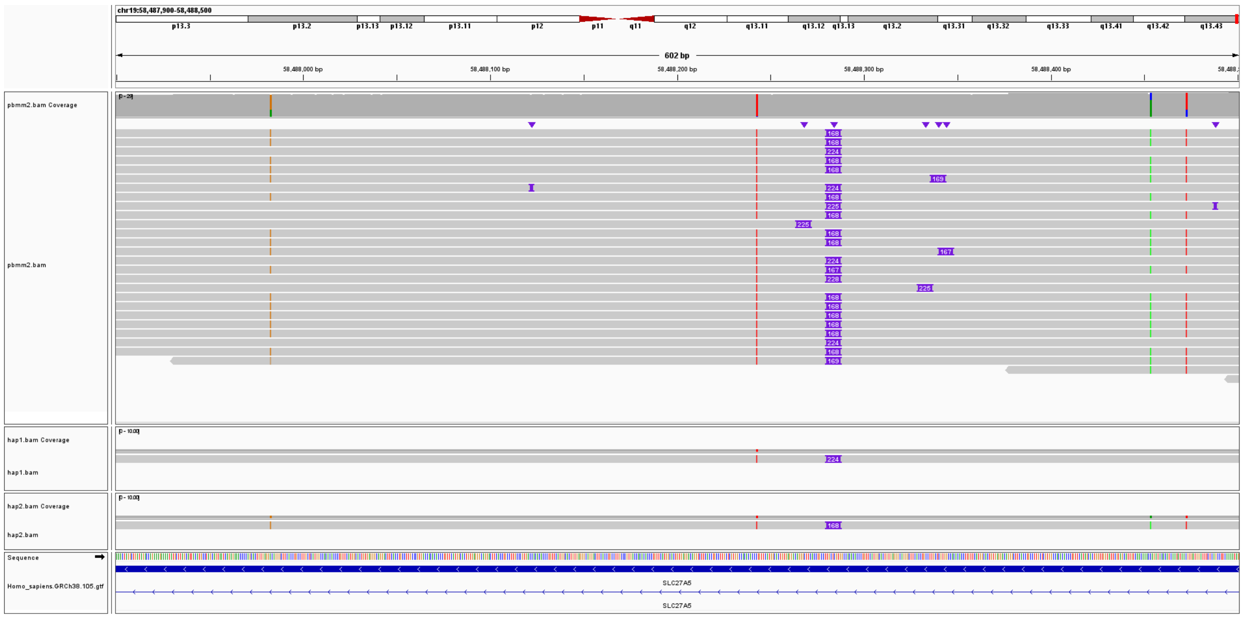


Figure 4.10: Full IGV image for the double insertion falling in the SLC27A5 medically-relevant gene.

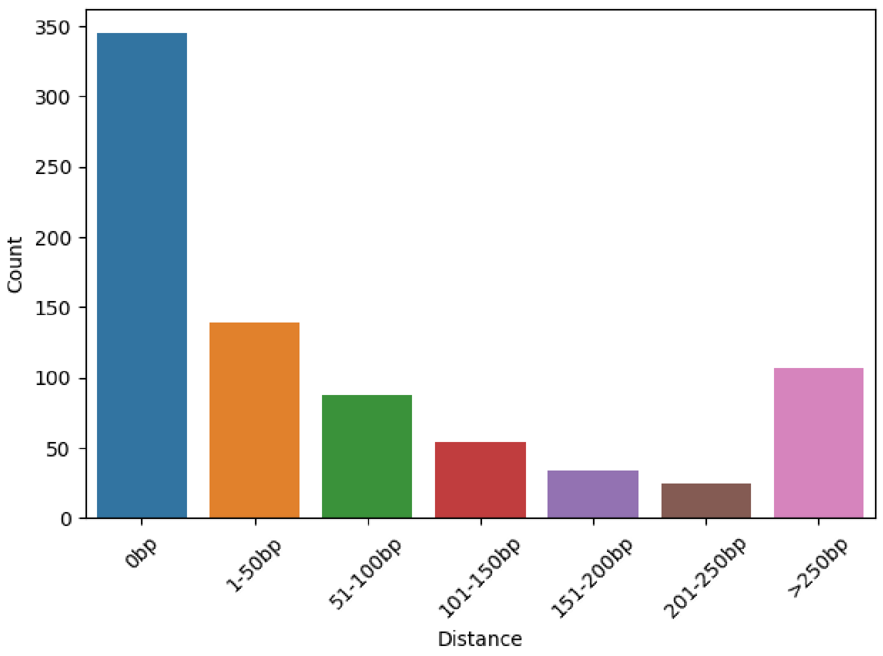


Figure 4.11: Bar plots showing the distance (in basepairs) of the SVs exclusively called by `SVDSS` and the closest SV. A distance of 0bp means that the SV is a heterozygous non-reference SV (i.e., a SV with two alleles and genotyped 1/2).

cludes a summary of the result. We calculate the baseline False Discovery Rate for each tool as the number of calls it makes against T2T divided by the number of calls it makes against GRCh38. `SVDSS` has the lowest number of calls against the T2T assembly and also has the lowest baseline error rate at.

We further investigated if any of our calls are indeed true variants. The T2T project provides a list of known heterozygous sites on CHM13 [3] and 13 of our SV calls intersect these regions, suggesting that may be actual heterozygous alleles missing from the homozygous assembly. We also report the number of intersecting calls in Table 4.4 for every tool. `SVDSS` has the highest ratio of calls intersecting known heterozygous regions. We did additional filtering on the calls using Merfin [50], a variant call polishing tool that filters VCF files based on whether the variants introduce *k*-mers not found in the sequencing reads. Only one of our calls passes Merfin's filtering and we verify that the call seems to be a heterozygous site (Figure 4.13).

[3] `https://github.com/marbl/CHM13-issues`

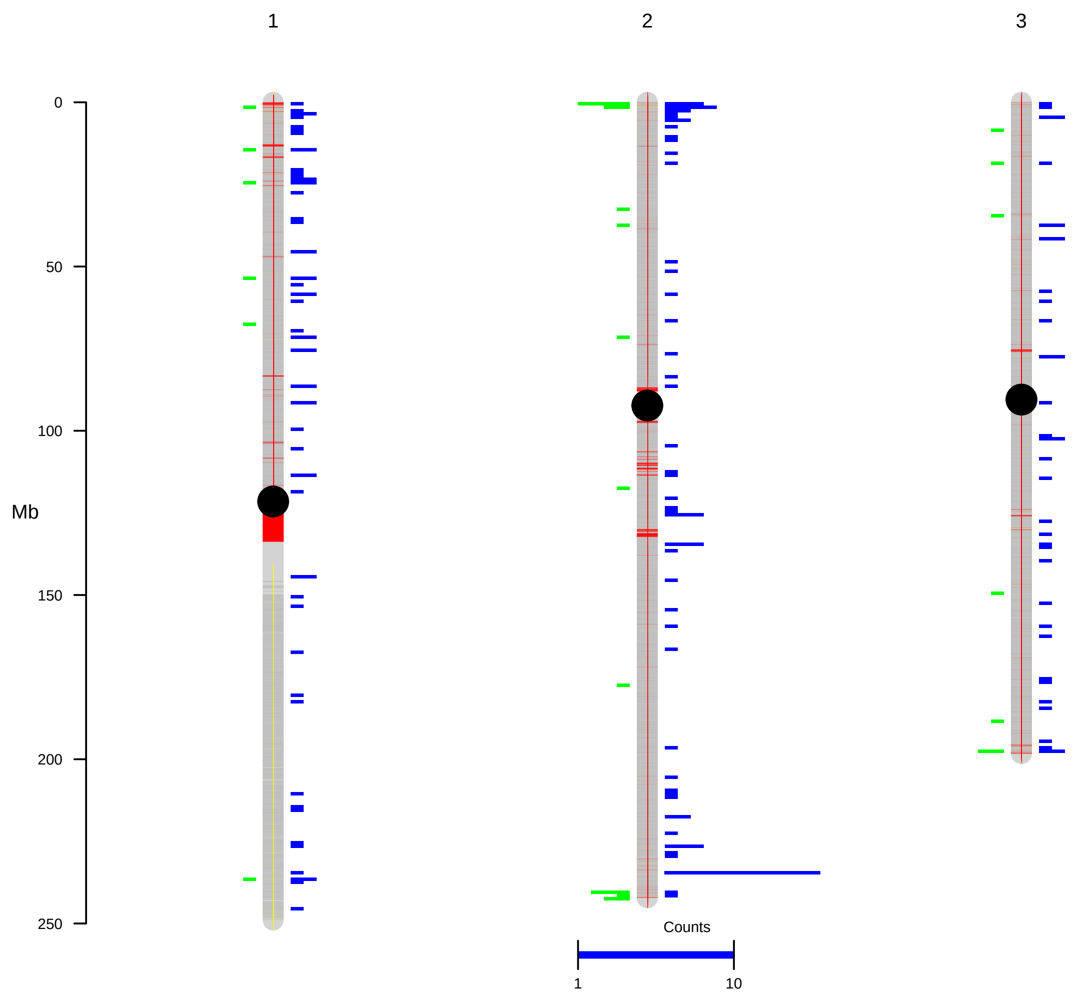


Figure 4.12: Comparison of SVDSS-specific and SVIM-specific Calls on Extended Tier 2 (red) on HG007. SVDSS (blue) has the highest number of specific calls on HG007 (739) while SVIM (green) has the second highest number of such calls (130).

Tool	GRCh38 calls	T2T calls	Baseline FDR	Het Intersections	Het Precision
`SVDSS`	**23777**	**102**	**0.4%**	13	**12.7%**
`cuteSV`	22654	667	2.94%	23	3.4%
`pbsv`	23707	616	2.59%	28	4.5%
`sniffles`	22680	314	1.38%	22	7.0%
`SVIM`	22176	948	4.27%	**29**	3.0%
`debreak`	23432	834	3.55%	24	2.8%

Table 4.4: Comparison of baseline FDR rate of `SVDSS` and other methods. Number of SV calls against both the reference genome and the CHM13 assembly is included. Baseline FDR is calculated as division of first two columns for each tool. The last two columns report the number of known CHM13 heterozygous (het) sites covered by each method and the precision of the method based on the number of covered heterzoygous sites.

In summary, `SVDSS` produces only 102 SV calls against the CHM13 assembly, some of which may be actual true heterozygous variants. Our analysis doesn't result in discovery of any homozygous SVs missed by the T2T assembly, however, this was expected given the quality of the assembly. Furthermore, with our pipeline producing 23777 SV calls on CHM13 against GRCh38, this amounts to a baseline error rate of less than 0.4% showing that `SVDSS` is robust to false detection of variants.

4.3.4 Discovery of SVs with Clinical Importance

Finally, to perform a more thorough analysis of the HG002 individual, we considered the CMRG (Challenging Medically Relevant Genes) callset provided in [116] and we evaluated callers' accuracy against it. The CMRG callset consists of 250 SVs falling in 126 challenging and medically relevant genes that were excluded from the previously published GIAB benchmark [106] due to their complexity: compound heterozygous insertions, complex variants in segmental duplications, and long tandem repeats. The CMRG callset was created by diploid assembly of the haplotypes using hifiasm and then `dipcall`, proving one more time the effectiveness of assembly-based meth-

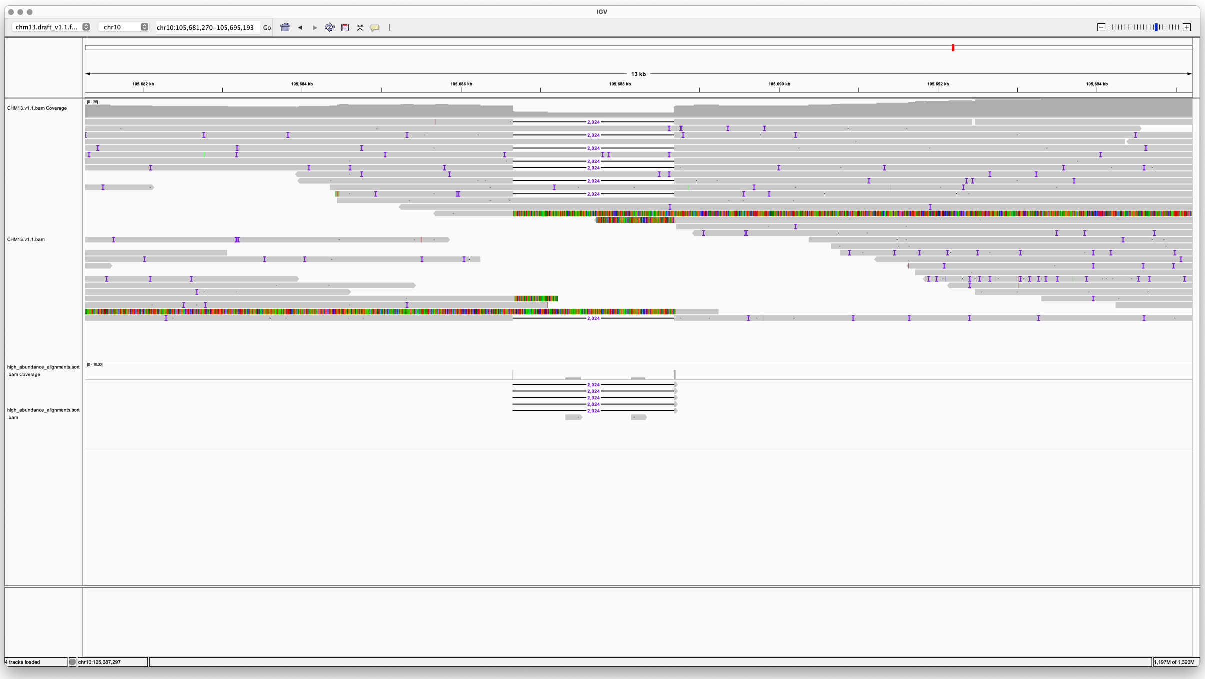


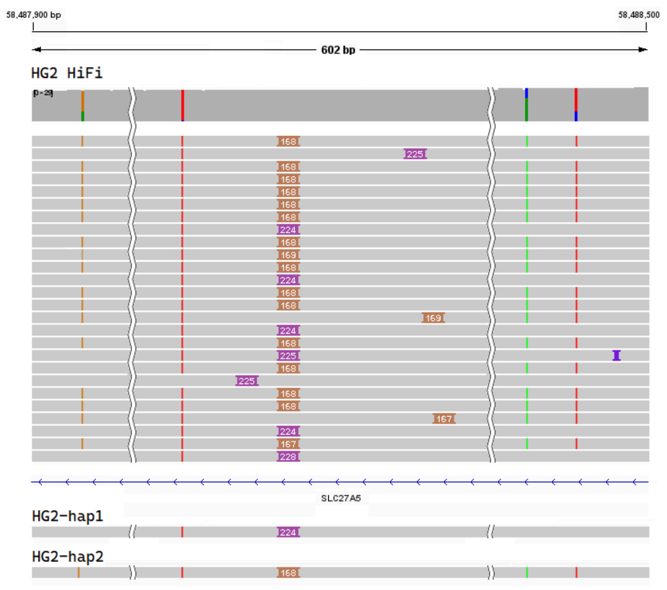
Figure 4.13: Example of heterozygous SV detected by our pipeline on CHM13 HiFi reads.

ods for hard-to-analyze SVs.

As done previously, we computed the accuracy of SVDSS and the other 4 mapping-based SV callers using Truvari. Out of the 250 SVs contained in the CMRG callset, SVDSS correctly called 232 SVs followed by pbsv (228) and cuteSV (225), sniffles (218) and SVIM (218). As shown in Figure 4.6(d), 5 SVs are exclusive to SVDSS, while 2 are missed exclusively by SVDSS: one was reported but with a length just under the evaluation threshold of Truvari, the other was missed due to being only detectable in clipped reads, which SVDSS does not consider by default. We then manually investigated the SVs that were exclusively called by SVDSS, discovering that all them exhibited two alleles, one per haplotype (i.e., heterozygous non-reference SVs).

This result confirms previous findings [116] that heterozygous insertions in tandem repeats are among the most challenging classes of SVs to discover with current methods.

Figure 4.14 shows one of the SVDSS-exclusive SVs, a double insertion inside the SLC27A5 gene on chromosome 19. Although the two haplotypes can be easily distinguished from read alignments, thanks to the adjacent heterozygous SNPs, the tested callers disagree on which allele



```
CMRG:     chr19:58488284   SVLEN=168,224   GT  1/2
SVDSS:    chr19:58488284   SVLEN=168,224   GT  1/2
cuteSV:   chr19:58488291   SVLEN=185       GT  0/1
pbsv:     chr19:58488284   SVLEN=168       GT  0/1
svim:     chr19:58488290   SVLEN=185       GT  0/1
sniffles: chr19:58488285   SVLEN=168       GT  0/1
debreak:  chr19:58488290   SVLEN=185       GT  1/1
```

Figure 4.14: Example of SV falling on medically-relevant genes that have been correctly called exclusively by SVDSS. On the right panel, IGV sketch of the 602bp region around the SV (the full region is reported in Figure 4.10). The sketch reports the HiFi reads alignment along with the haplotype alignment performed using minimap2 (as part of the dipcall pipeline). On the left panel, details on the SVs reported by the CMRG callset, SVDSS, and the other alignment-based callers considered in our evaluation.

to call. For instance only SVDSS calls two alleles of length 168bp and 224bp agreeing with the CMRG callset, whereas pbsv and sniffles report only one of the two (168bp). Surprisingly, cuteSV and SVIM report a single allele of length 185bp, which does not match any of the evidence from read alignment. Additionally, we considered the portion of the high-quality HG002 assembly covering that locus (chr19:58487900-58488500) and we checked its alignment against the reference genome (Figure 4.14 and 4.15). Although the considered locus is in a repetitive region (as also proven by the noisiness of the dotplots), the haplotype alignment confirm the presence of two alleles of different size.

4.4 Implementation Notes

The exact version of the Ping-Pong algorithm as presented in Chapter 3 is very computationally intensive. This means that the time it takes to load a batch of reads from a BAM or FASTQ file (I/O overhead) is negligible compared to the time it takes to process them. With the introduction

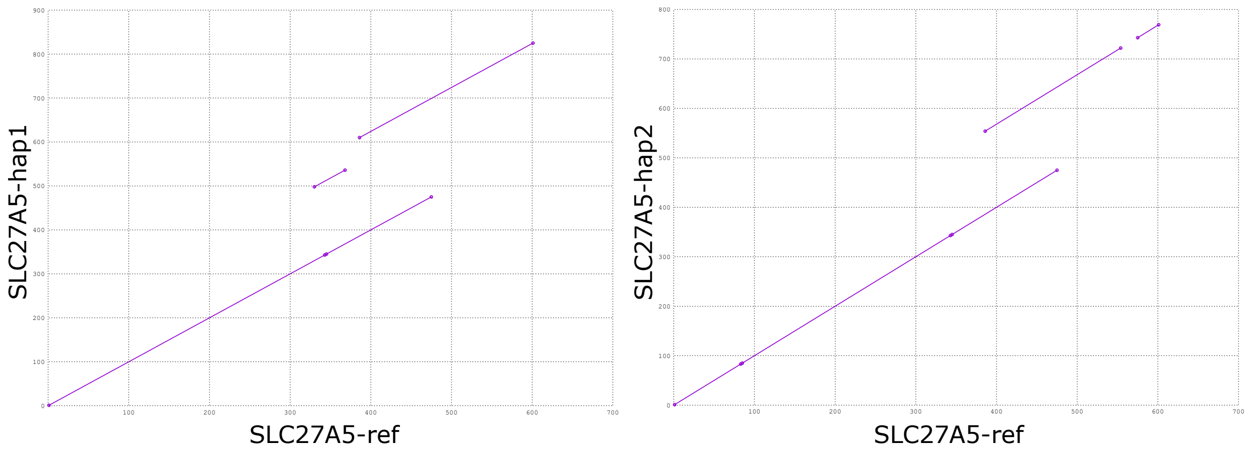


Figure 4.15: Dotplots of the alignment between the two HG002 high-quality haplotypes and the GRCh38 reference genome around the heterozygous SV falling in the medically-relevant gene SLC27A5 (locus: `chr19:58487900-58488500`).

of read smoothing and the resulting reduction in number of extracted SFS, processing times become fast enough that I/O overhead becomes significant. As a result, we have introduced several optimizations to improve the algorithm's throughput.

4.4.1 Dual and Triple Buffering

Ping-Pong algorithm works by loading batches of reads from input files and processing them in parallel. The I/O performance can be improved by parallelizing the loading of next batch of reads with the processing of current batch. We refer to this optimization as "dual buffering". Two buffers are allocated in memory, each holding a batch of SFS. The input file is processed in a `while` loop: at each iteration one buffer is processed while the other buffer is being populated with new reads. At the next iteration the buffer pointers are swapped and the newly-populated buffer is processed while the other buffer is flushed and reloaded.

During smoothing, processed reads have to be written back to the BAM file. This necessitates the introduction of a third buffer. At any iteration, reads in one buffer are being smoothed, the second buffer is being populated and the third buffer is being written to the output file and flushed afterwards. Figure 4.16 below shows this triple-buffering scheme in practice.

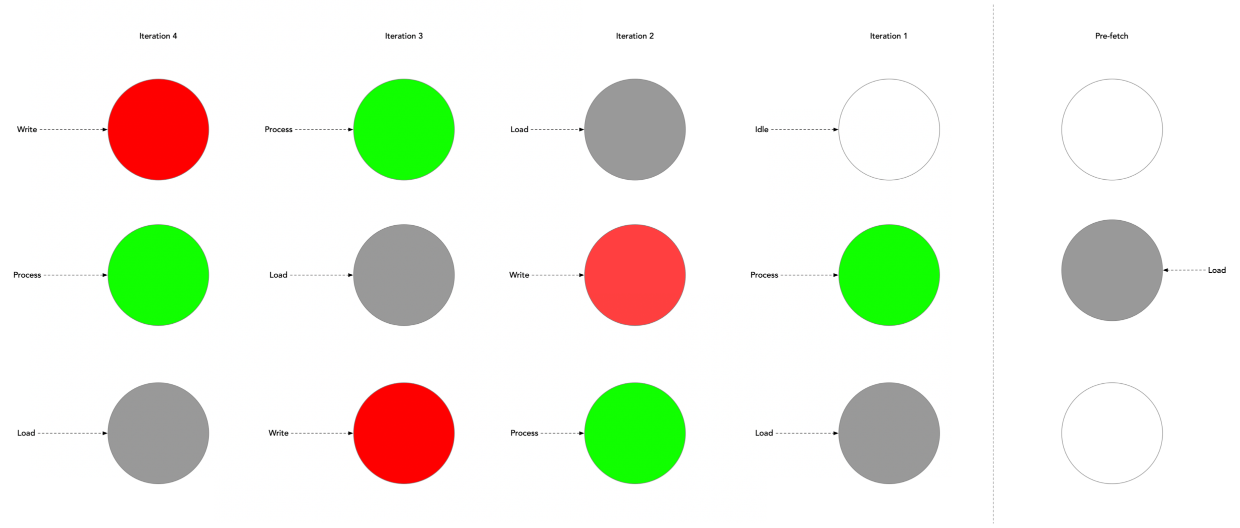


Figure 4.16: Triple-buffering for read smoothing. One buffer is processed while the other two buffers are populated and written out respectively. The buffers switch roles each iteration

4.4.2 Multi-thread BAM Decoding

While the multi-buffering optimizations significantly improve our performance, their impact is still limited by the overhead of decoding BAM files. By default, “htslib” loads one BAM entry from the file at a time, however it can be configured to use multiple reads when decoding reads by setting `bgzf_mt` flag when opening a BAM file. Using 8 internal decoder threads makes BAM decoding up to 5 times faster. We can make BAM I/O even faster by building `htslib` with `libdeflate`. This replaces the internal decoder implementation with a more efficient one, making BAM processing around 3 times faster.

When combined, these optimizations yield a 15x speed improvement in loading BAM files. The massive speedup means that once again I/O overhead becomes insignificant compared to compute overhead and so we don’t expect to see the same level of speedup in overall performance. In our experiments, SFS extraction becomes about 9x faster than before with the above optimizations, while read smoothing becomes 6-7 times faster.

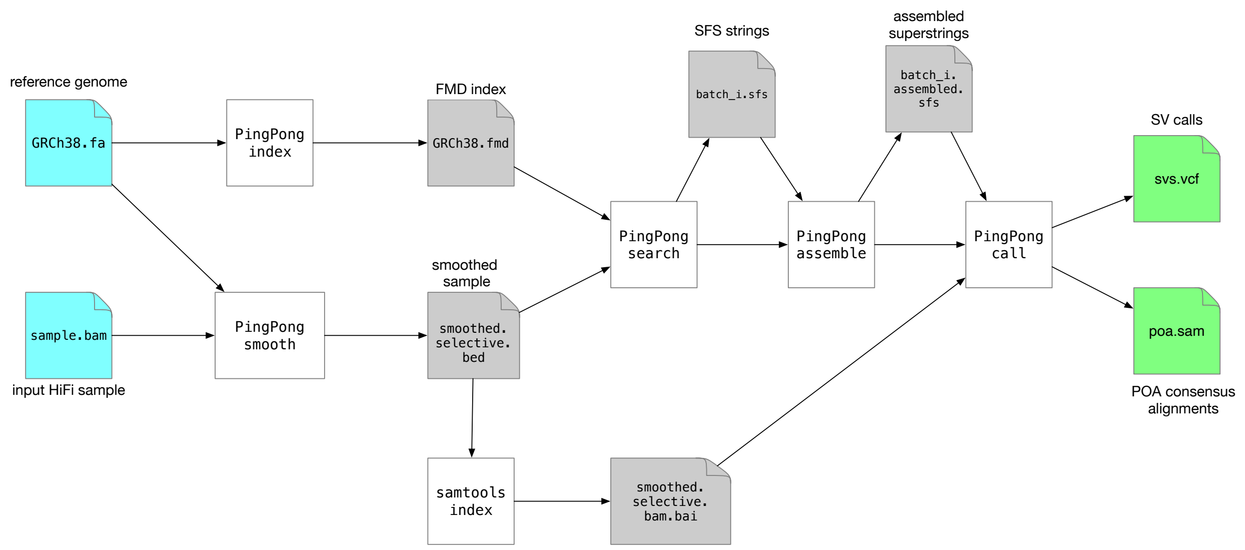


Figure 4.17: Overview of the SVDSS pipeline. Input files are shown in blue, output files are shown in green and intermediate and internal files are showed in grey. Square boxes show commands. The reference genome is FMD-indexed and the HiFi BAM sample is smoothed, then SFS are extracted and assembled into superstrings and finally SV calls are made from the superstrings. The only external command needed is `samtools index` to re-index the smoothed BAM file. The FMD index can be reused once created for a reference genome.

4.4.3 Command-line Usage

Figure 4.17 shows the complete pipeline for genotyping a sample `sample.bam` mapped to reference genome `GRCh38.fa` with `SVDSS`. All parts of the pipeline are implemented as subcommands of the `SVDSS` package. The only needed external command is ‘samtools index‘ for indexing the smoothed BAM file. Extensive documentation for using the package is available on the Github repostiory https://github.com/Parsoa/SVDSS.

4.4.4 Runtime

`SVDSS` is a generally compute-intensive method due to the theoretical complexity of SFS extraction with the Ping-Pong algorithm, POA and local alignment. However, with deep parallelization and careful optimizations, our runtime remains low compared to similar methods. We provide a breakdown of the runtime of different stages below:

- The FMD index creation and querying are handled internally by the FMD implementation from [85] and have not been further optimized. FMD index creation for the GRC38 reference genome takes around 30 minutes with 16 cores. The index can be reused for any number of samples so its creation is a one-time expense.

- read smoothing takes only about 15 minutes to run on 16 cores for a 30x Hi-Fi sample (CHM13 in our experiments).

- SFS extraction is the most computationally intensive step and takes about 45 minutes on 16 threads for the CHM13 HiFi data. Increasing the number of threads to 48 bring the runtime down to 25 minutes. Due to the heavy I/O load of SFS extraction, increasing the number of threads does not necessarily bring linear time improvements.

- Finally, the SV calling steps is extremely fast and takes less than 8 minutes.

Overall the runtime of the `SVDSS` pipeline is less than 70 minutes for a high-coverage (30x) HiFi sample on 16 cores excluding index creation time. The experiments in this chapter were performed on a computer with a Intel(R) Xeon(R) processors with 24 cores and 256GB of RAM.

For comparison, all the other methods tested in this chapter with the exception of `cuteSV` took more than 90 minutes to run using the same number of threads and runtime options (where allowed) to genotype the CHM13 samples. `cuteSV` was the fastest method and impressively took only 5 minutes to genotype a sample. The slowest method was Sniffles, taking upwards of 4 hours to genotype CHM13.

All compared methods used less than 64GB of memory during their runtime. `SVDSS` peaks at 34GB of memory during the SV calling stage due to holding several (depending on the number of threads) POA graphs and local alignment dynamic programming matrices in memory. The smoothing and SFS extraction step each use constant amount of memory. The smoothing stage keeps 3 batches of 10000 BAM reads each in memory at any given time while the Ping-Pong algorithm keeps 2 batches of 10000 reads each plus up to 10 million SFS pointers (read ID, start and end positions) in memory.

4.5 Discussion

In this chapter we introduced SVDSS, a hybrid method for SV discovery that combines advantages of different SV discovery approaches to achieve significant improvements in SV calling. Our method achieves much higher recall compared to state-of-the-art approaches in repeated regions of the genome (i.e., Extended Tier 2) while maintaining equal precision. Furthermore, SVDSS is also more robust to low coverage samples compared to other methods and displays significantly lower baseline error rate.

While the availability of low-error long-read data enables more extensive variant discovery on new samples, SV discovery in repetitive regions of the genome such as STRs and microsatellites remains both challenging and hard to evaluate. Despite SVDSS's significant performance improvements in repetitive regions, precision and recall in these regions are still lower than on the rest of the genome.

SVDSS currently supports the discovery of unbalanced SVs, i.e. deletions and insertions, however as the underlying SFS signatures capture nearly all variation in the genome, a next step could be to extend the method to finding other classes of SVs such as inversions and duplications. Our current approach for building a SV truth set from assemblies (`dipcall`) does not evaluate inversions and duplications, yet a recent study [117] provides one of the first gold standards.

We also highlight the importance of accurate benchmarks for SV calling methods. We evaluated SVDSS on a recent benchmark extensively curated over the HG002 sample [116] with the specific purpose of producing SVs occurring in genes of medical relevance, which are still considered challenging for mapping-based and assembly-based SV prediction methods even with highly accurate long reads. This benchmark revealed that while other methods fail to call heterozygous SVs in these regions, SVDSS was able to discover 5 heterozygous SVs in medically relevant gene regions.

4.6 Code and Data Availability

The implementation of the SVDSS algorithm along with detailed usage manual is publicly available at https://github.com/Parsoa/SVDSS.

The results in this chapter are pending journal review as of the time of this thesis's completion. A pre-print is available on bioRxiv at https://www.biorxiv.org/content/10.1101/2022.02.12.480198v1

4.7 Contributions

This chapter is a result of collaboration with colleagues Dr. Luca Denti [4], Dr. Rayan Chikhi[5] and Dr. Paola Bonizonni[6].

Dr. Denti and I are recognized as first authors. Both first authors equally contributed to development of the method and preparation of the results and experiments.

4.8 Funding

This work was partially funded by NSF award DBI-2042518 to Fereydoun Hormozdiari.

[4]luca.denti@pasteur.fr, Department of Computational Biology, Institut Pasteur, Paris, France

[5]rayan.chikhi@pasteur.fr, Department of Computational Biology, Institut Pasteur, Paris, France

[6]paola.bonizzoni@unimib.it, Department of Informatics, Systems and Communication, University of Milano-Bicocca, Milano, Italy

Chapter 5

Conclusion and Future Works

5.1 Conclusions

Throughout this dissertation, we have explored the applicability of mapping-free and sequence-based methods for analysis of SVs in both short and long-read data. This chapter summarizes the key findings of the presented works and proposes future research directions.

In Chapter 2 we presented Nebula, a k-mer-based mapping-free method for genotyping common SVs on Illumina short-read data that is an order of magnitude faster than similar approaches and can detect all types of variants using a universal k-mer-counting scheme. We showed that Nebula matches state-of-the-art mapping-based methods in accuracy and offers significant runtime advantage, in particular for large-scale studies. The method is theoretically applicable to other organisms as well, however we have not tested it on non-human data.

In Chapter 3 we presented the notion of SFS as an effective mapping-free tool for capturing effectively all variation in one PacBio Hi-Fi sample with regards to another sample or the reference genome. We introduced the theoretically-optimal Ping-Pong algorithm for SFS extraction and presented an efficient and highly-parallelized implementation of the algorithm for our experiments. Finally, we experimentally verified that the SFS framework can indeed detect nearly-all ($> 98\%$) between a pair or trio of HiFi samples in both real and simulated data.

Finally, in chapter 4 we introduced `SVDSS`, a hybrid method that incorporates advantages of mapping-based, mapping-free and assembly-based approaches for discovery of deletion and insertion SVs in PacBio HiFi sample. SVDSS uses the concept of SFS and the Ping-Pong algorithm from Chapter 3 along with partial-order assembly and local alignment to achieve significant improvements in recall and precision when genotyping SVs from highly-repetitive regions of the genome. We experimentally validated SVDSS's performance on three PacBio HiFi samples and showed that it consistently outperforms state-of-the-art purely mapping-based methods.

Despite the progress in quality of long-read data and the accuracy of computational methods, SV discovery in repetitive regions of the genome such as STRs and microsatellites is still a challenge. This is evidenced by comparisons presented in Section 2.3, Section 3.4.2 and Section 4.3. Despite SVDSS's significant performance improvements in repetitive regions, precision and recall in these regions are still lower than the rest of the genome. We believe that improving SV calling performance in these regions should be the focus of the community's efforts in the near future.

5.2 Future Works

There are several different directions to extend and expand on the works presented here. In particular, we believe there are many potential venues for extending the SFS framework presented in Chapter 3 beyond what we have already explored. We have discussed few of them below.

5.2.1 Improved SV Calling with Nebula using SNP information

The *k*-mer-counting based framework presented in Chapter 2 can be further extended for targeted genotyping of different types of variants including SNPs and small INDELs in short-read samples. Furthermore this allows incorporation of SNP-imputation information to the pipeline and can provide additional signal for genotyping SVs that may otherwise be difficult to genotype alone with *k*-mers. The inclusion of SNP discovery could also make Nebula more useful for large association studies that aim to test many samples for presence or absence of certain variants.

5.2.2 Variant Call and Assembly Polishing with SFS

As SFS will cover nearly all variations in a sample, a potential application of the method would be the comparative discovery of variants that are missed using traditional mapping-based approaches between two samples. The SFS remaining after filtering the SFS covering the variants predicted using a mapping-based approach may indicate potentially novel variants missed by that method.

SFS can also be used as a method for measuring quality of genome assemblies. By building the FMD index of the assembly and querying the sequencing data used to build the assembly against the index, one can potentially find variants that are missed by the assembly or other structural imperfections. SFS could be more flexible than fixed-length *k*-mers in this application due to their ability to cover long mismatches.

5.2.3 Deeper Theoretical Study of SFS

A potential venue for more theoretical research could be to investigate the connection between SFS strings and other related but different concepts in stringology, such as maximal exact/unique matches, minimum unique substrings [118], and shortest uncommon superstrings.

5.2.4 Inversion and Complex SV Discovery with `SVDSS`

We have so far utilized SFS for discovery of insertions and deletions, however as SFS capture nearly all variants regardless of type, the `SVDSS` framework could be extended to allow for detection of more complex types of SVs as as well. Inversions are generally difficult to detect using mapping-free methods due to the reverse-complemented inversion sequence producing the same canonical substrings as the non-inverted sequence. However, inversions can be detected using SFS by building two FMD indices, one for each direction of the reference genome and by detecting the presence of SFS from one direction on reads that are originating from the opposite haplotype. This allows for detection of the presence of inversions, however additional processing is required to establish the exact bounds of the variant which can be done by performing local assembly of reads and checking for the start position of the inversion base-base, guided by the placement of SFS sequences.

5.2.5 STR and VNTR Expansion Estimation Using RNNs

Short Tandem Repeats (STRs) are formed by multiple consecutive repetitive occurrences of small DNA motifs also called repeat units in the genome. The repeat units are usually short 2bp or 3bp motifs but can also be longer. Mutations in STRs have been linked to several diseases and genetic disorders such as Huntington's disease [119]. These mutations often manifest as positive or negative changes in the length of the repeat array, referred to as “expansions”. Tandem repeats with larger repeat units ($>$ 6bp) are called Variable-Length Tandem Repeats (VNTR). VNTRs can occur in coding regions of the genome and sometimes close to genes [120], and as a result, changes in their lengths can have significant functional effects. VNTRs have also been linked to many genetic disorders [121].

While many methods exists for genotyping of STRs [122, 123], relatively few methods have been developed for the specific problem of VNTR genotyping. Deep learning-based approaches have been recently applied with great effectiveness to biological problems such as SNP genotyping [124] and the long-unsolved problem of 3D protein structure prediction [125]. We believe that the application of these methods to the problem of STR and VNTR expansion could allow significant improvements over current methods.

References

[1] Peter H Sudmant, Tobias Rausch, Eugene J Gardner, Robert E Handsaker, Alexej Abyzov, John Huddleston, Yan Zhang, Kai Ye, Goo Jun, Markus Hsi-Yang Fritz, et al. An integrated map of structural variation in 2,504 human genomes. *Nature*, 526(7571):75–81, 2015.

[2] Paweł Stankiewicz and James R Lupski. Structural variation in the human genome and its role in disease. *Annual review of medicine*, 61:437–455, 2010.

[3] Andrew J Sharp, Ze Cheng, and Evan E Eichler. Structural variation of the human genome. *Annu. Rev. Genomics Hum. Genet.*, 7:407–442, 2006.

[4] Ryan L Collins, Harrison Brand, Konrad J Karczewski, Xuefang Zhao, Jessica Alföldi, Laurent C Francioli, Amit V Khera, Chelsea Lowther, Laura D Gauthier, Harold Wang, et al. A structural variation reference for medical and population genetics. *Nature*, 581 (7809):444–451, 2020.

[5] Peter H Sudmant, Swapan Mallick, Bradley J Nelson, Fereydoun Hormozdiari, Niklas Krumm, John Huddleston, Bradley P Coe, Carl Baker, Susanne Nordenfelt, Michael Bamshad, et al. Global diversity, population stratification, and selection of human copy-number variation. *Science*, 349(6253), 2015.

[6] Peter H Sudmant, John Huddleston, Claudia R Catacchio, Maika Malig, LaDeana W Hillier, Carl Baker, Kiana Mohajeri, Ivanela Kondova, Ronald E Bontrop, Stephan Persengiev, et al. Evolution and diversity of copy number variation in the great ape lineage. *Genome research*, 23(9):1373–1382, 2013.

[7] ICGC The, TCGA Pan-Cancer Analysis of Whole, Genomes Consortium, et al. Pan-cancer analysis of whole genomes. *Nature*, 578(7793):82, 2020.

[8] Yilong Li, Nicola D Roberts, Jeremiah A Wala, Ofer Shapira, Steven E Schumacher, Kiran Kumar, Ekta Khurana, Sebastian Waszak, Jan O Korbel, James E Haber, et al. Patterns of somatic structural variation in human cancer genomes. *Nature*, 578(7793):112–121, 2020.

[9] Kai Ye, Jiayin Wang, Reyka Jayasinghe, Eric-Wubbo Lameijer, Joshua F McMichael, Jie Ning, Michael D McLellan, Mingchao Xie, Song Cao, Venkata Yellapantula, et al. Systematic discovery of complex insertions and deletions in human cancers. *Nature medicine*, 22 (1):97–104, 2016.

[10] Emma C Scott, Eugene J Gardner, Ashiq Masood, Nelson T Chuang, Paula M Vertino, and Scott E Devine. A hot l1 retrotransposon evades somatic repression and initiates human colorectal cancer. *Genome research*, 26(6):745–755, 2016.

[11] Jeremiah A Wala, Pratiti Bandopadhayay, Noah F Greenwald, Ryan O'Rourke, Ted Sharpe, Chip Stewart, Steve Schumacher, Yilong Li, Joachim Weischenfeldt, Xiaotong Yao, et al. Svaba: genome-wide detection of structural variants and indels by local assembly. *Genome research*, 28(4):581–591, 2018.

[12] Tom Walsh, Jon M McClellan, Shane E McCarthy, Anjené M Addington, Sarah B Pierce, Greg M Cooper, Alex S Nord, Mary Kusenda, Dheeraj Malhotra, Abhishek Bhandari, et al. Rare structural variants disrupt multiple genes in neurodevelopmental pathways in schizophrenia. *science*, 320(5875):539–543, 2008.

[13] Donald F Conrad, Dalila Pinto, Richard Redon, Lars Feuk, Omer Gokcumen, Yujun Zhang, Jan Aerts, T Daniel Andrews, Chris Barnes, Peter Campbell, et al. Origins and functional impact of copy number variation in the human genome. *Nature*, 464(7289):704–712, 2010.

[14] Christian R Marshall, Abdul Noor, John B Vincent, Anath C Lionel, Lars Feuk, Jennifer Skaug, Mary Shago, Rainald Moessner, Dalila Pinto, Yan Ren, et al. Structural variation of chromosomes in autism spectrum disorder. *The American Journal of Human Genetics*, 82 (2):477–488, 2008.

[15] Mark JP Chaisson, John Huddleston, Megan Y Dennis, Peter H Sudmant, Maika Malig, Fereydoun Hormozdiari, Francesca Antonacci, Urvashi Surti, Richard Sandstrom, Matthew Boitano, et al. Resolving the complexity of the human genome using single-molecule sequencing. *Nature*, 517(7536):608–611, 2015.

[16] Peter Ebert, Peter A Audano, Qihui Zhu, Bernardo Rodriguez-Martin, David Porubsky, Marc Jan Bonder, Arvis Sulovari, Jana Ebler, Weichen Zhou, Rebecca Serra Mari, et al. Haplotype-resolved diverse human genomes and integrated analysis of structural variation. *Science*, 372(6537), 2021.

[17] Michael M Khayat, Sayed Mohammad Ebrahim Sahraeian, Samantha Zarate, Andrew Carroll, Huixiao Hong, Bohu Pan, Leming Shi, Richard A Gibbs, Marghoob Mohiyuddin, Yuanting Zheng, et al. Hidden biases in germline structural variant detection. *Genome biology*, 22(1):1–15, 2021.

[18] Claudia MB Carvalho and James R Lupski. Mechanisms underlying structural variant formation in genomic disorders. *Nature Reviews Genetics*, 17(4):224–238, 2016.

[19] Peter A Audano, Arvis Sulovari, Tina A Graves-Lindsay, Stuart Cantsilieris, Melanie Sorensen, AnneMarie E Welch, Max L Dougherty, Bradley J Nelson, Ankeeta Shah, Susan K Dutcher, et al. Characterizing the major structural variant alleles of the human genome. *Cell*, 176(3):663–675, 2019.

[20] Xuefang Zhao, Ryan L Collins, Wan-Ping Lee, Alexandra M Weber, Yukyung Jun, Qihui Zhu, Ben Weisburd, Yongqing Huang, Peter A Audano, Harold Wang, et al. Expectations and blind spots for structural variation detection from long-read assemblies and short-read

genome sequencing technologies. *The American Journal of Human Genetics*, 108(5):919–928, 2021.

[21] Jay Shendure and Hanlee Ji. Next-generation dna sequencing. *Nature biotechnology*, 26 (10):1135–1145, 2008.

[22] Teri A Manolio, Francis S Collins, Nancy J Cox, David B Goldstein, Lucia A Hindorff, David J Hunter, Mark I McCarthy, Erin M Ramos, Lon R Cardon, Aravinda Chakravarti, et al. Finding the missing heritability of complex diseases. *Nature*, 461(7265):747, 2009.

[23] Evan E Eichler, Jonathan Flint, Greg Gibson, Augustine Kong, Suzanne M Leal, Jason H Moore, and Joseph H Nadeau. Missing heritability and strategies for finding the underlying causes of complex disease. *Nature Reviews Genetics*, 11(6):446, 2010.

[24] Elizabeth T Cirulli and David B Goldstein. Uncovering the roles of rare variants in common disease through whole-genome sequencing. *Nature Reviews Genetics*, 11(6):415–425, 2010.

[25] 1000 Genomes Project Consortium et al. A global reference for human genetic variation. *Nature*, 526(7571):68–74, 2015.

[26] Swapan Mallick, Heng Li, Mark Lipson, Iain Mathieson, Melissa Gymrek, Fernando Racimo, Mengyao Zhao, Niru Chennagiri, Susanne Nordenfelt, Arti Tandon, et al. The simons genome diversity project: 300 genomes from 142 diverse populations. *Nature*, 538 (7624):201–206, 2016.

[27] Javier Prado-Martinez, Peter H Sudmant, Jeffrey M Kidd, Heng Li, Joanna L Kelley, Belen Lorente-Galdos, Krishna R Veeramah, August E Woerner, Timothy D O'Connor, Gabriel Santpere, et al. Great ape genetic diversity and population history. *Nature*, 499(7459): 471–475, 2013.

[28] Genome 10K Community of Scientists. Genome 10k: a proposal to obtain whole-genome sequence for 10 000 vertebrate species. *Journal of Heredity*, 100(6):659–674, 2009.

[29] Carl Kingsford, Michael C Schatz, and Mihai Pop. Assembly complexity of prokaryotic genomes using short reads. *BMC bioinformatics*, 11(1):21, 2010.

[30] Glennis A Logsdon, Mitchell R Vollger, and Evan E Eichler. Long-read human genome sequencing and its applications. *Nature Reviews Genetics*, pages 1–18, 2020.

[31] Justin M Zook, Nancy F Hansen, Nathan D Olson, Lesley Chapman, James C Mullikin, Chunlin Xiao, Stephen Sherry, Sergey Koren, Adam M Phillippy, Paul C Boutros, et al. A robust benchmark for detection of germline large deletions and insertions. *Nature biotechnology*, 38(11):1347–1355, 2020.

[32] Arda Soylev, Thong Minh Le, Hajar Amini, Can Alkan, and Fereydoun Hormozdiari. Discovery of tandem and interspersed segmental duplications using high-throughput sequencing. *Bioinformatics*, 35(20):3923–3930, 2019.

[33] Mark JP Chaisson, Ashley D Sanders, Xuefang Zhao, Ankit Malhotra, David Porubsky, Tobias Rausch, Eugene J Gardner, Oscar L Rodriguez, Li Guo, Ryan L Collins, et al. Multi-platform discovery of haplotype-resolved structural variation in human genomes. *Nature communications*, 10(1):1–16, 2019.

[34] Mehrdad Bakhtiari, Jonghun Park, Yuan-Chun Ding, Sharona Shleizer-Burko, Susan L Neuhausen, Bjarni V Halldórsson, Kári Stefánsson, Melissa Gymrek, and Vineet Bafna. Variable number tandem repeats mediate the expression of proximal genes. *Nature communications*, 12(1):1–12, 2021.

[35] Ryan M Layer, Colby Chiang, Aaron R Quinlan, and Ira M Hall. Lumpy: a probabilistic framework for structural variant discovery. *Genome biology*, 15(6):R84, 2014.

[36] Tobias Rausch, Thomas Zichner, Andreas Schlattl, Adrian M Stütz, Vladimir Benes, and Jan O Korbel. Delly: structural variant discovery by integrated paired-end and split-read analysis. *Bioinformatics*, 28(18):i333–i339, 2012.

[37] Kai Ye, Marcel H Schulz, Quan Long, Rolf Apweiler, and Zemin Ning. Pindel: a pattern growth approach to detect break points of large deletions and medium sized insertions from paired-end short reads. *Bioinformatics*, 25(21):2865–2871, 2009.

[38] Karen H Miga, Sergey Koren, Arang Rhie, Mitchell R Vollger, Ariel Gershman, Andrey Bzikadze, Shelise Brooks, Edmund Howe, David Porubsky, Glennis A Logsdon, et al. Telomere-to-telomere assembly of a complete human x chromosome. *Nature*, 585(7823): 79–84, 2020.

[39] Andrey V Bzikadze and Pavel A Pevzner. Automated assembly of centromeres from ultra-long error-prone reads. *Nature Biotechnology*, pages 1–8, 2020.

[40] Robert E Handsaker, Joshua M Korn, James Nemesh, and Steven A McCarroll. Discovery and genotyping of genome structural polymorphism by sequencing on a population scale. *Nature genetics*, 43(3):269, 2011.

[41] Colby Chiang, Ryan M Layer, Gregory G Faust, Michael R Lindberg, David B Rose, Erik P Garrison, Gabor T Marth, Aaron R Quinlan, and Ira M Hall. Speedseq: ultra-fast personal genome analysis and interpretation. *Nature methods*, 12(10):966, 2015.

[42] Paul Medvedev, Monica Stanciu, and Michael Brudno. Computational methods for discovering structural variation with next-generation sequencing. *Nature methods*, 6(11):S13–S20, 2009.

[43] Jiadong Lin, Xiaofei Yang, Walter Kosters, Tun Xu, Yanyan Jia, Songbo Wang, Qihui Zhu, Mallory Ryan, Li Guo, Chengsheng Zhang, et al. Mako: a graph-based pattern growth approach to detect complex structural variants. *bioRxiv*, 2021.

[44] Pacific Biosciences of California. pbsv: Pacbio structural variant (sv) calling and analysis tools. `https://github.com/PacificBiosciences/pbsv`, 2018.

[45] Tao Jiang, Yongzhuang Liu, Yue Jiang, Junyi Li, Yan Gao, Zhe Cui, Yadong Liu, Bo Liu, and Yadong Wang. Long-read-based human genomic structural variation detection with cutesv. *Genome biology*, 21(1):1–24, 2020.

[46] Raluca Uricaru, Guillaume Rizk, Vincent Lacroix, Elsa Quillery, Olivier Plantard, Rayan Chikhi, Claire Lemaitre, and Pierre Peterlongo. Reference-free detection of isolated snps. *Nucleic acids research*, 43(2):e11–e11, 2014.

[47] Ariya Shajii, Deniz Yorukoglu, Yun William Yu, and Bonnie Berger. Fast genotyping of known snps through approximate k-mer matching. *Bioinformatics*, 32(17):i538–i544, 2016.

[48] Chen Sun and Paul Medvedev. Toward fast and accurate snp genotyping from whole genome sequencing data for bedside diagnostics. *Bioinformatics*, 35(3):415–420, 2018.

[49] Luca Denti, Marco Previtali, Giulia Bernardini, Alexander Schönhuth, and Paola Bonizzoni. Malva: genotyping by mapping-free allele detection of known variants. *iScience*, 18:20–27, 2019.

[50] Giulio Formenti, Arang Rhie, Brian P Walenz, Françoise Thibaud-Nissen, Kishwar Shafin, Sergey Koren, Eugene W Myers, Erich D Jarvis, and Adam M Phillippy. Merfin: improved variant filtering and polishing via k-mer validation. *bioRxiv*, 2021.

[51] Jonas Andreas Sibbesen, Lasse Maretty, and Anders Krogh. Accurate genotyping across variant classes and lengths using variant graphs. *Nature genetics*, 50(7):1054–1059, 2018.

[52] Guillaume Marçais and Carl Kingsford. A fast, lock-free approach for efficient parallel counting of occurrences of k-mers. *Bioinformatics*, 27(6):764–770, 2011.

[53] Michael R Crusoe, Hussien F Alameldin, Sherine Awad, Elmar Boucher, Adam Caldwell, Reed Cartwright, Amanda Charbonneau, Bede Constantinides, Greg Edvenson, Scott Fay, et al. The khmer software package: enabling efficient nucleotide sequence analysis. *F1000Research*, 4, 2015.

[54] Guillaume Rizk, Dominique Lavenier, and Rayan Chikhi. Dsk: k-mer counting with very low memory usage. *Bioinformatics*, 29(5):652–653, 2013.

[55] Sebastian Deorowicz, Agnieszka Debudaj-Grabysz, and Szymon Grabowski. Disk-based k-mer counting on a pc. *BMC bioinformatics*, 14(1):160, 2013.

[56] Sergey Nurk, Sergey Koren, Arang Rhie, Mikko Rautiainen, Andrey V Bzikadze, Alla Mikheenko, Mitchell R Vollger, Nicolas Altemose, Lev Uralsky, Ariel Gershman, et al. The complete sequence of a human genome. *bioRxiv*, 2021.

[57] Samuel Karlin, Allan M Campbell, and Jan Mrazek. Comparative DNA analysis across diverse genomes. *Annual review of genetics*, 32(1):185–225, 1998.

[58] Ben Langmead and Steven L Salzberg. Fast gapped-read alignment with bowtie 2. *Nature methods*, 9(4):357, 2012.

[59] Ryan Poplin, Valentin Ruano-Rubio, Mark A DePristo, Tim J Fennell, Mauricio O Carneiro, Geraldine A Van der Auwera, David E Kling, Laura D Gauthier, Ami Levy-Moonshine, David Roazen, et al. Scaling accurate genetic variant discovery to tens of thousands of samples. *BioRxiv*, page 201178, 2017.

[60] Cornelis A Albers, Gerton Lunter, Daniel G MacArthur, Gilean McVean, Willem H Ouwehand, and Richard Durbin. Dindel: accurate indel calls from short-read data. *Genome research*, 21(6):961–973, 2011.

[61] Melissa Gymrek, David Golan, Saharon Rosset, and Yaniv Erlich. lobstr: a short tandem repeat profiler for personal genomes. *Genome research*, 22(6):1154–1162, 2012.

[62] Mehrdad Bakhtiari, Sharona Shleizer-Burko, Melissa Gymrek, Vikas Bansal, and Vineet Bafna. Targeted genotyping of variable number tandem repeats with advntr. *Genome research*, 28(11):1709–1719, 2018.

[63] Daniel L Cameron, Leon Di Stefano, and Anthony T Papenfuss. Comprehensive evaluation and characterisation of short read general-purpose structural variant calling software. *Nature communications*, 10(1):1–11, 2019.

[64] Ibrahim Numanagić, Alim S Gökkaya, Lillian Zhang, Bonnie Berger, Can Alkan, and Faraz Hach. Fast characterization of segmental duplications in genome assemblies. *Bioinformatics*, 34(17):i706–i714, 09 2018. ISSN 1367-4803. doi: 10.1093/bioinformatics/bty586. URL `https://doi.org/10.1093/bioinformatics/bty586`.

[65] Zechen Chong, Jue Ruan, Min Gao, Wanding Zhou, Tenghui Chen, Xian Fan, Li Ding, Anna Y Lee, Paul Boutros, Junjie Chen, et al. novobreak: local assembly for breakpoint detection in cancer genomes. *Nature methods*, 14(1):65, 2017.

[66] Pierre Peterlongo, Chloe Riou, Erwan Drezen, and Claire Lemaitre. Discosnp++: de novo detection of small variants from raw unassembled read set (s). *BioRxiv*, page 209965, 2017.

[67] Jérôme Audoux, Nicolas Philippe, Rayan Chikhi, Mikaël Salson, Mélina Gallopin, Marc Gabriel, Jérémy Le Coz, Emilie Drouineau, Thérèse Commes, and Daniel Gautheret. DE-kupl: exhaustive capture of biological variation in RNA-seq data through k-mer decomposition. *Genome biology*, 18(1):243, 2017.

[68] Giuseppe Narzisi, Jason A O'rawe, Ivan Iossifov, Han Fang, Yoon-ha Lee, Zihua Wang, Yiyang Wu, Gholson J Lyon, Michael Wigler, and Michael C Schatz. Accurate de novo and

transmitted indel detection in exome-capture data using microassembly. *Nature methods*, 11(10):1033, 2014.

[69] Laura Gómez-Romero, Kim Palacios-Flores, José Reyes, Delfino García, Margareta Boege, Guillermo Dávila, Margarita Flores, Michael C Schatz, and Rafael Palacios. Precise detection of de novo single nucleotide variants in human genomes. *Proceedings of the National Academy of Sciences*, 115(21):5516–5521, 2018.

[70] Daniel S Standage, C Titus Brown, and Fereydoun Hormozdiari. Kevlar: a mapping-free framework for accurate discovery of de novo variants. *BioRxiv*, page 549154, 2019.

[71] Atif Rahman, Ingileif Hallgrímsdóttir, Michael Eisen, and Lior Pachter. Association mapping from sequencing reads using k-mers. *eLife*, 7:e32920, 2018.

[72] Juliane C Dohm, Claudio Lottaz, Tatiana Borodina, and Heinz Himmelbauer. Substantial biases in ultra-short read data sets from high-throughput dna sequencing. *Nucleic acids research*, 36(16):e105, 2008.

[73] Leonardo Dagum and Ramesh Menon. Openmp: an industry standard api for shared-memory programming. *IEEE computational science and engineering*, 5(1):46–55, 1998.

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

[75] Sai Chen, Peter Krusche, Egor Dolzhenko, Rachel M Sherman, Roman Petrovski, Felix Schlesinger, Melanie Kirsche, David R Bentley, Michael C Schatz, Fritz J Sedlazeck, et al. Paragraph: A graph-based structural variant genotyper for short-read sequence data. *Genome Biology*, 20(1):1–13, 2019.

[76] Varuna Chander, Richard A Gibbs, and Fritz J Sedlazeck. Evaluation of computational genotyping of structural variation for clinical diagnoses. *GigaScience*, 8(9):giz110, 2019.

[77] Jessica W Lau, Erik Lehnert, Anurag Sethi, Raunaq Malhotra, Gaurav Kaushik, Zeynep Onder, Nick Groves-Kirkby, Aleksandar Mihajlovic, Jack DiGiovanna, Mladen Srdic, et al. The cancer genomics cloud: collaborative, reproducible, and democratized—a new paradigm in large-scale computational research. *Cancer research*, 77(21):e3–e6, 2017.

[78] Jan Graffelman. Exploring diallelic genetic markers: The HardyWeinberg package. *Journal of Statistical Software*, 64(3):1–23, 2015. URL `https://www.jstatsoft.org/v64/i03/`.

[79] Jan Graffelman and Jair Morales-Camarena. Graphical tests for hardy-weinberg equilibrium based on the ternary plot. *Human Heredity*, 65(2):77–84, 2008.

[80] Jonathan Marchini and Bryan Howie. Genotype imputation for genome-wide association studies. *Nature Reviews Genetics*, 11(7):499–511, 2010.

[81] Chen Sun and Paul Medvedev. Toward fast and accurate snp genotyping from whole genome sequencing data for bedside diagnostics. *Bioinformatics*, 35(3):415–420, 2019.

[82] Adam M Phillippy, Jacquline A Mason, Kunmi Ayanbule, Daniel D Sommer, Elisa Taviani, Anwar Huq, Rita R Colwell, Ivor T Knight, and Steven L Salzberg. Comprehensive dna signature discovery and validation. *PLOS Computational Biology*, 3(5):1–8, 05 2007. doi: 10.1371/journal.pcbi.0030098. URL `https://doi.org/10.1371/journal.pcbi.0030098`.

[83] Aaron M Wenger, Paul Peluso, William J Rowell, Pi-Chuan Chang, Richard J Hall, Gregory T Concepcion, Jana Ebler, Arkarachai Fungtammasan, Alexey Kolesnikov, Nathan D Olson, et al. Accurate circular consensus long-read sequencing improves variant detection and assembly of a human genome. *Nature biotechnology*, 37(10):1155–1162, 2019.

[84] Szymon M Kiełbasa, Raymond Wan, Kengo Sato, Paul Horton, and Martin C Frith. Adaptive seeds tame genomic sequence comparison. *Genome research*, 21(3):487–493, 2011.

[85] Heng Li. Exploring single-sample SNP and INDEL calling with whole-genome de novo assembly. *Bioinformatics*, 28(14):1838–1844, 05 2012. doi: 10.1093/bioinformatics/bts280. URL `https://doi.org/10.1093/bioinformatics/bts280`.

[86] Djamal Belazzougui, Fabio Cunial, Juha Kärkkäinen, and Veli Mäkinen. Linear-time string indexing and analysis in small space. *ACM Transactions on Algorithms (TALG)*, 16(2): 1–54, 2020.

[87] Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 390–398. IEEE, 2000.

[88] Tak Wah Lam, Ruiqiang Li, Alan Tam, Simon Wong, Edward Wu, and Siu-Ming Yiu. High throughput short read alignment via bi-directional bwt. In *2009 IEEE International Conference on Bioinformatics and Biomedicine*, pages 31–36. IEEE, 2009.

[89] Heng Li. Fast construction of fm-index for long sequence reads. *Bioinformatics*, 30(22): 3274–3275, 2014.

[90] Yukiteru Ono, Kiyoshi Asai, and Michiaki Hamada. PBSIM: PacBio reads simulator—toward accurate genome assembly. *Bioinformatics*, 29(1):119–121, 11 2012. ISSN 1367-4803. doi: 10.1093/bioinformatics/bts649. URL `https://doi.org/10.1093/bioinformatics/bts649`.

[91] David Porubsky, Peter Ebert, Peter A Audano, Mitchell R Vollger, William T Harvey, Pierre Marijon, Jana Ebler, Katherine M Munson, Melanie Sorensen, Arvis Sulovari, et al. Fully phased human genome assembly without parental data using single-cell strand sequencing and long reads. *Nature Biotechnology*, pages 1–7, 2020.

[92] René L Warren, Lauren Coombe, Hamid Mohamadi, Jessica Zhang, Barry Jaquish, Nathalie Isabel, Steven JM Jones, Jean Bousquet, Joerg Bohlmann, and Inanç Birol. ntedit: scalable genome sequence polishing. *Bioinformatics*, 35(21):4430–4432, 2019.

[93] Marek Kokot, Maciej Długosz, and Sebastian Deorowicz. Kmc 3: counting and manipulating k-mer statistics. *Bioinformatics*, 33(17):2759–2761, 2017.

[94] Brian Bushnell. BBMap: a fast, accurate, splice-aware aligner. Technical report, Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States), 2014.

[95] Repeatmasker open-4.0. 2013–2015.

[96] Peter Ebert, Peter A Audano, Qihui Zhu, Bernardo Rodriguez-Martin, David Porubsky, Marc Jan Bonder, Arvis Sulovari, Jana Ebler, Weichen Zhou, Rebecca Serra Mari, et al. De novo assembly of 64 haplotype-resolved human genomes of diverse ancestry and integrated analysis of structural variation. *bioRxiv*, 2020.

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

[98] Heng Li. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. *arXiv preprint arXiv:1303.3997*, 2013.

[99] Rayan Chikhi, Antoine Limasset, and Paul Medvedev. Compacting de Bruijn graphs from sequencing data quickly and in low memory. *Bioinformatics*, 32(12):i201–i208, 06 2016. ISSN 1367-4803. doi: 10.1093/bioinformatics/btw279.

[100] Christopher Lee, Catherine Grasso, and Mark F Sharlow. Multiple sequence alignment using partial order graphs. *Bioinformatics*, 18(3):452–464, 2002.

[101] Parsoa Khorsand, Luca Denti, Human Genome Structural Variant Consortium, Paola Bonizzoni, Rayan Chikhi, and Fereydoun Hormozdiari. Comparative genome analysis using sample-specific string detection in accurate long reads. *Bioinformatics Advances*, 1(1): vbab005, 2021.

[102] Yan Gao, Yongzhuang Liu, Yanmei Ma, Bo Liu, Yadong Wang, and Yi Xing. abpoa: an simd-based c library for fast partial order alignment using adaptive band. *Bioinformatics*, 37(15):2209–2211, 2021.

[103] Peter Edge and Vikas Bansal. Longshot enables accurate variant calling in diploid genomes from single-molecule long read sequencing. *Nature communications*, 10(1):1–10, 2019.

[104] Jeff Daily. Parasail: Simd c library for global, semi-global, and local pairwise sequence alignments. *BMC bioinformatics*, 17(1):1–11, 2016.

[105] Heng Li, Jonathan M Bloom, Yossi Farjoun, Mark Fleharty, Laura Gauthier, Benjamin Neale, and Daniel MacArthur. A synthetic-diploid benchmark for accurate variant-calling evaluation. *Nature methods*, 15(8):595–597, 2018.

[106] Justin M Zook, Nancy F Hansen, Nathan D Olson, Lesley Chapman, James C Mullikin, Chunlin Xiao, Stephen Sherry, Sergey Koren, Adam M Phillippy, Paul C Boutros, et al. A robust benchmark for detection of germline large deletions and insertions. *Nature biotechnology*, 38(11):1347–1355, 2020.

[107] Gunjan Baid, Daniel E Cook, Kishwar Shafin, Taedong Yun, Felipe Llinares-Lopez, Quentin Berthet, Aaron M Wenger, William J Rowell, Maria Nattestad, Howard Yang, et al. Deepconsensus: Gap-aware sequence transformers for sequence correction. *bioRxiv*, 2021.

[108] Pacific Biosciences of California. pbmm2: A minimap2 smrt wrapper for pacbio data. `https://github.com/PacificBiosciences/pbmm2`, 2018.

[109] Fritz J Sedlazeck, Philipp Rescheneder, Moritz Smolka, Han Fang, Maria Nattestad, Arndt Von Haeseler, and Michael C Schatz. Accurate detection of complex structural variations using single-molecule sequencing. *Nature methods*, 15(6):461–468, 2018.

[110] David Heller and Martin Vingron. Svim: structural variant identification using mapped long reads. *Bioinformatics*, 35(17):2907–2915, 2019.

[111] Yu Chen, Amy Wang, Courtney Barkley, Xinyang Zhao, Min Gao, Micky Edmonds, and Zechen Chong. Debreak: Deciphering the exact breakpoints of structural variations using long sequencing reads. 2022.

[112] Spiral Genetics. truvari: Svbenchmarkingtool. `https://github.com/spiralgenetics/truvari`, 2018.

[113] Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18): 3094–3100, 2018.

[114] Fritz J Sedlazeck, Philipp Rescheneder, Moritz Smolka, Han Fang, Maria Nattestad, Arndt Von Haeseler, and Michael C Schatz. Accurate detection of complex structural variations using single-molecule sequencing. *Nature methods*, 15(6):461–468, 2018.

[115] Alexander Lex, Nils Gehlenborg, Hendrik Strobelt, Romain Vuillemot, and Hanspeter Pfister. Upset: visualization of intersecting sets. *IEEE transactions on visualization and computer graphics*, 20(12):1983–1992, 2014.

[116] Justin Wagner, Nathan D Olson, Lindsay Harris, Jennifer McDaniel, Haoyu Cheng, Arkarachai Fungtammasan, Yih-Chii Hwang, Richa Gupta, Aaron M Wenger, William J Rowell, et al. Towards a comprehensive variation benchmark for challenging medically-relevant autosomal genes. *bioRxiv*, 2021.

[117] David Porubsky, Wolfram Höps, Hufsah Ashraf, PingHsun Hsieh, Bernardo Rodriguez-Martin, Feyza Yilmaz, Jana Ebler, Pille Hallast, Flavia AM Maggiolini, William T Harvey, et al. Haplotype-resolved inversion landscape reveals hotspots of mutational recurrence associated with genomic disorders. *bioRxiv*, 2021.

[118] Kai Ye, Zhenyu Jia, Yipeng Wang, Paul Flicek, and Rolf Apweiler. Mining unique-m substrings from genomes. *Journal of proteomics & bioinformatics*, 3(3):099, 2010.

[119] Christopher E Pearson, Kerrie Nichol Edamura, and John D Cleary. Repeat instability: mechanisms of dynamic mutations. *Nature Reviews Genetics*, 6(10):729–742, 2005.

[120] Helge Ræder, Stefan Johansson, Pål I Holm, Ingfrid S Haldorsen, Eric Mas, Véronique Sbarra, Ingrid Nermoen, Stig Å Eide, Louise Grevle, Lise Bjørkhaug, et al. Mutations in the cel vntr cause a syndrome of diabetes and pancreatic exocrine dysfunction. *Nature genetics*, 38(1):54–62, 2006.

[121] KJ Brookes. The vntr in complex disorders: the forgotten polymorphisms? a functional way forward? *Genomics*, 101(5):273–281, 2013.

[122] Nima Mousavi, Sharona Shleizer-Burko, Richard Yanicky, and Melissa Gymrek. Profiling the genome-wide landscape of tandem repeat expansions. *Nucleic Acids Research*, 47(15): e90–e90, 06 2019. ISSN 0305-1048. doi: 10.1093/nar/gkz501. URL `https://doi.org/10.1093/nar/gkz501`.

[123] Thomas Willems, Dina Zielinski, Jie Yuan, Assaf Gordon, Melissa Gymrek, and Yaniv Erlich. Genome-wide profiling of heritable and de novo str variations. *Nature methods*, 14 (6):590–592, 2017.

[124] Ryan Poplin, Pi-Chuan Chang, David Alexander, Scott Schwartz, Thomas Colthurst, Alexander Ku, Dan Newburger, Jojo Dijamco, Nam Nguyen, Pegah T Afshar, et al. A universal snp and small-indel variant caller using deep neural networks. *Nature biotechnology*, 36(10):983–987, 2018.

[125] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.