Algorithms in Molecular Biology Course
Final Project Report

Konstantinos Giatras
Alexandros Kouris

MSc DSIT 2022-2023

1 Introduction

The study we chose for our project[13] addresses a significant challenge in the field of genomics: the precise identification of complex insertions and deletions (indels), integral genetic variations, within next-generation sequencing (NGS) data[15]. These indels play a vital role in many diseases, including cancer, making their accurate detection crucial. Yet, existing variant callers often fail to detect them accurately due to their inherent complexity[16].

The crux of this research lies in the development of a sophisticated algorithm designed to identify complex indels accurately within NGS data. Several meticulous tests were devised to measure the algorithm's performance, including an in-depth comparison with other variant callers in the field.

The paper's pivotal contribution is the unveiling of a novel open-source tool named INDELseek. This tool stands out in its ability to detect complex indels within NGS data. Unlike conventional methods which examine each reference position across multiple alignments, INDELseek adopts a holistic approach, studying each NGS read alignment in its entirety. Its superior detection capability lies in its proficiency to identify complex indels that manifest as a cluster of multiple mismatches, insertions, and/or deletions within a concentrated window of reference/read positions. The robustness of INDELseek was validated through a series of rigorous experiments, including the detection of complex indels within clinical samples, comparative testing with other variant callers, identification of complex indels in simulated data, and finally, locating complex indels in actual cancer samples. The findings conclusively indicated that INDELseek surpassed other variant callers in identifying complex indels, thereby making it an invaluable asset for genomics studies. In essence, this paper's contribution is the creation of a groundbreaking tool, INDELseek, that facilitates researchers in detecting complex indels accurately within NGS data. This, in turn, paves the way for a deeper understanding of the genetic underpinnings of diseases like cancer, potentially transforming disease diagnosis and treatment.

2 Background and Related Work

The work presented in this paper is related to the field of genomics, which is the study of the complete set of DNA within an organism, including all of its genes. Next-generation sequencing (NGS) is a powerful tool used in genomics research to sequence large amounts of DNA quickly and accurately. However, NGS data can be complex and difficult to analyze, especially when it comes to detecting genetic variations like insertions and deletions (indels).

Indels are genetic variations that involve the insertion or deletion of one or more nucleotides in a DNA sequence. They can be caused by a variety of factors, including errors during DNA replication, exposure to mutagens, and genetic recombination. Indels can have a significant impact on gene function and can cause or contribute to many diseases, including cancer[15, 16].

Variant calling is the process of identifying genetic variations in NGS data. It involves comparing the sequence reads obtained from the NGS data to a reference genome and identifying differences between the two. However, variant calling can be challenging, especially when it comes to detecting complex indels. Complex indels are indels that involve multiple nucleotides and can be difficult to distinguish from sequencing errors or other types of genetic variation[14].

Several variant callers are available for detecting indels in NGS data, including GATK[18] and SAMtools[17]. However, these variant callers have limitations when it comes to detecting complex indels. For example, they may miss some complex indels or report false positives.

The work presented in this paper addresses the challenge of accurately detecting complex indels in NGS data. The authors developed this new variant caller called INDELseek, which is designed to detect complex indels in NGS data of random fragments and PCR amplicons. INDELseek examines each NGS read alignment as a whole, instead of looking at each reference position across multiple alignments. This approach allows INDELseek to accurately detect complex indels that are aligned as multiple mismatches, insertions, and/or deletions clustered within a short window of reference/read positions. The authors evaluated the performance of INDELseek using several experiments and showed that it outperformed other variant callers in detecting complex indels.

3 Method

3.1 The INDELseek Algorithm

The INDELseek algorithm is implemented as a single Perl script[10, 13] and requires SAMtools version 1.3 or higher to work. It directly reads SAM/BAM alignments, compares them to a reference genome and returns complex indel calls in standard VCF format, making it easy to incorporate into standard bioinformatics workflows. After refining CIGAR (Compact Idiosyncratic Gapped Alignment Report) matches and mismatches in the supplied alignments ('=' for match, 'X' for mismatch), it considers a window as a complex indel call if it fulfills specific criteria:

1. containing at least two CIGAR operations (X, I, and/or D) no more than a specified length of l nucleotides apart.

2. a length of at least two nucleotides.

It then marks and/or removes false positives based on configurable filters for improved specificity (a schematic representation of the algorithm is shown in Figure 1).

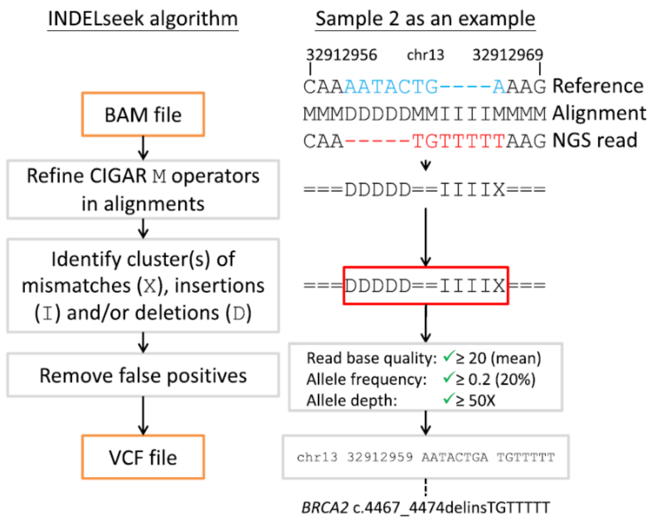

INDELseek algorithm
Sample 2 as an example
32912956 chr13 32912969
CAAAATACTG----AAAG Reference
MMMDDDDDMMIIIIMMMM Alignment
CAA-----TGTTTTTTAAG NGS read
===DDDDD==IIIIX===
===DDDDD==IIIIX===
BAM file
Refine CIGAR M operators in alignments
Identify cluster(s) of mismatches (X), insertions (I) and/or deletions (D)
Remove false positives
VCF file
Read base quality: ✓≥ 20 (mean)
Allele frequency: ✓≥ 0.2 (20%)
Allele depth: ✓≥ 50X
chr13 32912959 AATACTGA TGTTTTT
BRCA2 c.4467_4474delinsTGTTTTT

Figure 1: The INDELseek algorithm[13].

Below is a more detailed breakdown of the Perl script[10]:

- The script uses several Perl modules such as 'Getopt::Long' and 'Data::Dumper' to provide functionality like command-line options handling and data structure printing for debugging respectively. The script defines several variables for the analysis, such as reference sequence, SAMtools, flags for debugging and raw output, quality thresholds, minimum depth, and others.

- It uses 'GetOptions' from 'Getopt::Long' to specify command-line options for the script. Some of the options include 'refseq' (reference sequence file), 'SAMtools' (SAMtools command), 'phredoffset' (Phred score offset), 'min_depth' (minimum depth), and 'min_af' (minimum allele frequency). There are also flags to skip low quality, low depth, or low allele frequency.

- The next section starts with an empty hash named '%detected_mutations' and then processes the input data line by line. It uses a while loop to read in the input data, skipping empty lines or lines beginning with "@". It splits each line into fields, checks if they contain asterisks, and if they do, skips to the next line.

- The 'parse_cigar function decomposes the CIGAR string into individual operations, which is used for reconstructing the alignment.

- The 'reconstruct_alignment' function takes a CIGAR string and associated sequencing data, and reconstructs the alignment of the sequence data to a reference. This includes both the sequence of the reference and the quality scores for each base in the sequence. It returns a list of potential mutations detected during this reconstruction.

- Once the alignment is reconstructed, it categorizes the detected mutations into "simple deletion", "simple insertion", or "complex indel" based on the lengths of the reference and variant. These mutations are then stored in the '%detected_mutations' hash for further analysis.

- If the debug flag is set, the '%detected_mutations' hash is printed for debugging purposes.

- If the 'rawoutput' flag is not set, the script generates a VCF file (standard format in bioinformatics for storing gene sequence variations). The script prints out the header lines for the VCF file, then for each mutation detected, it calculates various statistics, checks quality thresholds, and adds information to the VCF fields. The resulting line is then printed to the VCF file.

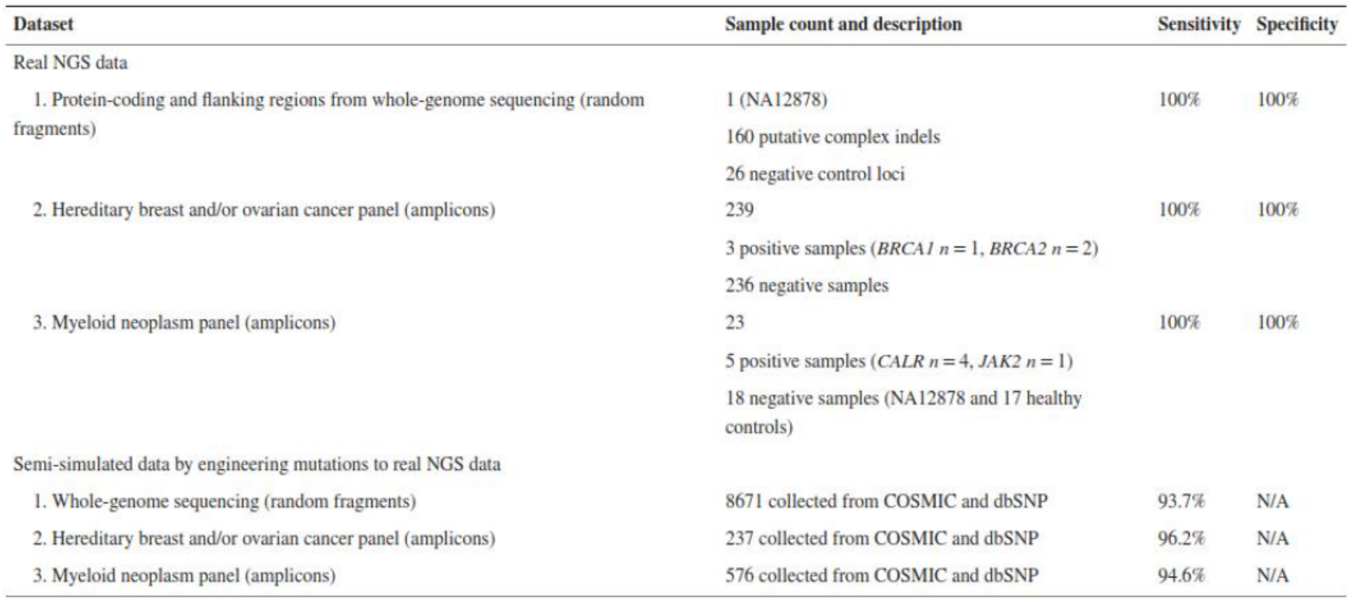
3.2 Authors' Evaluation

The authors evaluated the performance of INDELseek using Real NGS datasets from HapMap/1000 Genomes Project[1] (Case 1), where high confidence list of variants is known from the Genome In A Bottle (GIAB) Consortium[6], as well as two different NGS datasets of PCR amplicons (Cases 2 and 3), containing samples from cancer patients that were provided from local hospitals (Figure 2).

In Case 1, INDELseek was applied to reference sample NA12878 (Illumina HiSeq 2000)[9]. NA12878 is a human lymphoblastoid cell line that has been widely used as a reference sample for benchmarking NGS variant calling algorithms. The cell line was derived from a female donor of Northern and Western European ancestry and has been extensively characterized for its genomic and transcriptomic features.

The corresponding high-confidence variant calls were taken from the Genome in a Bottle (GIAB) Consortium. Although the high-confidence variant calls did not comprise complex indels as individual calls, the authors observed clusters of closely spaced variants calls that appeared in cis in the alignments of individual sequencing reads. Accordingly, 160 such loci from GIAB calls were manually curated as putative complex indels. So the final list of the expected complex indels to be detected in the sample was manually curated by the authors.

In Case 2, INDELseek was applied to a hereditary breast and/or ovarian cancer (HBOC) panel dataset of 239 probands. Prior Sanger sequencing revealed that three of the probands carried a unique pathogenic complex indel, while remaining 236 probands were negative for complex indel. INDELseek detected all three complex indels, demonstrating 100% sensitivity and 100% specificity.



Dataset	Sample count and description	Sensitivity	Specificity
Real NGS data			
1. Protein-coding and flanking regions from whole-genome sequencing (random fragments)	1 (NA12878)	100%	100%
	160 putative complex indels		
	26 negative control loci		
2. Hereditary breast and/or ovarian cancer panel (amplicons)	239	100%	100%
	3 positive samples (*BRCA1* $n = 1$, *BRCA2* $n = 2$)		
	236 negative samples		
3. Myeloid neoplasm panel (amplicons)	23	100%	100%
	5 positive samples (*CALR* $n = 4$, *JAK2* $n = 1$)		
	18 negative samples (NA12878 and 17 healthy controls)		
Semi-simulated data by engineering mutations to real NGS data			
1. Whole-genome sequencing (random fragments)	8671 collected from COSMIC and dbSNP	93.7%	N/A
2. Hereditary breast and/or ovarian cancer panel (amplicons)	237 collected from COSMIC and dbSNP	96.2%	N/A
3. Myeloid neoplasm panel (amplicons)	576 collected from COSMIC and dbSNP	94.6%	N/A

Figure 2: Datasets the authors used in the paper[13].

In Case 3, INDELseek was applied to a myeloid neoplasm (MN) panel dataset of 23 samples. From five samples known to carry a unique complex indel, INDELseek detected all five complex indels.

Using the above 3 datasets, the authors injected them with known mutations taken from the COSMIC[2] and dbSNP[3] genetic variation databases, to check if the algorithm will call them, resulting in sensitivities 93.7% (Case 1), 96.2% (Case 2) and 94.6% (Case 3). Specificity is not measured in these cases, as there are no control samples.

The authors also compared the performance of INDELseek with other variant callers, including GATK and SAMtools, in detecting complex indels. The results showed that INDELseek outperformed other variant callers in detecting complex indels, especially in the case of small complex indels.

Overall, the results of the authors' evaluation demonstrate that INDELseek is a highly sensitive and specific tool for detecting complex indels in NGS data of random fragments and PCR amplicons[13].

4 Our Evaluation

Our approach for this project was to:

- Reproduce the results of the paper to validate the authors' claims regarding the called variants.

- Improve the algorithm on various aspects, mainly focusing on performance metrics (which were absent in the paper) and create Python and C++ implementations of the algorithm to measure and compare performance.

- Compare with another variant discovery tool, namely GATK Haplotyper.

In order to test our implementations in a cohesive environment and not test the various versions of the algorithm on different machines, we created a virtual machine (VM) in Hypatia[8], with the following specifications:

- 7 CPU Cores @ 2.6GHz.

- 60GB RAM.

- 300GB of hard disk space.

4.1 Datasets

We downloaded the sample named NA12878, which is a 113GB BAM file, containing approximately 1.5 billion reads, from the Illumina Basespace[9]. This is the same sample used in Case 1 of the paper experiments. The BAM is aligned to hg38 reference genome. In order for the algorithm to run, the fasta file of the reference genome needs to be present in order to get the reference sequences for alignment regions for comparison. We downloaded the hg38 reference genome from Broad's Institute public Google Cloud Storage[7].

The cancer datasets for cases 2 and 3 could not be provided to us from the authors due to privacy reasons.

For the semi-simulated datasets, the list of complex indels were also not provided from the authors, so we had no means of recreating them.

We additionally downloaded a sample BAM file called HG00148.chr11, containing approximately 4 million reads, from the 1000 Genomes Project[1], for testing purposes.

The only available dataset that we could acquire to replicate the authors' results was the NA12878 dataset. The first run we tried on the Hypatia server was done using the original implementation in Perl, in order to get some basic metrics on read speed. After two days we stopped the run, as it seemed that the job was not gonna finish in a reasonable amount of time. The second step was to extract chr20 and chr15 from the dataset, and try again, but we stumbled upon the same case of the algorithm running too slow. On our third attempt to get some results, we used the HG00148.chr11 bam file. This was run on a local machine of ours, finishing after 5 hours and 49 minutes. So we decided for, at least, testing purposes to use the HG00148 dataset and compare to the Perl output produced by the latest run, on the same local machine.

4.2 INDELseek Implementations and Performance

In order to measure the performance of the INDELseek algorithm, we had to augment the Perl script. We needed to add some metrics, which were also implemented in the other two implementations:

- Progress indicators.
- Measure the execution time.
- Periodically measure the RAM usage (every one minute) so we can create a RAM usage plot.

Our Python implementation uses dictionaries for caching purposes. In Python, the Dictionary data types represent the implementation of hash tables. Hash tables are a type of data structure in which the address or the index value of the data element is generated from a hash function. That makes accessing the data faster as the index value behaves as a key for the data value. All in all, this implementation resulted in approximately 400 lines of code, which is comparable to the Perl script, which makes sense, as they are both high-level scripting languages.

Our C++ Implementation uses hash maps for caching purposes. It also uses hash maps instead of arrays in some of the cases of the original implementation in order to speed up lookup speed and prevent later sorting as maps are by default sorted in C++. Hash maps are a data structure that implements a dictionary (same as in Python). Basically a key pair data structure, so when you have a query for a specific key the value is returned very quickly. It has complexity of O(logN), where N is the number of entries, so it's very good for associating a key to a specific value as it's easy to use and has fast response time. Underlying algorithm implementation is based on the platform, compiler etc, but usually is a Red-Black tree which is a self-balancing binary search tree. Hash maps are highlighted with comments in the code. In addition, it uses custom structs instead of strings wherever possible, to decrease memory footprint and readability of the code. Structs are a way to represent objects in C++ similar to class. The difference between a struct and a class is that a class has by default all of its members as private whereas in a struct they are public. It's a way to model entities and a blueprint for constructing objects. Structs used are also highlighted with comments in the code. All the original filtering options are available via command line arguments. The main cpp file is accompanied with a *Makefile*, a special kind of script used by the *make* linux command in order to compile the code into an executable file, for easy compiling in any machine. All in all, this implementation resulted in approximately 900 lines of code, which is significantly more compared to the Perl script.

After finishing our implementations in Python and C++, we started our runs using the HG00148.chr11 file on the Hypatia server. Running speed in all 3 cases was significantly slower compared to our local runs during development. Based on our progress indicator, what the C++ could read in 2 minutes locally, would take almost 1 hour on the server. Our initial thought on this, is that the disk is slower on the server, resulting the program to be I/O bound, but unfortunately, we were ultimately unable to determine the cause of this irregularity. So finally, our runs were performed locally, but on the same machine (12 cores @ 4.1GHz, 16GB RAM, 500GB SSD). The results of those runs can be seen in Table 1 and Figure 3:

Table 1: Execution times of the INDELseek implementations for the HG00148.chr11 sample (approximately 4 million reads)

Implementation	Execution Time
Perl	5 hours and 49 minutes
Python	5 hours and 42 minutes
C++	40 minutes

Based on our results, the C++ implementation is much more efficient compared to the Perl and Python implementations. This makes sense, since compiled languages have a clear advantage on the performance side, but the drawback is that coding complexity increases substantially. Moreover, the resource footprint of all implementations seems to be relatively small (approximately 600MB of RAM for the 4 million reads file), making it even more curious as to why the runs were slower on the Hypatia, since the scripts do not appear to be particularly memory intensive.

Since it wasn't mentioned by the authors of the paper, we do not know what kind of hardware they used during the production of their results, but on our local machine, it would take the Perl script over 5000 hours to complete a run for the original NA12878 bam file (approximately 1.5 billion reads). It is evident that this tool has not been optimised for use in a production environment, but perhaps it could find its way into research applications, where execution time is not so critical.

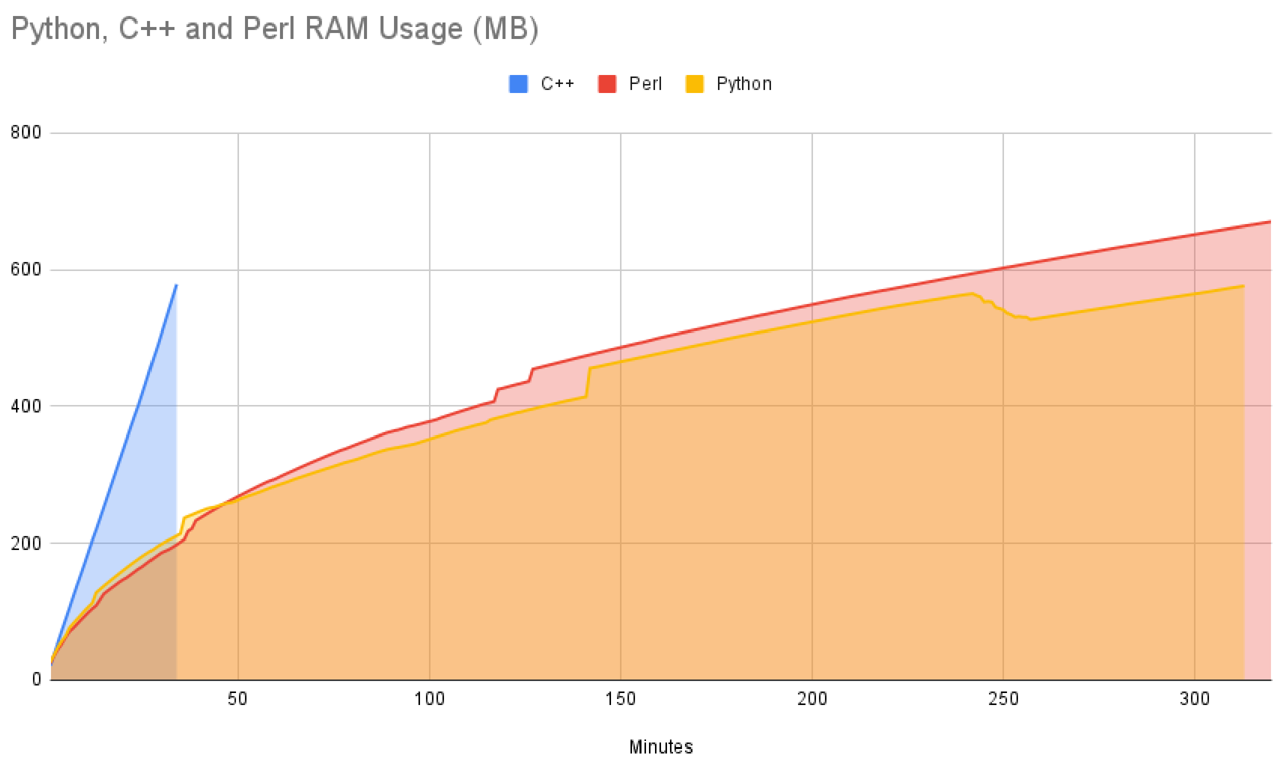


Figure 3: Python, C++ and Perl INDELseek implementations RAM usage (MB). The RSS memory (actual memory that a process is using) is the one being used for this plot

4.3 Reproducing Paper Results

Due to time limitations and the slow runtime of the Perl algorithm that we have already mentioned, we decided to extract specific regions from the original NA12878 sample file. Those regions were selected from the provided list of regions that the complex indels were detected[11].

Unfortunately, the list provided doesn't include the complex indels themselves. So our validation doesn't focus on confirming the accuracy of the algorithm, but rather on replicating the results of the algorithm. We take the output VCFs of the Perl script as the ground truth and confirm that our C++ and Python implementations output the same results.

The regions we selected can be found in Table 2.

Table 2: NA12878 Selected Regions

Entry Number	Region
1.	chr1:889158-889159
21.	chr10:124271589-124271595
22.	chr11:244106-244115
83.	chr2:3653842-3653844
120.	chr5:176936646-176936650

The BAM files follow the naming format: NA12878_S1.variant_NNN.bam, where NNN is the entry number mentioned above.

In Tables 3, 4, 5, 6 and 7, as well as Figure 4, we present the complex indels found in each of the samples using the Perl original implementation of the INDELSeek algorithm.

Table 3: NA12878_S1.variant_1.bam

#CHROM	POS	REF	ALT
chr1	889113	TCT	GCG
chr1	889154	TTGCT	CTGCC
chr1	889199	TGAGAT	CGAGAG

Table 4: NA12878_S1.variant_21.bam

#CHROM	POS	REF	ALT
chr10	124271585	GCCA	TCCT

Table 5: NA12878_S1.variant_22.bam

#CHROM	POS	REF	ALT
chr11	244106	AAA	GAG
chr11	244167	CCCTT	TCCTC

Table 6: NA12878_S1.variant_83.bam

#CHROM	POS	REF	ALT
chr2	3653843	AGTCTT	CGTCTC

Table 7: NA12878_S1.variant_120.bam

#CHROM	POS	REF	ALT
chr5	176936677	TTCAC	GTCAT

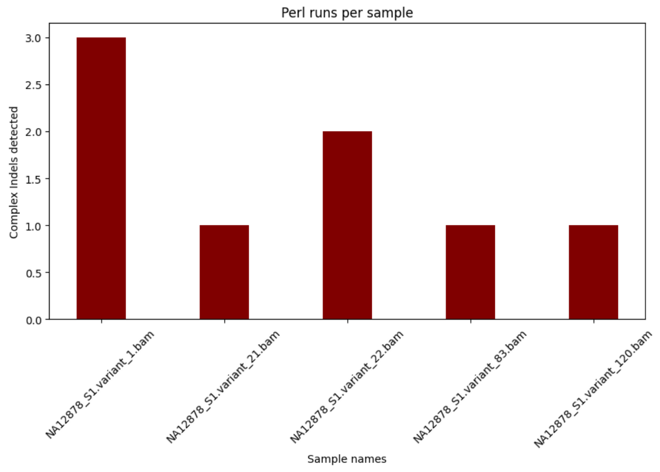


Figure 4: Complex indels detected using the Perl implementation of INDELseek

At a first glance we can confirm that all the variants detected and reported are complex indels. Also worth mentioning is the fact that, even though the BAM files are very targeted and with a low number of read alignments, the INDELSeek algorithm was able to detect complex indels. As the authors state in their original paper, INDELSeek focuses on each alignment separately and takes into account a windowed region around a variant in order to detect the indels, allowing it to perform even in low coverage regions.

Following the Perl runs, we ran our own implementations. Both Python and C++ performed exactly the same as the original algorithm and detected the same complex indels as reported above for each sample (Figures 5 and 6).

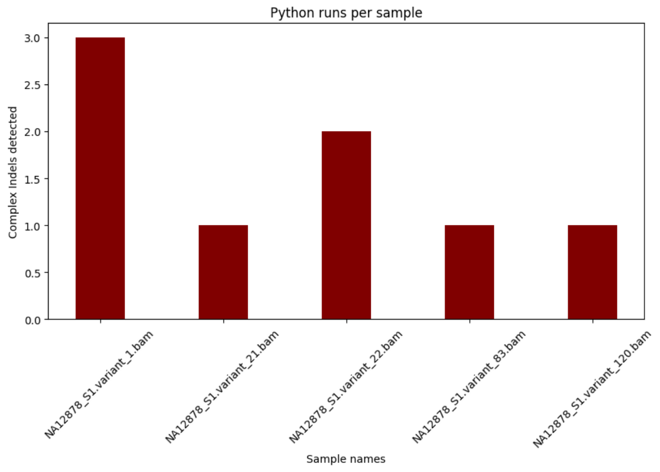


Figure 5: Complex indels detected using the Python implementation of INDELseek

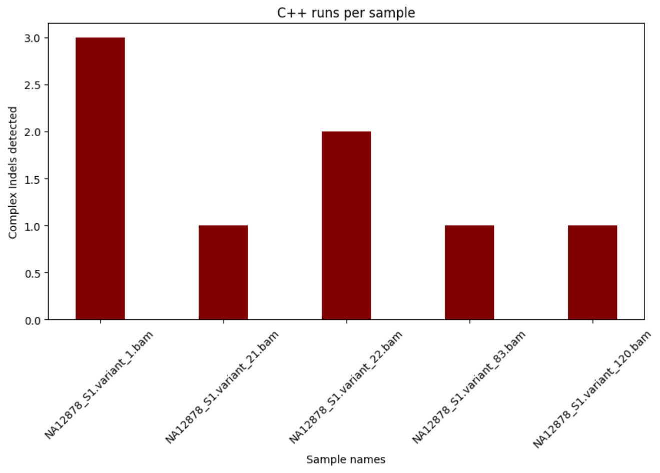


Figure 6: Complex indels detected using the C++ implementation of INDELseek

So regarding our validation process, we couldn't validate the accuracy that the authors reported due to omissions from the original paper. But we were able to confirm that the original implementation does detect complex indels and successfully replicate those results with our own implementations.

4.4 Comparison with GATK Haplotyper

In order to demonstrate the usefulness of the INDELSeek algorithm in a bioinformatics variant detection pipeline, the authors compared their results with one of most well known variant calling tools, GATK from Broad Institute[4]. Based on the original paper, GATK was not able to detect any of the complex indels reported by INDELSeek from the NA12878 sample.

Wanting to validate their claims, we decided to run GATK on the targeted BAM files found above and check the output VCFs. The variant caller used was HaplotypeCaller from GATK, version 4.4.0.0[5]. The results from every sample showed that, indeed, HaplotypeCaller was unable to detect any complex indels. Not only the complex indels reported by INDELSeek, but any kind of indel.

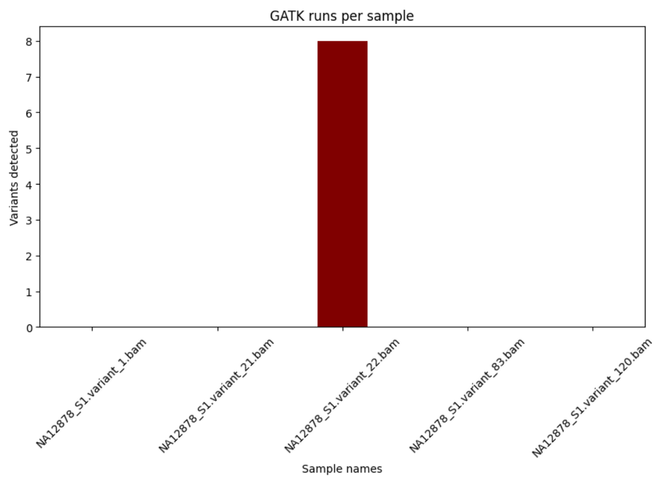


Figure 7: Complex indels detected using GATK HaplotypeCaller

Based on the above, we confirm that INDELSeek performs better than GATK HaplotypeCaller regarding complex indels detection, assuming that the complex indels are correctly detected.

5 Conclusion

This research addresses the critical challenge of accurately identifying complex insertions and deletions (indels), an essential category of genetic variations, within next-generation sequencing (NGS) data[15]. The importance of this task is emphasized by the role these indels play in numerous diseases, including cancer, underscoring the urgent need for their accurate detection[16]. Given the insufficiencies of current variant callers, which often inaccurately identify these indels due to their complexity, this study's contribution is both timely and crucial.

The research revolved around the development and testing of a novel algorithm, implemented in a tool called INDELseek, designed to address the complex task of accurately identifying indels in NGS data. INDELseek operates distinctively, examining each NGS read alignment holistically, providing it with a unique advantage in identifying complex indels over traditional methods, which inspect each reference position across multiple alignments[13].

The robustness of INDELseek was validated through extensive experimentation, spanning the detection of complex indels in clinical samples, comparative tests with other variant callers, the identification of complex indels in simulated data, and the recognition of these elements in actual cancer samples. Comparatively, INDELseek outperformed other variant callers, demonstrating its capability in detecting complex indels[13].

Our attempt to optimize the performance of INDELseek by developing Python and C++ implementations yielded interesting results. Although each implementation had a relatively small resource footprint, it was the C++ implementation that proved to be the most efficient. In contrast, the original algorithm required significant computing resources and time, taking more than 5000 hours to complete a run on a local machine for the original NA12878 BAM file.

Additionally, it is noteworthy that both Python and C++ implementations of INDELseek managed to replicate the results of the original Perl implementation, further validating the accuracy of the tool across different programming languages.

We further compared INDELseek with other variant callers and found it to outperform them in complex indel detection. This superiority was also observed when compared against the widely-used GATK HaplotypeCaller[5], which was unable to detect any complex indels in the same datasets. This underlines the necessity for specialized tools like INDELseek to analyze complex genomic alterations.

While the original algorithm's speed may limit its integration into a production variant calling pipeline, the use of a compiled language and more targeted BAM files, such as a Gene List Panel, can provide a useful and efficient solution. Focusing the analysis on specific gene regions can significantly enhance the speed of the process.

Future research could focus on optimizing the algorithm and expanding its application across more genomic datasets, maximizing its value in genetic research and clinical diagnostics. Increasing processing speed could be achieved by directly querying the reference genome or reading from the BAM file using libraries, thus removing reliance on SAMtools. Implementing Linux's 'mmap' command[12] could also enhance the reading speed of large files, leveraging its capability for "lazy loading" and efficient memory management, given that the reads are sequential and do not involve random queries.

Overall, this research underlines the utility and superiority of INDELseek in detecting complex indels in different NGS datasets. Its high sensitivity and specificity, along with its efficient use of resources and fast execution speed, make it a valuable tool for genomic studies.

6 References

[1] 1000 Genomes Project, comprehensive catalog of human genetic variation. https://www.internationalgenome.org/.

[2] COSMIC, the Catalogue Of Somatic Mutations In Cancer. https://cancer.sanger.ac.uk/cosmic.

[3] dbSNP, database of single nucleotide polymorphisms. https://www.ncbi.nlm.nih.gov/snp/.

[4] GATK (Genome Analysis Toolkit). https://gatk.broadinstitute.org/hc/en-us.

[5] GATK GitHub repository. https://github.com/broadinstitute/gatk/releases.

[6] GIAB (Genome in a Bottle). https://www.nist.gov/programs-projects/genome-bottle.

[7] Hg38 reference genome. https://console.cloud.google.com/storage/browser/genomics-public-data/resources/broad/hg38/v0.

[8] Hypatia Cloud Infrastructure. https://hypatia.athenarc.gr/index.php.

[9] Illumina Basespace. https://emea.illumina.com/platinumgenomes.html.

[10] INDELseek GitHub repository. https://github.com/tommyau/indelseek/blob/master/indelseek.pl.

[11] INDELseek Supplementary Material. https://static-content.springer.com/esm/art

[12] Mmap Linux Command. https://en.wikipedia.org/wiki/mmap.

[13] Chun Hang Au, Anskar Y. H. Leung, Ava Kwong, Tsun Leung Chan, and Edmond S. K. Ma. INDELseek: detection of complex insertions and deletions from next-generation sequencing data. *BMC Genomics*, 18(1), jan 2017.

[14] Petr Danecek, Adam Auton, Goncalo Abecasis, Cornelis A. Albers, Eric Banks, Mark A. DePristo, Robert E. Handsaker, Gerton Lunter, Gabor T. Marth, Stephen T. Sherry, Gilean McVean, and Richard Durbin and. The variant call format and VCFtools. *Bioinformatics*, 27(15):2156–2158, jun 2011.

[15] Johan T. den Dunnen and Stylianos E. Antonarakis. Mutation nomenclature extensions and suggestions to describe complex mutations: A discussion. *Human Mutation*, 15(1):7–12, jan 2000.

[16] Niall G. Howlett, Toshiyasu Taniguchi, Susan Olson, Barbara Cox, Quinten Waisfisz, Christine de Die-Smulders, Nicole Persky, Markus Grompe, Hans Joenje, Gerard Pals, Hideyuki Ikeda, Edward A. Fox, and Alan D. D'Andrea. Biallelic inactivation of iBRCA2/i in fanconi anemia. *Science*, 297(5581):606–609, jul 2002.

[17] Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor Marth, Goncalo Abecasis, and Richard Durbin and. The sequence alignment/map format and SAMtools. *Bioinformatics*, 25(16):2078–2079, jun 2009.

[18] Aaron McKenna, Matthew Hanna, Eric Banks, Andrey Sivachenko, Kristian Cibulskis, Andrew Kernytsky, Kiran Garimella, David Altshuler, Stacey Gabriel, Mark Daly, and Mark A. DePristo. The genome analysis toolkit: A MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Research*, 20(9):1297–1303, jul 2010.