WASHINGTON UNIVERSITY IN ST. LOUIS SCHOOL OF ENGINEERING AND APPLIED SCIENCE

CSE584 ALGORITHMS FOR BIOSEQUENCE COMPARISON

μInvert

John Gibson (ID 450315, WUSTL Key johngibson)

May 4, 2018

ABSTRACT

Microinversions (MIs) are small inversions in DNA. Due to difficulty in detection, microinversions and other rare small structural variants on the scale of 50-300bp are not well characterized in variant databases. Microinversions act as genetic markers that distinguish species, strains, and lineages, making their identification a useful tool in distinguishing evolutionarily-related species. In this work, we present μInvert, a fast and accurate software tool for the detection of microinversions. μInvert uses the Burrows-Wheeler transform to perform fast, inexact matching on short reads. In our test, we discovered 19 short reads containing possible microinversions across unaligned reads in 50 inbred yeast isolates.

1 INTRODUCTION

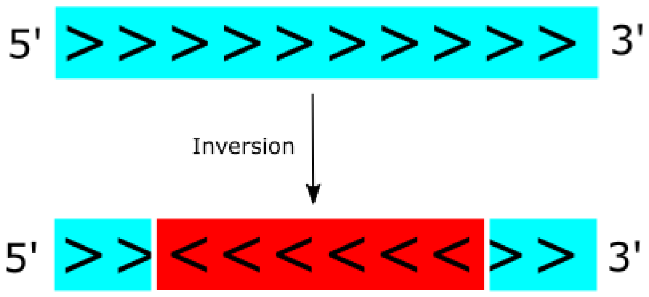
Microinversions (MIs) are small inversions in DNA. Due to difficulty in detection, microinversions and other rare small structural variants on the scale of 50-300bp are not well characterized in variant databases. Microinversions act as genetic markers that distinguish species, strains, and lineages, making their identification a useful tool in distinguishing evolutionarily-related species. Microinversions in parental generations can lead to microdeletions in the child generation, increasing the offspring's change of Williams syndrome and other diseases associated with small deletions. Microinversions have also been linked to Huntington's disease.

Due to the small size and difficulty of detection of microinversions, most current mapping processes (BWA, Bowtie2, STAR, etc.) do not attempt to find microinversions in short reads, instead finding no match and assigning them a very low mapping score or leaving them unmapped. As a result, microinversions tend to be greatly ignored by the scientific community and even in clinical environments.

2 Problem Definition

Let us first define $RC[S]$ to be the reverse complement of the sequence S in the standard biological fashion. Let us also concatenate strings using the + operator.

Formally, we state the problem as follows:
Given a query DNA sequence S of length n and a text T of length m, we say that S contains a d-microinversion with respect to T if, for $1 \le i \le j \le n$, the sequence $S[1..i]+RC[S[i..j]]+S[j..n]$ matches T with at most d total differences.



In order to improve running speed and tractability, we will focus on a sub-problem of the original: the **anchored microinversion problem**.
Given a query DNA sequence S of length n and a text T of length m, we say that S contains a k, d_1, d_2-microinversion with respect to T if, for $k \le i \le j \le n-k$, the sequence $S[k..i] + RC[S[i..j]] + S[j..n-k]$ matches T with at most d_1 differences, and $S[1..k]$ and $S[n-k..n]$ match T at the left and right boundaries of $S[k..i] + RC[S[i..j]] + S[j..n-k]$, respectively.

The anchored version of the problem allows us to use k-mers having d_2 differences to "anchor" the inverted region in the text. We will use this observation to develop a fast, sensitive method of inversion detection in large genomes.

3 Methods

3.1 Indexing

To achieve the desired, speed, we turn to matching schemes that use indexed forms of the genomic text and admit an $O(n)$ upper bound on pattern matching. In this case, we chose to use the Burrows-Wheeler transformation (BWT) to index the genome and used the generated suffix array (SA) for efficient match position retrieval.

To build the suffix array and the BWT, we used the common Larsson-Sadakane algorithm with common improvements (ternary quicksort, no length array, group merging, etc.). Although faster algorithms exist, for our indexing purposes this reasonably-fast algorithm was sufficient. For efficient traversal of both the prefix tree and the suffix tree, we generated both the BWT of the forward text and the BWT of the backwards text.

To allow for genomes with multiple chromosomes, we generated the BWT and SA of all chromosomes concatenated together. In addition, we built an index of the FASTA file, storing the names and offsets into the text of all scaffolds in the FASTA file. All files were saved with a user-specified index base name and reserved filetype endings for the forward BWT (.fbwt), the backwards BWT (.rbwt), the suffix array (.fsa), and the FASTA index (.idx).

3.2 Read Trimming

Since we attempt to match anchors of length k on each end of the read, those sequences must be of high quality. Therefore we implemented the quality-trimming algorithm of BWA and cutadapt and applied it to the 3' end of each read. The algorithm works as follows:

1. Convert ASCII representations of per-base Phred quality scores to numeric values.
2. Subtract the threshold value from each score.
3. Compute partial sums (e.g. consecutive running totals) starting from the 3' end.
4. Move backwards from the 3' end, keeping track of position of the minimum score. If the partial sum rises above zero, stop.
5. Trim the read at the marked minimum position. If the read length is above the minimum length cutoff, return it.

This algorithm allows poor-quality bases to be trimmed off of the read, giving greater confidence that the reverse anchor is a true match to the genome and not a sequencing error. In addition, the algorithm is robust against occasional high-quality base calls in the trimmed-off region that do not affect the overall quality of the region.

3.3 Preprocessing of BWT

In order to achieve the constant-time rprev computations needed for inexact matching, the 2BWT is read into main memory and indexed. The occ array is generated with a user-defined decimation factor k for both forward and reverse BWTs, and the C array is generated from the forward BWT, from a single pass over each BWT. The decimation factor is tunable for space-efficiency vs speed, and allows the computation of all intermediate occ values for the BWTs in at most k accesses of the BWT.

3.4 Inexact Matching

Inexact matching proceeds via a simulated traversal of the suffix or prefix tree using the decimated BWT. Traversal is done in constant-time per character as follows: The index in the suffix array of a suffix formed by prepending a character a onto the current pattern P, which we will denote $\text{rprev}_a(P)$, is calculated using one access of the C array and one occ value calculation, which is performed in $O(k)$ time, where k is the decimation factor, as discussed above – note that this does not depend at all on the state of the pattern or text, and thus the calculation of $\text{rprev}_a(P)$ can be achieved in constant time. Two $\text{rprev}_a(P)$ calculations can be used to find the bound of the interval corresponding to the extension of the pattern P to aP. We will take advantage of this constant-time extension in our inexact matching algorithm.

We use a variant of the A* algorithm to perform a heuristic-based state search over all possible matches. Our heuristic is derived from the matching statistics used by BWA, which lower-bound the number of differences needed to match a suffix of the pattern, $P[i..n]$, to the text for all i. In order to create an admissible heuristic – one that overestimates the possible score of the alignment, in this case, since we are maximizing score – we multiply the number of remaining characters in the pattern (i.e. the number of characters left to match) minus the minimum number of differences (given by the matching statistic) by the match bonus. This

Algorithm 1 Calculate matching statistic values from pattern P and match bonus b

```
1: procedure CALCULATED($P, b$)
2:     $P^R \leftarrow P.reverse$
3:     $k \leftarrow 1$
4:     $l \leftarrow text.length$
5:     $z \leftarrow 0$
6:     for $i \leftarrow 1..P.length$ do
7:         $k \leftarrow C[P^R[i]] + occ[P^R[i], k-1] + 1$
8:         $l \leftarrow C[P^R[i]] + occ[P^R[i], l]$
9:         if $k > l$ then
10:            $k \leftarrow 1$
11:            $l \leftarrow text.length$
12:            $z \leftarrow z + 1$
13:        $D[i] \leftarrow (P.length - (i + z)) * b$
```

overestimates the remaining score it is possible to achieve, and by adding this value to the current score of the interval, we develop an appropriate metric on which to base our A* node priorities. Briefly, our graph search algorithm is as follows:

1. Given a pattern P, attempt to match it exactly. If this succeeds, return the alignment with the maximum score. Otherwise, proceed.

2. Calculate the matching statistics discussed above for P. Establish a priority queue and add the root (e.g. the bi-interval covering the entire BWT) of the simulated suffix tree to it.

3. While the priority queue is not empty, pop off the node with the largest heuristic score. If we have reached the end of the pattern, return that interval. Attempt to extend the interval with each character in the alphabet and gaps in both the text and the pattern, calculating the alignment score for each extension. Set the heuristic score of each extension to the alignment score of each extension plus the matching statistic for the unmatched suffix of the pattern and add each extension to the priority queue.

In addition, the user has the ability to specify the maximum number of mismatches to be considered a valid inexact match; the match bonus, mismatch penalty, and gap penalty; the number of matches to report; and an optional seed length. Specifying a nonzero seed length requires that many characters to be matched at the start of the pattern. This greatly improves the speed of matching, but obviously decreases the sensitivity of the algorithm.

3.4.1 Anchor Matching

In order to bound the region in which an inversion can occur, we require that a forward and reverse anchor of length k (default 18) match the text with at most d_2 mismatches. The process of matching these anchors is quite similar to the inexact matching procedure described above, but with two notable differences: 1) only mismatches are allowed, not gaps; and 2) by default, a large number of possible locations are reported.

When all locations for the forward and reverse anchors have been reported, each forward anchor position is compared with each reverse anchor position, and those that are within a small multiplier of the read length (default 3) of each other are passed to the anchor extension step.

The user has the ability to specify a number of bases to clip off of the front or back of the read before matching occurs; this compensates for the presence or adapters or other sequencing artifacts (for example, strings of high-quality G's at the 3' end of reads in Illumina NovaSeq runs).

3.4.2 Anchor Extension and Inversion Matching

When candidate regions have been determined, they are passed to an anchor extension step. In this step, consecutive k-mers are generated starting from the anchors at each end and matched using standard inexact matching with d_2 differences. If a c-mer is within a user-defined maximum distance of the previous k-mer's location and also within the candidate region, it is merged with the anchor. The total number of differences is tracked for all merged k-mers. When a k-mer no longer matches within the region, that anchor is closed. When both anchors have been closed, the total number of differences is checked against a user-defined threshold, and if it is less than the threshold and the anchors are nonoverlapping, the postion of the read not covered by the anchors is reverse-complemented and matched against the text using inexact matching. If the reverse complement matches within the candidate region with at most d_1 differences, the region is considered to be a microinversion and is reported.

3.5 Data Reporting

Data are reported in tab-delineated, BED-like format. The data fields are:

1. Chromosome
2. Start
3. End
4. Read ID
5. Alignment score of inverted region
6. Number of differences in inverted region

4 Implementation

μInvert is written in native Scala. It utilizes many aspects of the standard library, including the Scala collections module for priority queues, hash maps, and sets. μInvert includes various utilities for indexing, testing alignments and anchors, reading and writing various file formats, and running the main program. μInvert has no dependencies and can easily be built using a standard Scala compiler. In addition, as the indexing object is read-only after the initial construction, multiple threads can access the object at the same time. Therefore, μInvert can run on multiple threads at once. By default, μInvert uses a thread pool to spawn and run inversion checks on multiple reads at one time. When a thread detects an inversion, it prints its output to standard out after that resource is released by any other threads using it.

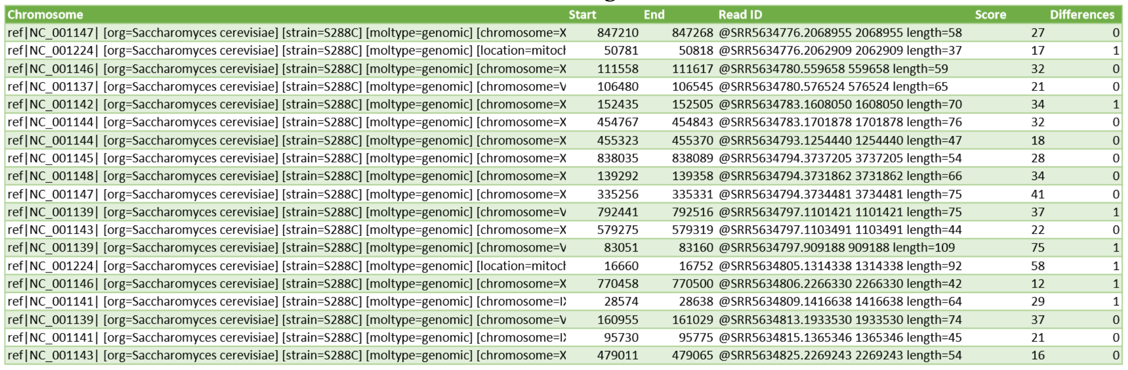
5 Data Sources and Resources

Initially, μInvert was tested on artificially generated inverted reads. In this process, a test genome was split into 100bp fragments, and inversion start points in the range [18,24] and end points in the range [76,82] were selected, and the sequence between these midpoints was reverse-complemented. μInvert was able to recover inversion boundaries and read locations from these artificial data with extremely high fidelity (indeed, the only failures to align were due to the presence of N's in some test genomes).

μInvert was also run against the full genome of *Saccharomyces cerevisiae*, strain S288C, obtained from the *Saccharomyces* Genome Database (SGD). Indexing was performed on the entire genome using the indexing utility built as part of the project, which used approximately 4Gb of memory. In addition, 50 samples (accessions SRR5634776 to SRR5634826) of an inbred yeast cross were selected from a large-scale yeast sequencing project (BioProject accession PRJNA387489), and unaligned reads from these samples were downloaded using the SRA toolkit.

6 Results

μInvert was run on unaligned reads from each selected sample, and 19 inversions with inverted region alignment scores > 10 were discovered in total. This amounts to a 0.000228% inversion detection rate out of 7970456 total unaligned reads.



Chromosome	Start	End	Read ID	Score	Differences
ref|NC_001147| [org=Saccharomyces cerevisiae] [strain=S288C] [moltype=genomic] [chromosome=X	847210	847268	@SRR5634776.2068955 2068955 length=58	27	0
ref|NC_001224| [org=Saccharomyces cerevisiae] [strain=S288C] [moltype=genomic] [location=mitocl	50781	50818	@SRR5634776.2062909 2062909 length=37	17	1
ref|NC_001146| [org=Saccharomyces cerevisiae] [strain=S288C] [moltype=genomic] [chromosome=X	111558	111617	@SRR5634780.559658 559658 length=59	32	0
ref|NC_001137| [org=Saccharomyces cerevisiae] [strain=S288C] [moltype=genomic] [chromosome=V	106480	106545	@SRR5634780.576524 576524 length=65	21	0
ref|NC_001142| [org=Saccharomyces cerevisiae] [strain=S288C] [moltype=genomic] [chromosome=X	152435	152505	@SRR5634783.1608050 1608050 length=70	34	1
ref|NC_001144| [org=Saccharomyces cerevisiae] [strain=S288C] [moltype=genomic] [chromosome=X	454767	454843	@SRR5634783.1701878 1701878 length=76	32	0
ref|NC_001144| [org=Saccharomyces cerevisiae] [strain=S288C] [moltype=genomic] [chromosome=X	455323	455370	@SRR5634793.1254440 1254440 length=47	18	0
ref|NC_001145| [org=Saccharomyces cerevisiae] [strain=S288C] [moltype=genomic] [chromosome=X	838035	838089	@SRR5634794.3737205 3737205 length=54	28	0
ref|NC_001148| [org=Saccharomyces cerevisiae] [strain=S288C] [moltype=genomic] [chromosome=X	139292	139358	@SRR5634794.3731862 3731862 length=66	34	0
ref|NC_001147| [org=Saccharomyces cerevisiae] [strain=S288C] [moltype=genomic] [chromosome=X	335256	335331	@SRR5634794.3734481 3734481 length=75	41	0
ref|NC_001139| [org=Saccharomyces cerevisiae] [strain=S288C] [moltype=genomic] [chromosome=V	792441	792516	@SRR5634797.1101421 1101421 length=75	37	1
ref|NC_001143| [org=Saccharomyces cerevisiae] [strain=S288C] [moltype=genomic] [chromosome=X	579275	579319	@SRR5634797.1103491 1103491 length=44	22	0
ref|NC_001139| [org=Saccharomyces cerevisiae] [strain=S288C] [moltype=genomic] [chromosome=V	83051	83160	@SRR5634797.909188 909188 length=109	75	1
ref|NC_001224| [org=Saccharomyces cerevisiae] [strain=S288C] [moltype=genomic] [location=mitocl	16660	16752	@SRR5634805.1314338 1314338 length=92	58	1
ref|NC_001146| [org=Saccharomyces cerevisiae] [strain=S288C] [moltype=genomic] [chromosome=X	770458	770500	@SRR5634806.2266330 2266330 length=42	12	1
ref|NC_001141| [org=Saccharomyces cerevisiae] [strain=S288C] [moltype=genomic] [chromosome=I)	28574	28638	@SRR5634809.1416638 1416638 length=64	29	1
ref|NC_001139| [org=Saccharomyces cerevisiae] [strain=S288C] [moltype=genomic] [chromosome=V	160955	161029	@SRR5634813.1933530 1933530 length=74	37	0
ref|NC_001141| [org=Saccharomyces cerevisiae] [strain=S288C] [moltype=genomic] [chromosome=I)	95730	95775	@SRR5634815.1365346 1365346 length=45	21	0
ref|NC_001143| [org=Saccharomyces cerevisiae] [strain=S288C] [moltype=genomic] [chromosome=X	479011	479065	@SRR5634825.2269243 2269243 length=54	16	0

A subset of results were investigated using NCBI-BLAST on the S288C genome, confirming that the inverted regions do match to the reported region. Running time was approximately 3-5 hours per file.

7 Further Work

Various improvements can be made to μInvert, chief among them speeding up the rate of indexing, which is currently the limiting factor preventing usable speed for larger genomes. In addition, sensitivity could be improved by not requiring anchors on both ends of the inversion, and accuracy could be improved by considering the mapping region of the read pair (if paired-end reads are used).

8 References

1. He, F., Li, Y., Tang, Y.-H., Ma, J. & Zhu, H. Identifying micro-inversions using high-throughput sequencing reads. BMC Genomics 17, (2016).

2. Chaisson, M. J., et al. "Microinversions in Mammalian Evolution." Proceedings of the National Academy of Sciences, vol. 103, no. 52, 2006, pp. 19824-19829., doi:10.1073/pnas.0603984103.

3. Szamalek, Justyna M., et al. "Polymorphic Micro-Inversions Contribute to the Genomic Variability of Humans and Chimpanzees." Human Genetics, vol. 119, no. 1-2, 2005, pp. 103- 112., doi:10.1007/s00439-005-0117-6.

4. Hobart, Holly H., et al. "Inversion of the Williams Syndrome Region Is a Common Polymorphism Found More Frequently in Parents of Children with Williams Syndrome." American Journal of Medical Genetics Part C: Seminars in Medical Genetics, vol. 154C, no. 2, 2010, pp. 220-228., doi:10.1002/ajmg.c.30258.

5. FAIDX. Faidx(5) Manual Page, www.htslib.org/doc/faidx.html.

6. Cutadapt. cutadapt.readthedocs.io/en/stable/guide.html#quality-trimming.

7. Geeks-for-Geeks A* algorithm. https://www.geeksforgeeks.org/a-search-algorithm/

8. Saccharomyces Genome Database. https://www.yeastgenome.org/

9. BioProject PRJNA387489. https://www.ncbi.nlm.nih.gov/bioproject/PRJNA387489