Mapping-Free and Assembly-Free Discovery of Inversion Breakpoints from Raw NGS Reads

Claire Lemaitre[1], Liviu Ciortuz[1,2], and Pierre Peterlongo[1]

[1] INRIA/IRISA/GenScale, Campus de Beaulieu, 35042 Rennes cedex, France
{claire.lemaitre,pierre.peterlongo}@inria.fr
[2] Faculty of Computer Science Iasi, Romania
ciortuz@info.uaic.ro

Abstract. We propose a formal model and an algorithm for detecting inversion breakpoints without a reference genome, directly from raw NGS data. This model is characterized by a fixed size topological pattern in the *de Bruijn Graph*. We describe precisely the possible sources of false positives and false negatives and we additionally propose a sequence-based filter giving a good trade-off between precision and recall of the method. We implemented these ideas in a prototype called TakeABreak. Applied on simulated inversions in genomes of various complexity (from *E. coli* to a human chromosome dataset), TakeABreak provided promising results with a low memory footprint and a small computational time.

Keywords: structural variant, NGS, reference-free, de Bruijn graph.

1 Introduction

Structural variation is an important source of variations in genomes, that can be involved in phenotypic variations, inherited diseases, evolution and speciation. The extent of structural variations in populations has been only recently acknowledged, thanks mainly to next generation sequencing (NGS). In fact, by sequencing the genomes of several human individuals, one can find more DNA involved in structural variations than in single nucleotide polymorphism (SNP) [8]. However, due to the small size of the reads these variants are much more difficult to identify than SNPs. Most methods proposed so far rely on mapping the reads on a reference genome. The main approach calls structural variant breakpoints when mapped read pairs show discordant mappings with respect to expected insert-size and orientation of the reads [7]. Due mainly to repetitions in complex genomes and mapping errors, these methods suffer from high false positive rates and a small overlap between predictions obtained by different methods [1]. Noteworthy, copy number variations seem to have focussed most attention and efforts, whereas balanced structural variants such as inversions have been less investigated [8], suggesting that the latter are even more difficult to detect in short read data.

All these approaches rely on a reference genome and on a first mapping step. This is a strong limitation when dealing with organisms with no available reference genome or one of poor quality or too distantly related. On the other hand,

A.-H. Dediu, C. Martín-Vide, and B. Truthe (Eds.): AlCoB 2014, LNBI 8542, pp. 119–130, 2014.

one can also perform full *de novo* assembly of re-sequenced genomes and compare the resulting assemblies [6], however *de novo* assembly remains a difficult and resource intensive task and this could be reduced by targeting directly inversion variants. The problem we address is therefore: can we identify inversion signatures directly in raw NGS reads without the need of any reference genome nor full assembly of the reads? Several methods have been developed recently for calling biological events of interest directly from raw unassembled reads, by targeting specific patterns in assembly graphs. Some of them are dedicated to detect SNPs or small indels [10,9,13] or alternative slicing events in RNA-seq data [11]. Cortex_var [4] claims to detect any variant generating a *bubble* pattern, but does not target specifically inversions or other balanced structural variants.

The main contribution of this paper is an analysis and a formal modeling of topological patterns generated by inversions in the *de Bruijn Graph*. Additionally, we propose an algorithm detecting such inversion patterns. This algorithm was implemented in a tool called TakeABreak that was used as a proof of concept and that can be downloaded from `http://colibread.inria.fr/TakeABreak/`. Applying this tool on simulated datasets showed that i) the described model detects with high recall and precision inversion breakpoints ii) approximate repeats present in complex genomes only slightly decrease performances iii) time and memory are very limited.

2 Inversion Pattern in the de Bruijn Graph

2.1 Preliminaries

Given a sequence s on the DNA alphabet $\Sigma = \{A, C, G, T\}$, we use the concept of *k-mers* that are words of length k in s. We denote by $\overleftarrow{s}$ the reverse-complement of sequence s, for instance with $s = TTGC$, $\overleftarrow{s} = GCAA$.

de Bruijn Graph. The approach we propose is based on the use of a *de Bruijn Graph*. Given a set of sequences such as reads in the assembly framework, a *de Bruijn Graph* is a directed graph where the set of vertices corresponds to the k-mers from the sequences, and a directed edge connects a source k-mer a to a target k-mer b if the $k-1$ suffix of a is equal to the $k-1$ prefix of b. Usually, in the genome assembly framework [14], a *de Bruijn Graph* node stores explicitly a k-mer and implicitly its reverse complement. Thus there are two ways of traversing a node: either reading the explicit k-mer or reading the implicit one; we denote this notion by the *state* of the node: *explicit* or *implicit*. In this context, each edge is labeled by the states of its source and target nodes. In the following $n^{\rightarrow}_{\omega}$ denotes the node storing explicitly or implicitly ω, in the state such that reading $n^{\rightarrow}_{\omega}$ provides the k-mer ω. Respectively, $n^{\leftarrow}_{\omega}$ denotes the same node in the state such that reading $n^{\leftarrow}_{\omega}$ provides the k-mer $\overleftarrow{\omega}$.

Given two nodes $n^{\rightarrow}_{start}$ and $n^{\rightarrow}_{stop}$, we say that a path of length l exists from node $n^{\rightarrow}_{start}$ to node $n^{\rightarrow}_{stop}$, *iif* node $n^{\rightarrow}_{stop}$ can be reached using l nodes from node $n^{\rightarrow}_{start}$ and for any traversed node, it should be entered and left in the same state (i.e. explicit or implicit). Let k-path denote a path of length k.

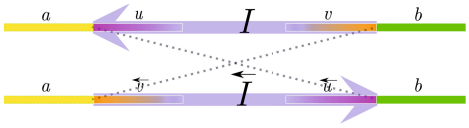

Fig. 1. Sequences aIb and $a\overleftarrow{I}b$, showing the four particular k-mers a, u, v and b at the breakpoints

Inversion. Given a fixed k value and a set of input sequences, we define an inversion as a sequence I of length larger or equal to k such that aIb and $a\overleftarrow{I}b$ occur both at least once, each in any of the input sequences, with a and b being two k-mers. We call u (resp. v) the prefix (resp. suffix) of length k of I. Our inversion model imposes $a \neq \overleftarrow{b}$ and $u \neq \overleftarrow{v}$. Figure 1 proposes a graphical representation of an inversion. We call the *breakpoints* of the inversion, the junctions between the inverted segment and the non-inverted parts. Such a rearrangement generates therefore two breakpoints in each sequence.

2.2 Inversion Pattern

An inversion, such as shown in Figure 1, generates a particular motif in the *de Bruijn Graph*. The only differences in terms of k-mers between both sequences, with and without the inversion, involve the breakpoints of the inversion: only the $k-1$ k-mers spanning each breakpoints differentiate the two sequences. The breakpoints at the left of the inverted segments are then characterized by a *fork* in the *de Bruijn Graph*, which is defined by a common node $n_a^{\rightarrow}$ that branches to two distinct k-paths that end respectively in $n_u^{\rightarrow}$ and $n_v^{\leftarrow}$. Similarly, the other two breakpoints (at the right) are characterized by two k-paths starting from $n_u^{\leftarrow}$ and $n_v^{\rightarrow}$ that join in $n_b^{\rightarrow}$. These two *forks*, being connected by two common nodes (corresponding to the k-mers u and v respectively, and their reverse complements), lead to a particular motif in the *de Bruijn Graph*, that we call the *inversion pattern*, as exemplified in Figure 2.

It is important to note that the definition of the inversion pattern imposes conditions on the four k-mers a, u, v, b. First, $a \neq \overleftarrow{b}$ and $u \neq \overleftarrow{v}$ for the two distinct forks to exist. Second the node $n_a^{\rightarrow}$ must be branching, that is the first nucleotide of u must be different from the first nucleotide of $\overleftarrow{v}$.

One major advantage of this motif is that it can be traversed by 4 k-paths in the *de Bruijn Graph*: one from $n_a^{\rightarrow}$ to $n_u^{\rightarrow}$; one from $n_u^{\leftarrow}$ to $n_b^{\rightarrow}$, one from $n_b^{\leftarrow}$ to $n_v^{\leftarrow}$; one from $n_v^{\rightarrow}$ to $n_a^{\leftarrow}$. Being composed of only fixed length paths, finding the presence of such motif in a *de Bruijn Graph* is rapid and rather simple.

Notice that this motif presents some drawbacks. First, it detects the presence of inversion breakpoints but it does not provide the inversion itself. As second drawback, the motif is perfectly symmetrical: starting from node $n_b^{\leftarrow}$, or $n_u^{\leftarrow}$ or $n_v^{\rightarrow}$ leads to the discovery of the same inversion. As presented Section 3.2, we propose a way to output only once each inversion breakpoints. Finally, such a motif may witness approximate repeats instead of inversions (see Section 3.4).

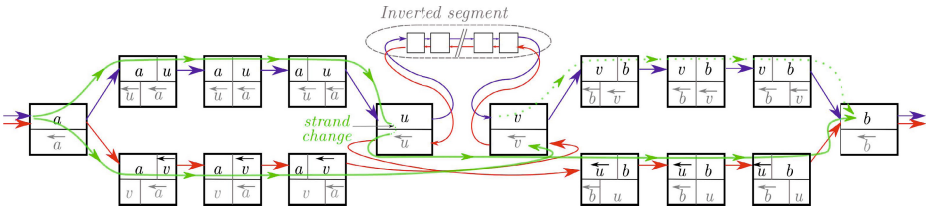

Fig. 2. Schematic example of the inversion pattern generated by sequences aIb (the blue path) and $a\overleftarrow{I}b$ (the red path) in a *de Bruijn Graph* with $k = 4$. Nodes are represented as two-stage boxes, with the upper part in black showing the explicit k-mer and the lower part, in grey, the implicit one. DNA k-mers are not represented, instead the node content shows proportion of full or junction of the four main k-mers a, u, v, b and their reverse complements. For sake of simplicity and without loss of generality, we consider that all k-mers of au, vb, $a\overleftarrow{v}$ and $\overleftarrow{u}b$ are explicitly stored. The state of a node traversed by edges entering and leaving its upper (resp. lower) part is explicit (resp. implicit). The green paths represent the paths enumerated by TakeABreak algorithm. The dashed green path is only checked, once the nodes n_v and n_b are found.

3 Algorithm for Inversion Pattern Detection

3.1 Main Algorithm

This section describes an algorithm for efficient detection of the inversion pattern from an already constructed *de Bruijn Graph*.

A "naive" algorithm for detecting the inversion pattern would be to check for each possible starting k-mer a a k-path from $n_a^{\rightarrow}$ to $n_u^{\rightarrow}$, then from $n_u^{\leftarrow}$ to $n_b^{\rightarrow}$, then from $n_b^{\leftarrow}$ to $n_v^{\leftarrow}$ and then from $n_v^{\rightarrow}$ to $n_\alpha^{\leftarrow}$ and finally checking that $a = \alpha$. This approach would lead to the construction of $4k$-paths from $n_a^{\rightarrow}$ leading to a combinatorial explosion in complex genomes.

We propose an algorithm whose longest walked paths are of length $2k$, then strongly limiting the search space. The main idea is to start from any branching node (a node having more than one outgoing edge) $n_a^{\rightarrow}$, to detect all nodes reachable by a k-path, storing them in several sets N_α depending on the first letter α of these nodes. The second main step is then to detect any node $n_b^{\rightarrow}$ ($\overleftarrow{b} \neq a$) such that there exist a k-path from $n_u^{\leftarrow}$ to $n_b^{\rightarrow}$ and a k-path from $n_b^{\leftarrow}$ to $n_v^{\leftarrow}$, with $n_u^{\leftarrow} \in N_\alpha$ and $n_v^{\leftarrow} \in N_\beta$ and $\alpha \neq \beta$. In such case the pair of sequences au and vb is output. Algorithm 1 proposes a high level presentation of our algorithm.

3.2 Canonical Representation of Occurrences

The inversion pattern presents some symmetries. In most cases (see Section 3.3), the inversion pattern generated by an inversion will be detected by our algorithm as four distinct occurrences each starting from one of the four main nodes: $n_a^{\rightarrow}$, $n_u^{\leftarrow}$, $n_b^{\leftarrow}$ and $n_v^{\rightarrow}$. The output of the algorithm 1 is a pair of words au and vb

1. **Input:** A list of branching nodes and a *de Bruijn Graph* of all input reads.
2. **Provides:** A set of pairs of inversion breakpoint sequences
3. **for** each branching node $n_a^{\rightarrow}$ **do**
4. Compute all paths of length k starting from $n_a^{\rightarrow}$
5. Store all reached nodes starting with letter α in N_α ($\alpha \in \{A, C, G, T\}$)
6. **for** each $\alpha \in \{A, C, G\}$ **do**
7. **for** each $n_u^{\rightarrow} \in N_\alpha$ **do**
8. Compute all paths of length k from $n_u^{\leftarrow}$
9. Store all reached nodes in B
10. **for** each $n_b^{\rightarrow} \in B$ **do**
11. **for** each $n_v^{\leftarrow} \in \cup N_{\beta>\alpha}$ **do**
12. **if** a path of length k exists from $n_b^{\leftarrow}$ to node $n_v^{\leftarrow}$ **then**
13. Output (au, vb)

Algorithm 1. Main algorithm to detect the inversion pattern

depending both on the starting node $n_a^{\rightarrow}$ and the order of detection between $n_u^{\rightarrow}$ and $n_v^{\leftarrow}$. To avoid outputting several times the same inversion, we define its *canonical representation* as the smallest 2-words output in lexicographical order among the eight possible rearrangements: (au, vb), $(a\overleftarrow{v}, \overleftarrow{u}\,b)$, $(\overleftarrow{u}\,\overleftarrow{a}, \overleftarrow{b}\,\overleftarrow{v})$, $(\overleftarrow{u}\,b, a\overleftarrow{v})$, (vb, au), $(v\overleftarrow{a}, \overleftarrow{b}\,u)$, $(\overleftarrow{b}\,\overleftarrow{v}, \overleftarrow{u}\,\overleftarrow{a})$, $(\overleftarrow{b}\,u, v\overleftarrow{a})$. Only the canonical representation is reported and only once.

3.3 Presence of Small Inverted Repeats at the Breakpoints

If an inversion contains an inverted repeat of size larger or equal to $k-1$ at its breakpoints, this inversion will not generate the inversion pattern since it does not generate new k-mers nor new paths in the *de Bruijn Graph* with respect to the non inverted sequence. This is the case for instance if $a = \overleftarrow{b}$ or $u = \overleftarrow{v}$.

In the case of an inverted repeat whose length is smaller than $k-1$, such inversion still generates the inversion pattern, however the latter is not be fully symmetrical. Suppose there is an inverted repeat of size $x < k-1$ at the breakpoints or overlapping the breakpoints. As the first node $n_a^{\rightarrow}$ must be branching, it imposes that the repeated sequence is included in k-mer a and considered outside the inverted segment (note that even with the full sequences at hand, we can not decide if the inversion includes the repetition entirely, partially or not at all). The suffix of size x of a is then equal to the prefix of size x of $\overleftarrow{b}$. It implies also that there are no more $k-1$ distinct k-mers at each breakpoint that differentiate the two sequences, but $k-1-x$ k-mers. Therefore the two forks of the inversion pattern, represented in Figure 2, are shortened. In this case, the nodes $n_u^{\leftarrow}$ and $n_v^{\rightarrow}$ reached after k-paths are not necessarily branching and can not constitute starting k-mers in other occurrences of the inversion pattern. Instead k-mers at the end of $(k-x)$-paths in the fork constitute the other starting k-mers.

In fact, such an inversion will still be detected as 4 occurrences but the sets of k-mers a, u, v and b will be different depending on the starting k-mer. Starting from inside ($n_u^{\leftarrow}$ or $n_v^{\rightarrow}$) or outside ($n_a^{\rightarrow}$ or $n_b^{\leftarrow}$) the inverted segment I will

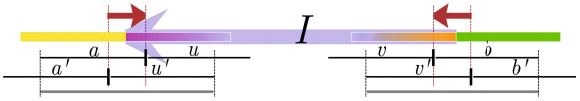

Fig. 3. Example of an inversion with small inverted repeats (red arrows) at the breakpoints. Breakpoint sequences au, vb (resp. $a'u', v'b'$) are obtained starting from nodes $n_a^{\rightarrow}$ or $n_b^{\leftarrow}$ (resp. $n_{u'}^{\leftarrow}$ or $n_v^{\rightarrow}$). The unique canonical representative is represented by the two grey bottom lines.

generate two distinct sets of $2k$ words overlapping on $2k - x$ characters. To avoid duplicating once again artificially the number of occurrences, the output of the algorithm truncates the k-mers u and $\overleftarrow{v}$ such that all starting k-mers give the same sets of words (here of size $2k-x$) and a unique canonical representative can be computed for each of the four occurrences (Figure 3).

3.4 Distinguishing Inversions from Approximate Repeats

Some approximate repeats may generate the inversion pattern in the *de Bruijn Graph* and are thus a source of false positives. Consider for instance that a given sequence au has at least four approximate copies in the sequence, such that au, au', $a'u$ and $a'u'$ with $u' \simeq u$ (at least the first letter is different) and $a' \simeq a$ (at least one substitution or indel anywhere in a). In this situation, without loss of generality, calling $\overleftarrow{b} = a'$ and $\overleftarrow{v} = u'$, the four paths au, $vb(= \overleftarrow{a'u'})$, $a\overleftarrow{v}(= au')$, and $\overleftarrow{u}b(= \overleftarrow{a'u})$ exist and mimics the inversion pattern. More generally, high similarity between a and $\overleftarrow{b}$ and between u and $\overleftarrow{v}$ is characteristic of an approximate repeat.

In order to distinguish inversions from false positives due to approximate repeats, we filter out occurrences of the inversion model where a and $\overleftarrow{b}$ and where u and $\overleftarrow{v}$ have a Longest Common Subsequence (LCS) size higher than a given threshold. As an optimization, we try to detect earlier cases where a and $\overleftarrow{b}$ are too similar during the k-path search from $n_u^{\leftarrow}$ to $n_b^{\rightarrow}$. During this step, we forbid paths that go back on the previous path towards first node $n_a^{\rightarrow}$, since the longer we take the former path, the more similar will be k-mers a and $\overleftarrow{b}$. However, to permit the detection of inversions with small inverted repeats at the breakpoints, we tolerate to go back on the former path for a given maximum number of nodes (this parameter is usually fixed to 8).

Additionally, it is well known that high copy number repeats with approximate copies are an important source of complexity generating highly branching subparts in the *de Bruijn Graph*. Searching for inversions in such complex part of the graph presents two main drawbacks. First, as previously mentioned, it is a source of false positives, and second, it generates a possible huge number of k-paths whose enumeration can be highly time consuming. To overcome these two drawbacks we stop the inversion pattern detection from a node n_a as soon as the product of the cardinality of the two largest sets N_α is bigger than a limit (called

LCT for *Local Complexity Threshold*). This product is a lower approximation of the minimal number of couples of k-paths that are to be enumerated from the starting n_a. Similarly, we apply the same strategy once a set of nodes B (see Algorithm 1 line 9) is detected from a node $n_u^{\rightarrow}$: if the cardinality of B times the cardinality of the largest set N_α is larger than LCT, then the exploration from node $n_u^{\rightarrow}$ stops. This last product reflects another lower bound of the number of paths to be checked. Note that this approach highly limits both false positives and computational time, while having a limited impact on recall, as shown in results Section 4.2.

3.5 TakeABreak Implementation

We implemented the proposed algorithm in a prototype called TakeABreak. It takes as input one or several sets of sequences in fasta or fastq format. Its main parameters are the k-mer size k; $max\_sim \in [0, 100]$ the maximal similarity authorized between a and $\overleftarrow{b}$ and between u and $\overleftarrow{v}$, expressed as a percentage of k-mer size; and LCT: the Local Complexity Threshold (see Section 3.4). Prior to the inversion pattern detection phase, the *de Bruijn Graph* is constructed using the Minia data structure [2,12]. This graph is constructed using only k-mers having at least 3 occurrences in order to discard sequencing errors. This is a very common parameter used for *de Bruijn Graph*-based assembly. The second phase implements algorithm 1. The output is a *fasta* file containing, for each detected inversion, its breakpoint sequences. These are the $2k-2$ (or $2k-x-2$ in the case of an inverted repetition of size x) words centered on the canonical representation (au, vb). By removing the two extreme nucleotides, it ensures that the output paths are made of the k-mers that overlap the breakpoints and that must be specific to each sequence.

TakeABreak was implemented in C++ with the GATB library [3], providing notably the Minia data structure, and it can be downloaded from `http://colibread.inria.fr/TakeABreak/`.

4 Results

To evaluate the ability of TakeABreak to detect inversions in reads, we generated artificial read datasets. First, non-overlapping inversions of varying sizes were simulated in a copy of a real genome. Then we simulated the sequencing processing on both genomes, the original one and the one with artificial inversions. Finally both read sets were given as input to TakeABreak. To classify the results of TakeABreak as true positive or false positive, we first generated for each simulated inversion its canonical representation of breakpoints such as described in sections 3.2 and 3.3 and then called a prediction of TakeABreak as true positive if it is exactly present in this set of true breakpoints. Finally, recall and precision were computed as follows: recall as the number of true positives over the number of simulated inversions, and precision as the number of true positives over the number of predictions.

In more details, inversions were simulated as follows. Each inversion was put sequentially. For each inversion, its first breakpoint is chosen uniformly along the sequence, then its size is sampled uniformly in a given interval (here $[k - 1000]$), finally if it does not overlap and is sufficiently far from a formerly placed inversion (the min distance was fixed to k nucleotides) the inversion is kept and its sequence is reversed-complemented. To simulate reads, 100 bp reads are sampled uniformly along the genome, sequencing errors are put also uniformly with 1 % rate, the depth of coverage was fixed to 40x for each genome.

4.1 Results on a Bacterial Genome

TakeABreak was first evaluated on a simple and small dataset based on the bacterial *E. coli K12* genome, in which 1000 random inversions were simulated. TakeABreak was applied on this simulated dataset with default parameters ($k = 31$, $max\_sim = 80\%$, $LCT = 100$). On this simple dataset, TakeABreak proved to be highly efficient to detect inversion breakpoints, since it predicted the 1000 true positive inversions, leading to a 100% recall for 100% precision (see Table 1). Cortex_var bubble caller [4] was run on the same data and failed to detect any of the simulated inversions.

Table 1. Precision and recall results for TakeABreak on simulated datasets. The first part of the table presents results obtained with default parameters ($k = 31$, $max\_sim = 80\%$, $LCT = 100$), the second part shows the decrease of precision when relaxing filtering parameters ($k = 31$, $max\_sim = 100$, $LCT = 10000$). # FP indicates the amounts of false positives.

	Recall (%)	Precision (%)	# FP
E. coli genome - default parameters	100.00	100.00	0
C. elegans genome - default parameters	96.00	99.07	9
Human chromosome 22 - default parameters	87.60	92.50	71
C. elegans genome - relaxed parameters	99.60	0.37	271,374
Human chromosome 22 - relaxed parameters	93.50	0.06	1,442,760

4.2 Results on More Complex Genomes

Bacterial genomes are small and contain few repeats, leading to rather simple *de Bruijn Graph* and few false positives of the inversion pattern. To evaluate TakeABreak on more complex genomes, we simulated inversions in eukaryotic genomes and chromosomes, first in the full *C. elegans* genome ($\sim$ 100 Mbp) and second in human chromosome 22 ($\sim$ 35 Mbp without N bases). As expected (see Section 3.4), precision and recall decrease when the repeat content of the genome increases, as shown in Table 1. However, this effect is greatly limited by the use of filtering parameters max_sim and LCT, since relaxing these parameters leads to millions of false positives (see Table 1).

Note that these parameters have to be fixed carefully as they can also affect the recall, as shown in Figure 4 where precision and recall results are represented for varying values of *max_sim* and *LCT*. This figure shows that both parameters are useful to decrease the false positive rate and that the proposed default parameters offer a good trade-off between precision and recall.

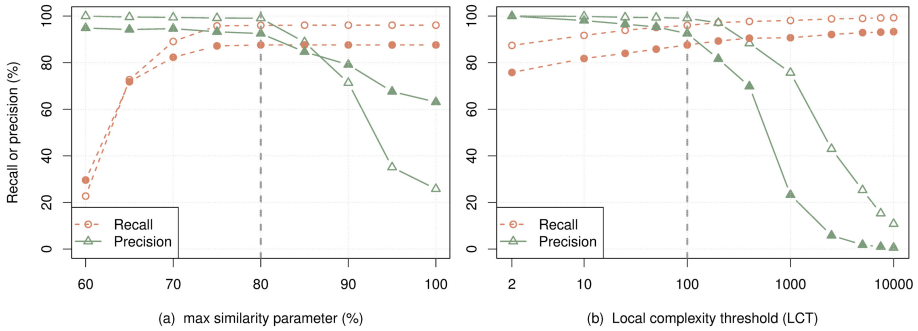

Fig. 4. Effect of the filtering parameters, *max_sim* (a) and *LCT* (b), on precision and recall values for the *C. elegans* (open symbols) and human chromosome 22 (solid symbols) datasets. Vertical dashed lines represent the default parameters.

4.3 Time and Memory Performances

These tests were performed with 2.3 GHz Intel Core i7 processors, with 8GB RAM. Table 2 shows time and memory performances of the prototype TakeABreak for the different datasets. Time and memory increase with the complexity of the datasets. Even if the human chromosome dataset is smaller than the *C. elegans* one, the computational time is much larger for human. This shows that the complexity of the graph is not solely linked to the size of the genome, but also to its repeat content, with human chromosome 22 high copy number repeats generating sub-parts of the graph with high density of branching nodes and imbricated patterns of inversions.

Nevertheless, as presented Table 2, TakeABreak scales up to complex and large datasets. The highest memory consumption is reached during the *de Bruijn Graph* construction and is limited to 1GB, allowing TakeABreak to be executed on a standard desktop (note that the full human genome would need 6GB of memory [12]). The graph construction time is limited to at most 20 minutes for the most complex dataset we used. The time needed for enumerating all inversion patterns is sensitive to genome complexity (from 1 second for *E. coli* to one hour and a half for human chromosome 22) and still remains acceptable. Moreover, in addition to dramatically improving the precision (see Section 4.2), we can notice that the default filters highly reduce the computational time (*e.g.* from 7h40 without filters to 1h30 with filters on the human dataset).

Table 2. Time and memory performances of TakeABreak on simulated datasets with default parameters. For each dataset we indicated the number of reads and the total number of nucleotides it contains. Time values given in parenthesis are those obtained while relaxing the filter parameters (bottom part of Table 1).

	Time (s)		Memory	
	Graph construction	Inversion detection	Graph construction	Inversion detection
E. coli genome (3.7M reads 370 Mbp)	24	1	1GB	3MB
C. elegans genome (80M reads, 8 Gbp)	78	935 (7408)	1GB	53MB
Human chromosome 22 (28M reads 2.8 Gbp)	1205	5412 (27554)	1GB	153MB

5 Discussion and Conclusion

In this work, we formalized for the first time the topological pattern generated by the inversion of a DNA segment in the *de Bruijn Graph* representing both sequences, with and without the inversion.

We also proposed a first analysis of what kind of variant or sequence feature can or can not generate this pattern. The pattern involves only the $2k$ sequences around the breakpoints of the inversion (k being the k-mer size of the *de Bruijn Graph*). Therefore the size of the inversion does not limit the existence of the pattern as long as it is greater than k. The pattern is based on four k-mers at each side of the breakpoints that must be identical between both sequences with and without the inversion. As a consequence, the breakpoint regions must not contain any substitution or indel at distance less than k from both breakpoints, that is as if the inversion was generated by perfect blunt-ended double strand breaks. Finally, another feature that can prevent an inversion from generating this pattern is the presence of an inverted repeat of size $\geq k-1$ at each breakpoints since all breakpoint sequences will follow the same paths in the *de Bruijn Graph*.

On the other hand, we showed that the pattern can be generated by other sequence features than inversions. First, some approximate repeats with appropriate combinations of differences can easily generate this pattern, these are considered as false positive or noise since they do not differentiate the compared genomes. If in small bacterial genomes, this situation is quite rare, our tests show that in more complex genomes this can dramatically increases the number of false positive calls, explaining why we added a sequence-based filter to this topology-based pattern. Indeed, with high copy number repeats, such as transposable elements in eukaryotic genomes, such combinations of at least two differences in repeats of size $2k$ is very likely to happen. Another variant that can generate the inversion pattern is the reciprocal translocation, since it has also two breakpoints per sequence (with and without the translocation) with the same combinations of four k-mers. We consider this as another advantage

of this pattern, because, in this case, this is also a structural variant that can differentiates genomes and has therefore a potential biological interest.

In this work, we also proposed and implemented an efficient algorithm to enumerate all inversion patterns in a *de Bruijn Graph*, together with powerful filtering strategies to avoid false positives due to approximate repeats. The tests we performed on simulated data prove that this approach enables to recover almost all simulated inversions quite rapidly. The power of this pattern lies mainly in its fixed size. Contrary to structural variants with only one breakpoint, such as insertions and deletions, it is not necessary to traverse in the graph the full inverted segment to detect the presence of the inversion. In fact, insertions and deletions generate *bubble* patterns that can only be detected by traversing the full inserted or deleted sequence [4,11], this strongly limits the size of detectable events (at least in complex genomes) and increases the computational time. Inversions too could be detected as *bubbles* but Cortex_var did not detect them, even the smallest ones, probably because it requires bubbles not to contain any branching node while such nodes are inherent of inversion patterns, as shown in this paper.

The tests and simulations we performed were meant to demonstrate the validity of our pattern and of our algorithm, we are aware that they can still be improved to better fit actual genome re-sequencing data. Indeed, only inversions were simulated without any other polymorphism that could impact the breakpoints. Inversions were put following a uniform distribution, whereas rearrangement distribution is likely not random and some rearrangements can be linked for instance to repeated sequences. Finally, only perfect blunt-ended breakpoints were simulated which may not reflect all molecular mechanisms of such events (for instance, non-homologous end joining is known to generate small indels at the very breakpoint). For all these reasons, recall values we obtained are likely to be over-estimated with respect to real inversions. However, our promising results on such simulated inversions open the way to further improvements of the model.

First, the model could largely be improved by additionally including SNP or small indel detection models such as [13]. Thus both SNP and inversion detection would not suffer from each other. This would improve recall for events that lie close to each other and could be used as preliminary step of the assembly process. Second, the breakpoint detection algorithm could be coupled with a third party local assembly or gap-filling tool, such as MindTheGap [5], to get the sequence of the inverted segment and not only its breakpoints. Finally, other biological variants can benefit from this approach. As already mentioned, reciprocal translocations can be detected by the proposed model as is. Additionally, the model could be extended to the detection of other rearrangements that have more than two breakpoints, such as transpositions that generate a three-fork model, thus showing high similarity with the model proposed in this paper.

Acknowledgments. The authors warmly thank Erwan Drezen and Guillaume Rizk for implementation support and Marie-France Sagot for interesting discussions. This work was supported by the Région Bretagne SAD-MIRAGE

project and by the ANR (French National Research Agency), ANR-12-BS02-0008 *Colib'read* project and ANR-12-EMMA- 0019-01 *GATB* project.

References

1. Alkan, C., Coe, B.P., Eichler, E.E.: Genome structural variation discovery and genotyping. Nat Rev. Genet. 12, 363–376 (2011)
2. Chikhi, R., Rizk, G.: Space-efficient and exact de bruijn graph representation based on a bloom filter. Algorithms for Molecular Biology 8, 22 (2013)
3. Drezen, E., et al.: The Genome Assembly and Analysis Tool Box, `http://gatb.inria.fr/` (Manuscript in Prep. 2014)
4. Iqbal, Z., Caccamo, M., Turner, I., Flicek, P., McVean, G.: De novo assembly and genotyping of variants using colored de bruijn graphs. Nature Genetics 44, 226–232 (2012)
5. Lemaitre, C., et al.: MindTheGap Software, `http://mindthegap.genouest.org/` (Manuscript in Prep. 2014)
6. Li, Y., Zheng, H., Luo, R., Wu, H., Zhu, H., Li, R., et al.: Structural variation in two human genomes mapped at single-nucleotide resolution by whole genome de novo assembly. Nat. Biotechnol. 29, 723–730 (2011)
7. Medvedev, P., Stanciu, M., Brudno, M.: Computational methods for discovering structural variation with next-generation sequencing. Nat Methods 6, S13–S20 (2009)
8. Mills, R.E., Walter, K., Stewart, C., Handsaker, R.E.: 1000 Genomes Project: Mapping copy number variation by population-scale genome sequencing. Nature 470, 59–65 (2011)
9. Nordström, K.J.V., Albani, M.C., James, G.V., et al.: Mutation identification by direct comparison of whole-genome sequencing data from mutant and wild-type individuals using k-mers. Nature Biotechnology 31, 325–330 (2013)
10. Peterlongo, P., Schnel, N., Pisanti, N., Sagot, M.-F., Lacroix, V.: Identifying sNPs without a reference genome by comparing raw reads. In: Chavez, E., Lonardi, S. (eds.) SPIRE 2010. LNCS, vol. 6393, pp. 147–158. Springer, Heidelberg (2010)
11. Sacomoto, G.A., Kielbassa, J., Chikhi, R., Uricaru, R., et al.: Kissplice: de-novo calling alternative splicing events from rna-seq data. BMC Bioinformatics 13, S5 (2012)
12. Salikhov, K., Sacomoto, G., Kucherov, G.: Using Cascading Bloom Filters to Improve the Memory Usage for de Brujin Graphs. In: Darling, A., Stoye, J. (eds.) WABI 2013. LNCS, vol. 8126, pp. 364–376. Springer, Heidelberg (2013)
13. Uricaru, R., et al.: discoSnp Software, `http://colibread.inria.fr/discosnp/` (Manuscript in Prep. 2014)
14. Zerbino, D.R., Birney, E.: Velvet: algorithms for de novo short read assembly using de bruijn graphs. Genome Research 18, 821–829 (2008)