Computational genomics – project 3

Adrian Kamiński, Jakub Fołtyn

June 2024

1 Introduction

This document contains a detailed description of all the actions taken and resources used during work on the third project in the *Computational Genomics* course. This project involved creating an algorithm capable of *structural variant calling*.
In this report, we will introduce some theoretical background, especially explaining what structural variants are and why detecting them is crucial. We will later describe the algorithm that we have chosen, outlining our implementation and what changes have been made. Next, we will provide some results of our calling, as well as provide some tests and comparisons with other variant detection methods. Finally, we will make a short summary and discuss the results.

2 Theory and methods

In this section, we will describe the structural variants as a concept and introduce the detection algorithm we used in this project. We will also provide a short description of the different data formats that were used.

2.1 Structural Variants

In simple terms, a *structural variant* is a region of the DNA that is changed in comparison to a reference ("typical") genome. Usually, for a region to be considered a structural variant, it needs to be rather significant in size (the requirements may vary, but in our project, we take into account only regions larger than 50 base pairs (bp)).
Structural variants are known to be associated with a plethora of different diseases [3], [6], especially hereditary diseases, but also cancer [5]. For this reason accurate methods for their identification in genome samples are of vital importance.
There are many different types of structural variances. Some of the most "popular" ones include:

- **deletions** – a fragment of the genome is missing in comparison to the reference.
- **duplications** – a fragment of the genome is duplicated in comparison to the reference.
- **insertions** – a new fragment of the genome is inserted into the sample in comparison to the reference
- **translocations** – a fragment of the genome is translocated in comparison to the reference.
- **inversions** – a fragment of the genome is inverted in comparison to the reference.

There is also one important "category" of structural variants. Those are the so called *Copy Number Variants* (*CNVs*). Those variations are "a molecular phenomenon in which sequences of the genome are repeated, and the number of repeats varies between individuals of the same species." [9]. Those types of variants are especially important in this project, as they are primarily targeted by the methodology that we have chosen.

2.2 Algorithm

In our project, we developed a structural variant calling algorithm that is a slight modification of a *ReadDepth* algorithm [7]. This algorithm is concentrated on detecting the *copy number variants* in particular. The idea of this algorithm is to group the reads from a given file into smaller sequences or bins. reads that fall into those bins are then counted. Bins that contain a higher-than-usual or lower-than-usual number of counts are then considered to be "prime suspects" of containing some structural variation (in the case of this algorithm, the variations detected are either deletions or insertions/duplications). This type of structural variation detection is called the read-depth method (hence the algorithm's name).
It is worth noting that binning itself is not the only part of the algorithm. Authors of [7] introduced several additional steps aimed at improving the algorithm's effectiveness. One of these steps involved multiplying the number of reads by the inverse of a given area's mappability. To simulate that, we have decided not to exclude the unmapped reads included in *.bam* files used in our project.
We grouped the whole genome into bins (one chromosome at a time) of size *bin_size*, with *bin_size* being a parameter (the default value has been set to 1,000, a value in our opinion combining a sufficiently through bin coverage with computation speed. We then counted the number of reads that fall into the respective bins. As a read "inside a bin" we consider a read that has at least half of its length covered by the bin.
After having grouped the reads into bins, we then run the *Circular binary segmentation* algorithm [8]. This algorithm's purpose is to further group the bins

into segments, identifying structural variant breakpoints (boundaries) between these segments. This is achieved by creating segments such that the difference between the partial means of these segments is maximal. This partition is further validated by performing a t-test [4]. The "circular" part of the algorithm's name stems from the fact that we initially divide the sequence into "inner" and "outer" partitions, with the outer partition's ends being treated as neighboring each other (so the data is treated as if it was a circle).
The segmentation is then run recursively until the partitions are either small enough (with the size being the algorithm's parameter) or are no longer significant (as determined by the t-test).
To further validate the partitions, we perform a bootstrapping method in which we randomly shuffle the data 100 times, again performing the t-test each time and comparing the results. This allows us to avoid any hypotheses about the structure of the data. A similar validation is also run after the algorithm ends, on all the found breakpoints, to check their validity (using a more strict p-value, usually of *0.01*, where during algorithm runs we used *0.05*.
We based our implementation of this algorithm on the instructions provided in this blog.
Finally, having obtained new segment boundaries, we consolidate the reads number into those boundaries (we sum the number of reads inside these segments and then divide it by the length of the segment to account for varying segment lengths. We then set the thresholds for detecting deletions (lower threshold) or duplications (higher threshold). It is worth noting that these thresholds may also be set during the initialization of the algorithm as parameters. If not previously set, we set them to the 5th and 95th percentile of the read number distribution, respectively. That means that all segments with the number of reads below the 5th percentile are classified as deletions and all segments with the number of reads above the 95th percentile – as duplications.
In the final step, we filter the segments to retain only the structural variants.

2.2.1 Algorithm – usage

Our algorithm is implemented as a *python* class called *ReadDepthAlgorithm*. To use it, one needs to first initialize this class, and then invoke the *run_algorithm* method.
The *run_algorithm* method takes as an input one parameter: *contig_list*. It is a list of strings, containing the names of all the contig names on which the algorithm is to be run (e.g. chr1, chr2 etc.)
The algorithm initialization parameters are as follows:

- **bin_size** – int, required, the size of the bin
- **file** – string, required, .bam file name with reads for variant detection
- **reference** – string, required, .fasta file name with reference genome sequences
- **output_filename** – string or None, the filename of the output .vcf file

- **average** – boolean, whether to average the number of reads across segments
- **dup_threshold** – float or None, the thershold for duplication
- **del_threshold** – float or None, the thershold for deletions
- **check_validity** – bool, whether to check the validity of the reads (if True, unmapped and duplicated reads will not be considered for the algorithm)
- **cbs** – CBS or None, cbs algorithm class
- **num_cpus** – int or None, the number of cpus to use during parallelization of CBS algorithm, if None uses all available

2.3 File formats

In this project, we used two rather unusual file formats – the **BAM format** and the **VCF format**. Understanding their structure is of vital importance for understanding the whole of our project. Therefore, we will explain each of these formats in a dedicated subsection.

2.3.1 BAM file format

Files in the *.bam* format contain reads aligned to a reference genome. There are also additional information stored in a header (lines beginning with the @ symbol). Each row in this file contains one read in a tab-delimited format. The *.bam* format is, in essence, a compressed, binarized version of *.sam* format, which contains the same information but saved in a human-readable text form. Example of a *.bam* file can be seen on figure 1

```
@HD VN:1.6 SO:coordinate
@SQ SN:ref LN:45
r001   99 ref  7 30 8M2I4M1D3M = 37  39 TTAGATAAAGGATACTG *
r002    0 ref  9 30 3S6M1P1I4M *  0   0 AAAAGATAAGGATA    *
r003    0 ref  9 30 5S6M       *  0   0 GCCTAAGCTAA       * SA:Z:ref,29,-,6H5M,17,0;
r004    0 ref 16 30 6M14N5M    *  0   0 ATAGCTTCAGC       *
r003 2064 ref 29 17 6H5M       *  0   0 TAGGC             * SA:Z:ref,9,+,5S6M,30,1;
r001  147 ref 37 30 9M         =  7 -39 CAGCGGCAT         * NM:i:1
```

Figure 1: Example reads from a .bam/.sam file

Information about a singular read in a *.bam* file is divided into columns, as seen in figure 1. Some of the most important columns are (in order, from the left):

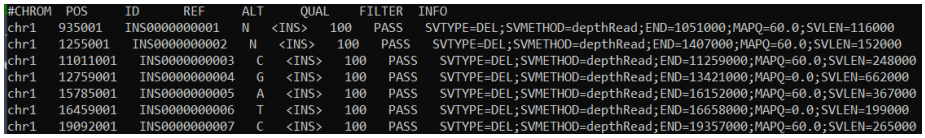
- **QNAME** – query template name.
- **FLAG** – bit-wise flag.

- **RNAME** – reference sequence name.
- **POS** – 1-based leftmost mapping position
- **MAPQ** – mapping quality.
- **CIG** – CIGAR string (specification).
- **RNEXT** – reference name of the mate/next read.
- **PNEXT** – position of the mate/next read.
- **TLEN** – observed template lengthm.
- **SEQ** – segment sequence.

For reading *.bam* files, we used *pysam*, a *python* wrapper for the *SamTools* [2] software.

2.3.2 VCF file format

Files in the *.vcf* format are mainly used for storing detected structural variants. Each row of this file contains a single variant in a tab-delimitted format. There are also header lines (starting with ##) containing additional information. An example of a *.vcf* file can be seen in figure 2. The most important columns of



```
#CHROM  POS     ID      REF     ALT     QUAL    FILTER  INFO
chr1    935001  INS0000000001   N   <INS>   100   PASS   SVTYPE=DEL;SVMETHOD=depthRead;END=1051000;MAPQ=60.0;SVLEN=116000
chr1    1255001   INS0000000002   N   <INS>   100   PASS   SVTYPE=DEL;SVMETHOD=depthRead;END=1407000;MAPQ=60.0;SVLEN=152000
chr1    11011001  INS0000000003   C   <INS>   100   PASS   SVTYPE=DEL;SVMETHOD=depthRead;END=11259000;MAPQ=60.0;SVLEN=248000
chr1    12759001  INS0000000004   G   <INS>   100   PASS   SVTYPE=DEL;SVMETHOD=depthRead;END=13421000;MAPQ=0.0;SVLEN=662000
chr1    15785001  INS0000000005   A   <INS>   100   PASS   SVTYPE=DEL;SVMETHOD=depthRead;END=16152000;MAPQ=60.0;SVLEN=367000
chr1    16459001  INS0000000006   T   <INS>   100   PASS   SVTYPE=DEL;SVMETHOD=depthRead;END=16658000;MAPQ=0.0;SVLEN=199000
chr1    19092001  INS0000000007   C   <INS>   100   PASS   SVTYPE=DEL;SVMETHOD=depthRead;END=19357000;MAPQ=60.0;SVLEN=265000
```

Figure 2: Example reads from a .vcf file

a *.vcf* file include:

- **CHROM** – chromosome identifier.
- **POS** – 1-based starting position of the variant.
- **ID** – variant id.
- **REF** – reference base at the starting position. It may be longer than 1 base.
- **ALT** – a list of alternate non-reference alleles called on at least one of the samples.
- **QUAL** – variant quality score.
- **FILTER** – information whether the variant has passed filtering (PASS).
- **INFO** – additional information about the variant.

We have developed our own method of writing a *.vcf* file using *python*. We also include some additional information in our files, such as variant segment length, median mapping quality, segment end, and detection method name.

3 Tests and results

All the tests and results from this project have been conducted on a *.bam* file: *SRR_final_sorted.bam*, along with its respective reference *.fasta* file: *GRCh38_full_analysis_set_plus_decoy_hla.fa*.

3.1 Experiments and results

The preliminary output of our algorithm is the *.vcf* file (format explained in the previous section) containing all the identified structural variants. It is, however, possible to extract a dict of dicts containing all the individual variants identified, grouped into their chromosomes of origin. Example output of our algorithm can be viewed in figure 6. It contains the following information (for each chromosome in a reference genome file):

- **Contig len** – the original contig sequence length. Used for determining the number of bins.
- **Read numbers** – number of reads in each segment. This is the total number of reads inside the segment divided by the segment length, as to account for the varying lengths.
- **Read qualities** – The median read quality of reads from the segment.
- **Bin boundaries** – the segment boundaries in the sequence (start and end).
- **References** – first base references of each segment.
- **Variants** – the segment variant name, either "deletion" or "duplication".

By default, we do not filter unmapped or duplicated reads. With this setting, we have run our algorithm on the *SRR_final_sorted.bam*. As a result, it identified 330 duplications and 2136 deletions (it is worth noting that the large numbers of deletions stem from the significant number of shorter contigs with little to no reads. Excluding them, the numbers are: 131 duplications and 20 deletions). In general, due to the usage of the CBS algorithm, we detect very long structural variant segments (121168.20 bases on average).

We also run the algorithm with valid reads filtering. This time, we detected 337 duplications and 2139 deletions, as we can see, the number of detected variants increased slightly, but not in a really meaningful way.

The other experiment we tried was to change the bin size used during the binning process. The default value was set to 1000 (this is the value used in the previous experiments). We first changed it to a 500. Our algorithm then detected 417 duplications and 2184 deletions (51 deletions and 201 duplications in the "smaller" version), and the average segment length was 73579.27 bases. As expected, the algorithm detected more variants with lower average segment length due to a more "thorough" search through the sequences.

Next, we changed the bin size to 2000. We then detected 1102 deletions and 232 duplications (16 deletions and 108 duplications). The average variant length was 131291.35 bases. As we can see, the segment length did not expand that rapidly, which may indicate that the CBS algorithm is indeed searching for proper breakpoints. However, the number of variants (especially deletions) has fallen drastically. We believe that such high bin sizes are detrimental to the algorithm's performance, as too many reads are being grouped together during binning, preventing us from properly detecting deletion areas.

We also tried a version of the algorithm in which we did not divide the number of reads in a segment by segment length. The number of detected duplications here was 2110, and the number of deletions was 346. The difference with our baseline (our first experiment) was not that significant until we checked the "smaller" version (with only 23 chromosomes), which yielded 9 deletions and 344 duplications. That may be because the majority of reads are concentrated in those first 23 chromosomes, which is why our algorithm detects their numbers as an anomaly.

For that reason, we decided to run the algorithm on default settings (no only valid reads, 1000 bin size) on only the first 23 contigs from the file (as the rest are suspiciously short in length and have very few reads). The results were 347 deletions and 165 duplications (with an average segment length of 1092488.28 bases).
Finally, we run the same experiment as before, but without averaging the number of reads. Now, the results were 166 deletions and 165 duplications, with the average segment length of 1478447.13 bases.

It is also worth noting that the algorithm, to a certain degree, relies on randomness, which is why we have set up an appropriate seed value before conducting all the experiments described above experiments.

3.2 Tests

We conducted tests of all the different individual algorithm steps to ensure that they work as intended and to examine how the data changes between them. The first step involved binning the data and counting the number of reads that fall into the bins. The result is visible in figure 3. As we can see, data at this stage consists of the contig (e.g. chromosome) length, number of reads in each bin, the read qualities from the bins (as we can see the 2nd bin has 2 reads, and as

such, 2 quality scores), bin boundaries and base references at the start of each bin.

Data after the 2nd step (segmentation) is presented in figure 4. As we can

```
Chromosome 1 after binning:
Contig len: 248956422
Read numbers: [0, 2, 0, 0, 0]
Read qualities: [[], [25, 25], [], [], []]
Bin boundaries: [(50000, 51000), (51000, 52000), (52000, 53000), (53000, 54000), (54000, 55000)]
References: ['A', 'T', 'A', 'T', 'A']
```

Figure 3: Data extract after the first algorithm step (binning)

see, there is an additional information now: segment boundaries (breakpoints), presented in bins (e.g.value 3707 means that the breakpoint is located at the 3707th bin).

```
Chromosome 1 after segmentation:
Contig len: 248956422
Read numbers: [0, 2, 0, 0, 0]
Read qualities: [[], [25, 25], [], [], []]
Bin boundaries: [(50000, 51000), (51000, 52000), (52000, 53000), (53000, 54000), (54000, 55000)]
References: ['A', 'T', 'A', 'T', 'A']
Segment boundaries: [3707, 3894, 4791, 6654, 7639]
```

Figure 4: Data extract after the second algorithm step (segmentation)

Data after the 3rd step (bin consolidation) can be viewed in figure 5. As we can see, the data fields themselved remained unchanged. The values, however, have been consolidated according to the segment boundaries: the reads have been summed inside the segments and then divided by the segment lengths as to achieve the average number of reads per segment. Read qualities now use median values, and bin boundaries have also been updated (to represent the segment boundaries).

```
Chromosome 1 after bin consolidation:
Contig len: 248956422
Read numbers: [5.9, 0.5, 2.1, 0.5, 5.2]
Read qualities: [60.0, 60.0, 60.0, 60.0, 60.0]
Bin boundaries: [(3707000, 3894000), (3894000, 4791000), (4791000, 6654000), (6654000, 7639000), (7639000, 7853000)]
References: ['T', 'A', 'C', 'A', 'A']
Segment boundaries: [3707, 3894, 4791, 6654, 7639]
```

Figure 5: Data extract after the third algorithm step (bin consolidation)

Finally, the last two steps (setting thresholds and bin filtration) have been applied, and the final data format can be found in figure 6. As we can see, the data fields are once again unchanged (with the only exception of the segmentation field, which has been replaced by variant names). The only difference is that the segments have been filtered to only include those that may be classified as "duplications" or "deletions" (according to thresholds set during the previous step).

```
Chromosome 1 after bin filtration
Contig len: 248956422
Read numbers: [15.7 19.6  8.4 10.1 10.6]
Read qualities: [ 0. 60. 60. 60. 60.]
Bin boundaries: [[16459000 16658000]
 [19092000 19200000]
 [19200000 19357000]
 [21821000 22013000]
 [31652000 31815000]]
References: ['G' 'G' 'G' 'A' 'A']
Variants: ['duplication', 'duplication', 'duplication', 'duplication', 'duplication']
```

Figure 6: Data extract after the fourth algorithm step (segment filtering)

3.3 Comparison

We compared our method to another detection algorithm called *Delly* [10]. We tested the results of each of our experiments against the output produced by the Delly algorithm. The results of the comparison are visible in figure 7. The

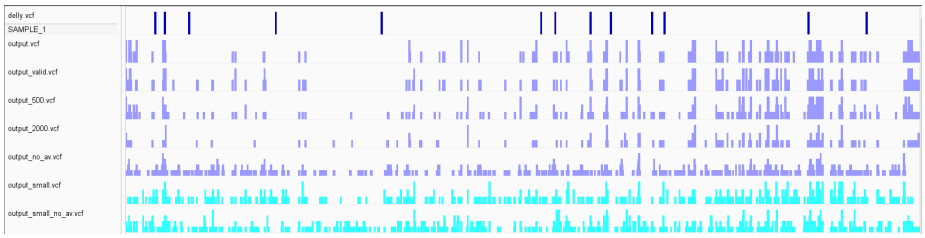


Figure 7: Comparison of the Delly algorithm with our experiment runs

track names correspond to the experiments as follows:

- **delly** – output of the delly algorithm.
- **output** – output of our baseline experiment.
- **output_valid** – output of the experiment with filtering unmapped and duplicated reads.
- **output_500** – output of the experiment with bin size 500.
- **output_2000** – output of the experiment with bin size 2000.
- **output_no_av** – output of the experiment with no averaging over segments.
- **output_small** – output of the experiment with fitting the algorithm to only the 23 first chromosomes.
- **output_small_no_av** – output of the experiment with fitting the algorithm to only the 23 first chromosomes without averaging over segments.

As we can see, the found variant coverage between the different configurations (experiments) of our algorithm seems to be mostly constant. The experiments with fitting the algorithm to a smaller number of chromosomes, as well as not averaging the number of reads by the segments seems to yield a lot more found variances, however – to a point that nearly the entire sequence is covered.

Unfortunately, when it comes to comparison with the Delly algorithm, we can see some clear differences. The Delly algorithm detected far fewer variances (only 13, in fact). Moreover, those variances are usually much shorter in size (around 50-100 bases). However, as we investigated the results of Delly algorithm further, it soon became apparent that even those few detected variants were considered "imprecise", as seen in figure 8.

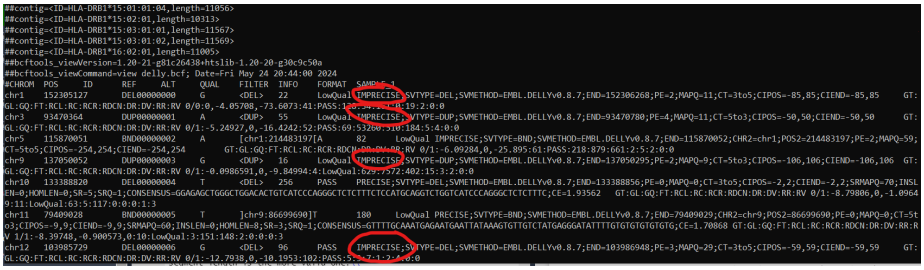


Figure 8: Example variants found by the Delly algorithm

As the results from Delly algorithm were not satisfactory, we decided to once again compare our experiments with another algorithm – *BreakDancer* [1]. The results of alignment can be viewed in figure 9, and with a zoom in a chosen region: 10.

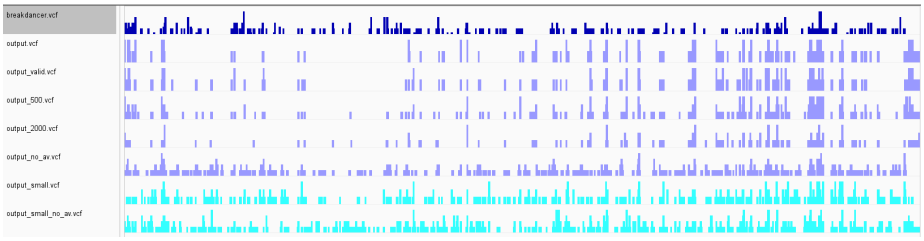


Figure 9: Comparison of the BreakDancer algorithm with our experiment runs

As we can see, now the variants are much more aligned with our results. This statement remains true even after zooming in into a chosen region. However, again, our algorithm tends to find much wider structural variant segments. What is also interesting is that here the experiments without using averaging seem to be better aligned with the results from the algorithm being compared.

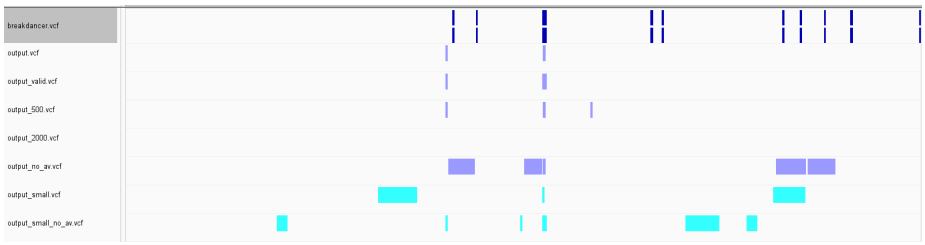


Figure 10: Comparison of the BreakDancer algorithm with our experiment runs (zoomed in)

4 Summary and discussion

In summary, we believe that we succeeded in creating an algorithm capable of detecting copy number variations, given a file with aligned reads. This statement, however, is based mainly of the comparison results with some more well-established algorithms, such as *BreakDancer*. Unfortunately, we lack ground-truth data that would enable us to state with complete certainty whether or not our algorithm is capable of detecting proper variants in the genome's structure.

We also found that our algorithm tends to detect much wider regions than other algorithms, which may mean that it tends to be less precise. Finally, after comparing our experiment results to other algorithms, we have decided that the validity of the approach of averaging read numbers across regions is inconclusive. As such, we have left this matter for the user to decide by adding an additional parameter ("average") to the algorithm's initialization. This parameter dictates whether this function should be switched on or not.

References

[1] Ken Chen, John W Wallis, Michael D McLellan, David E Larson, Joelle M Kalicki, Craig S Pohl, Sean D McGrath, Michael C Wendl, Qunyuan Zhang, Devin P Locke, et al. Breakdancer: an algorithm for high-resolution mapping of genomic structural variation. *Nature methods*, 6(9):677–681, 2009.

[2] Petr Danecek, James K Bonfield, Jennifer Liddle, John Marshall, Valeriu Ohan, Martin O Pollard, Andrew Whitwham, Thomas Keane, Shane A McCarthy, Robert M Davies, et al. Twelve years of samtools and bcftools. *Gigascience*, 10(2):giab008, 2021.

[3] Jennifer L Freeman, George H Perry, Lars Feuk, Richard Redon, Steven A McCarroll, David M Altshuler, Hiroyuki Aburatani, Keith W Jones, Chris Tyler-Smith, Matthew E Hurles, et al. Copy number variation: new insights in genome diversity. *Genome research*, 16(8):949–961, 2006.

[4] Tae Kyun Kim. T test as a parametric statistic. *Korean journal of anesthesiology*, 68(6):540, 2015.

[5] Le Li, Chenyang Hong, Jie Xu, Claire Yik-Lok Chung, Alden King-Yung Leung, Delbert Almerick T Boncan, Lixin Cheng, Kwok-Wai Lo, Paul BS Lai, John Wong, et al. Accurate identification of structural variations from cancer samples. *Briefings in Bioinformatics*, 25(1):bbad520, 2024.

[6] Steven A McCarroll and David M Altshuler. Copy-number variation and association studies of human disease. *Nature genetics*, 39(Suppl 7):S37–S42, 2007.

[7] Christopher A Miller, Oliver Hampton, Cristian Coarfa, and Aleksandar Milosavljevic. Readdepth: a parallel r package for detecting copy number alterations from short sequencing reads. *PloS one*, 6(1):e16327, 2011.

[8] Adam B Olshen, E Seshan Venkatraman, Robert Lucito, and Michael Wigler. Circular binary segmentation for the analysis of array-based dna copy number data. *Biostatistics*, 5(4):557–572, 2004.

[9] Ondrej Pös, Jan Radvanszky, Gergely Buglyó, Zuzana Pös, Diana Rusnakova, Bálint Nagy, and Tomas Szemes. Dna copy number variation: Main characteristics, evolutionary significance, and pathological aspects. *biomedical journal*, 44(5):548–559, 2021.

[10] 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.