Annexe de l'HDR :
recueil de publications

Pierre Peterlongo

Publications associées au chapitre 2

Identifying snps without a reference genome by comparing raw reads

Identifying SNPs without a reference genome by comparing raw reads

Pierre Peterlongo[1], Nicolas Schnel[1], Nadia Pisanti[2], Marie-France Sagot[3], and Vincent Lacroix[3]

[1] INRIA Rennes - Bretagne Atlantique, EPI Symbiose, Rennes, France
[2] Dipartimento di Informaticà, Universita di Pisa, Italy
[3] INRIA Rhône-Alpes, 38330 Montbonnot Saint-Martin, France and Université de Lyon, F-69000 Lyon; Université Lyon 1; CNRS, UMR5558, Laboratoire de Biométrie et Biologie Evolutive, F-69622 Villeurbanne, France

Abstract. Next generation sequencing (NGS) technologies are being applied to many fields of biology, notably to survey the polymorphism across individuals of a species. However, while single nucleotide polymorphisms (SNPs) are almost routinely identified in model organisms, the detection of SNPs in non model species remains very challenging due to the fact that almost all methods rely on the use of a reference genome. We address here the problem of identifying SNPs without a reference genome. For this, we propose an approach which compares two sets of raw reads. We show that a SNP corresponds to a recognisable pattern in the de Bruijn graph built from the reads, and we propose algorithms to identify these patterns, that we call *mouths*. We outline the potential of our method on real data. The method is tailored to short reads (typically Illumina), and works well even when the coverage is low where it reports few but highly confident SNPs. Our program, called kisSnp, can be downloaded here: http://alcovna.genouest.org/kissnp/.

1 Introduction

Biology in general, and genomics more particularly, witnessed a revolution in the middle 1970s with the development of rapid DNA sequencing techniques, notably the Sanger method which remained the standard approach for sequencing including whole genomes until the early years of the twenty first century. We have since then been witnessing a second revolution, various orders of magnitude bigger than the first, with the advent of the so-called "next generation sequencers" (NGS for short) which enable to obtain up to several hundred million bases in one single run at increasingly lower costs. These include (not exclusively) the 454 Life Sciences, SOLiD Applied Biosystems and Illumina technologies, each with its own characteristics in terms of read length and error rate. Such characteristics are however evolving extremely fast, faster indeed than the algorithms developed to handle the data such technologies produce.

This incredible acceleration has two implications that motivate the work presented in this paper: first it is now possible to obtain data for various individuals of a same species and thus to investigate the genetic differences among such individuals, and second, increasingly more often this will concern species for which

we have no genome of reference, that is no genome already fully sequenced and assembled that could guide the investigation.

The genetic markers that will be of interest in this paper are so-called Single Nucleotide Polymorphisms (SNP for short). These correspond to a DNA sequence variation that occurs when a single nucleotide – A, T, C, or G – in a genome differs among members of a species or between paired chromosomes in an individual. There are two types of SNPs: substitutions or insertions/deletions. We focus here on the first type, that is on substitutions of single nucleotides.

Identifying SNPs in a population may have a wide range of applications that goes from assessing the polymorphism of the population, linking this polymorphism to phenotype information, or selecting SNPs as markers of subpopulations. However, while SNPs are almost routinely identified in model organisms, the detection of SNPs in non model species remains very challenging due to the fact that almost all methods to identify SNPs rely on the use of a reference genome.

Our objective is, given high-throughput read data for a pair of individuals, to identify a set of SNPs with good confidence, without having to perform an assembly of the reads with all the possible mistakes this entails, in a context where we do not have a reference sequence to help the identification.

We are aware of only two publications, dating both from 2010, that deal with the same problem [3, 9]. Recognisably, the major difficulty one faces is due to the presence of errors in the reads, which may be mistaken for a SNP. Additionally, the presence of inexact repeats in the genomes of the studied individuals, may further harden the task. In this paper, we restrict to the case where there is only one genomic variant for each individual (we say that the individuals are homozygous). In this context, the issue of repeats is greatly reduced.

Ratan *et al.* [9] first filter the reads in order to remove the repetitive sequences, then create clusters of overlapping reads which they assemble using a short read assembler. The SNPs are finally identified in the micro-assembled regions using a combination of filters, based on the number of reads supporting each variant or the distance of the SNP w.r.t. the end of the contig.

Unlike Ratan *et al.*, we chose not to use an assembler, which we think can make undesired choices as to sequence variants to remove during the assembly. Indeed, the purpose of an assembler is not to identify SNPs but to propose one reference sequence compatible with the data. Similar to Canon *et al.* [3], we work with raw reads, but we go further than a statistical description of the reads and propose to locally reconstruct the de Bruijn graph in order to identify SNPs. The use of a de Bruijn graph in computational biology was introduced by Pevzner *et al.* in 2001 [8] and used since then as a first step by many short read assemblers.

The key point of our method is that a SNP corresponds to a recognisable pattern in the de Bruijn graph, which we call a *mouth*, each *lip* of the mouth representing an individual variant of a same genomic locus.

Our aim is to directly find the mouths that may be reliably associated to a SNP without making use of any preliminary filters that may eliminate repeats. This is important because, although not explicitely stated, such filters seem to strongly rely on the assumption of an approximately uniform coverage of

each sequence position by the reads in the available data. This assumption is usually not true. Moreover, the biases in read coverage may even vary across two sequencing experiments of the same genomic sample. This means that filters may remove sequences which in fact do not belong to repeats.

We thus present in this paper an algorithm which takes as input two sets of short reads (Illumina or AB/SOLiD) and outputs candidate SNPs (i.e. mouths in the de Bruijn graph), without performing any filtering nor using a short read assembler. This is what we call a comparative micro-assembly. This method is new as, as far as we know, no other treats data coming from distinct sequencing experiments. This approach presents the interest of taking advantage of differences in the data directly into the heart of the algorithm and not in a post-treatment step. SNPs are thus detected on raw read data instead of on pre-assembled sequences. We applied our algorithm on data simulated using MetaSim [10], where we show under which sets of parameters the method works best. We finally apply the method to real data for *Escherichia coli*, for which experimentally validated SNPs are available [2], which is very rare. We show that our method successfully identifies the previously known SNPs, but also predicts new SNPs missed by the conservative method used in the original publication [2].

2 Preliminaries

Sequence, k-mers, prefix, suffix. A *sequence* is composed by zero or more symbols from an alphabet Σ containing $|\Sigma|$ distinct characters. A sequence s of length n on Σ is denoted also by $s[0]s[1]\dots s[n-1]$, where $s[i] \in \Sigma$ for $0 \leq i < n$. The length of s is denoted by $|s|$. Finally, we denote by $s[i,j]$ the *substring* $s[i]s[i+1]\dots s[j]$ of s. In this case, we say that the substring $s[i,j]$ occurs at position i in s. We call *k-mer* a substring of length k. If $s = uv$ for $u, v \in \Sigma^*$, we say that v is a *suffix* of s and that u is a *prefix* of s.

De Bruijn graph. Each node of a de Bruijn graph stores exactly one k-mer. An edge connects a node n_0 to a node n_1 if the suffix of length $k-1$ of the k-mer corresponding to node n_0 is equal to the prefix of length $k-1$ of the k-mer corresponding to node n_1.

A category of *de novo* read-assembly methods such as SOAPdenovo [7], Euler [8] and Velvet [11] (to mention a few) uses the de Bruijn graph as a fundamental data structure. In a few words, reads are first divided into overlapping k-mers, then the associated de Bruijn graph is created and finally Eulerian paths are found in the graph for reconstructing the initial genomic sequence, or fragments thereof that are as large as possible (contigs).

One of the main difficulties encountered by such methods comes from the sequencing errors that generate substitutions and insertions/deletions in the data that must then be assembled. Such errors lead to loops in the de Bruijn graph which may hinder the Eulerian path detection. A first step in such algorithms consists thus in "cleaning the data" by removing suspicious reads and substituting suspect nucleotides. This cleaning step may be problematic when looking

for SNPs as it may remove a significant part of them that will be mistakenly considered as sequencing errors.

3 Comparative micro-assembly model

Our method compares reads generated by two distinct sequencing experiments, and creates parts of the de Bruijn graph potentially linked to a SNP between these two experiments, thereby detecting such SNPs.

The main idea is that the de Bruijn graph of k-mers stemming from two sequences that contain a SNP presents a mouth shape as shown in Fig. 1. The algorithm described in Section 4 detects and constructs such graph shapes directly from the non-assembled k-mers coming from the sets of reads of two distinct sequencing experiments. It is important to notice that the algorithm does not reconstruct the full de Bruijn graph but focuses only on putative SNPs by building mouths.

Mouth model definition. In a de Bruijn graph of k-mers coming from reads of two sequencing experiments (reads A and reads B), a *mouth* is composed by:

- an upper path of k overlapping k-mers $\{a_0..a_{k-1}\}$ resulting from the reads of at least set A. This path is called the *upper lip* of the mouth;
- a lower path of k overlapping k-mers noted $\{b_0..b_{k-1}\}$ resulting from the reads of at least set B. The a_i's and the b_i's differ by one substitution. This path is called the *lower lip* of the mouth;
- a left (resp. right) node, noted c_{-1} (resp. c_k) that corresponds to a k-mer present in both sets A and B and is connected to both a_0 and b_0 (resp. a_{k-1} and b_{k-1}). These k-mers are called the *closing k-mers* of the mouth.

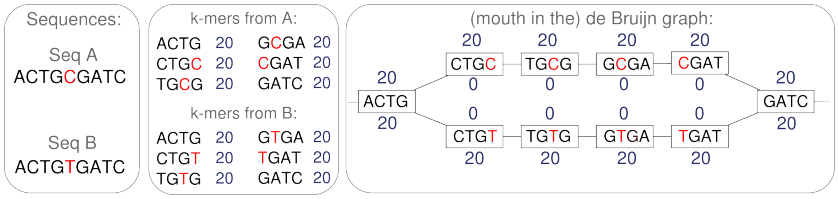


Fig. 1. A SNP between two genome fragments (Seq A and Seq B) generates a mouth shape (rightmost frame) in the de Bruijn graph of the k-mers (here $k = 4$) extracted from Seq A and Seq B. It is assumed in this example that the coverage is exactly 20 (each position of each sequence is covered by 20 reads, thus each position gives rise to 20 k-mers). In the rightmost frame, the number above (resp. below) the nodes indicates the number of occurrences in Seq A (resp. Seq B).

Taking into account the *k*-mer counts. Let us first consider the case where the sequencing is perfect: uniform coverage $\mathcal{C}$ and no sequencing errors nor repeats. In such case, all k-mers from A covering a SNP have $\mathcal{C}$ occurrences more than the same k-mers from B, and *vice-versa*. We considered this theoretical perfect coverage $\mathcal{C} = 20$ in the example of Fig. 1. In practice, the coverage is not uniform and the reads contain errors. The consequence is that the k-mer count difference between experiments will not be constant along the mouth. To account for this, we introduce a parameter called δ, which is meant to capture a deviation from the exact case. Below, we describe the mouth model with counts.

Mouth model integrating k-mer counts. For any k-mer ω, let $mult_A(\omega)$ (resp. $mult_B(\omega)$) be its number of occurrences in the set of reads A (resp. B). We define $diff(\omega) = mult_A(\omega) - mult_B(\omega)$. The SNP mouth model integrates the k-mer number of occurrences as follows. A special k-mer a_{op} called the *opening k-mer* is chosen as reference (see Section 4). If a_{op} is in the upper (resp. lower) lip, for any k-mer a_i contained in a node of the upper (resp. lower) lip, we have $diff(a_i) = diff(a_{op}) \pm \delta$ while for any k-mer b_i contained in a node of the lower (resp. upper) lip, we have $diff(b_i) = -diff(a_{op}) \pm \delta$. The left and right closing k-mers c_{-1} and c_k have no counting properties.

4 Algorithm kisSnp for mouth detection

Algorithm outline. The algorithm kisSnp takes as input two sets of reads (A and B) coming from two distinct sequencing experiments. The output is a set of pairs of micro-assembled sequences, each of length $2k-1$, differing by exactly one substitution located at the central position. Those correspond to putative SNPs detected thanks to the mouth model. The algorithm is divided into three main steps:

- For each set A and B, extract the k-mers and their reverse complement and store them in a tree together with their number of occurrences.
- Create the mouths (detailed in Section 4.1):
  - For each possible opening k-mer a_{op}, detect all possible opposite opening k-mers b_{op} distant by one substitution from a_{op} and fulfilling the counting model.
  - For each pair (a_{op}, b_{op}), construct the mouth by extending the k-mers to the right and to the left with coherent k-mers (*i.e.* overlapping on $k-1$ characters and fulfilling the counting model).
  - Stop the right and left extensions once the mouth is closed or no extension can be found.
- Check that the found mouths are coherent with the reads (detailed in Section 4.3).

4.1 Creating the mouths

Selection of the k-mers opening the mouth. The opening k-mer a_{op} is selected such that $max\,(mult_A(a_{op}), mult_B(a_{op})) < max\,(mult_A(a_i), mult_B(a_i))$ for any k-mer $a_i \neq a_{op}$ and $diff(a_{op}) \neq 0$. In other words, a_{op} is the k-mer having the smallest number of occurrences in either set A or B (possibly zero occurrence in one of the two sets), and is such that it occurs more in a set than in the other, possibly due to a SNP. The rationale for choosing the k-mer with the smallest count is to avoid choosing a k-mer involved in a repeat for opening the mouth. The opposite opening k-mer b_{op} is selected such that a_{op} and b_{op} are distant by exactly one substitution and $diff(b_{op}) = -diff(a_{op}) \pm \delta$. Notice that the substitution position between the k-mers a_{op} and b_{op} may be anywhere. We denote by p this substitution position ($p \in [0, k-1]$). It is worth noticing that, for a given opening k-mer a_{op}, several (at most $(|\Sigma|-1)k$) distinct k-mers b_{op} may satisfy these conditions. They are all iteratively tested as mouth openers.

Extending and closing the mouth. Once a pair of opening k-mers (a_{op}, b_{op}) is selected, a recursive procedure extends them to the right and left with other k-mers fulfilling the following conditions (also shown in Fig. 2). The k-mers a_{i+1} and b_{i+1} may extend a_i and b_i iff:

- $p \geq 0$ (the closing k-mer c_k has not yet been reached), and
- $a_i[1, k-1] = a_{i+1}[0, k-2]$ and $b_i[1, k-1] = b_{i+1}[0, k-2]$ (the new k-mers overlap on $k-1$ characters with their predecessors), and
- $a_{i+1}[k-1] = b_{i+1}[k-1]$ (the extension is done with a same character), and
- $diff(a_{i+1}) = diff(a_{op}) \pm \delta$ and $diff(b_{i+1}) = -diff(a_{op}) \pm \delta$ (the counting model is fulfilled).

Similar conditions apply to a_{i-1} and b_{i-1} for extending a_i and b_i on the left.

The two lips of a mouth have to be closed. A mouth can be right-closed (resp. left-closed) if there exists a k-mer c_k (resp. $c_{-1)}$) whose prefix (resp. suffix) of length $k-1$ is equal to the suffix (resp. prefix) of length $k-1$ of a_{k-1} (resp. a_0), by definition itself equal to the suffix (resp. prefix) of length $k-1$ of b_{k-1} (resp. b_0). Once the mouth is right- and left- closed, the procedure stops.

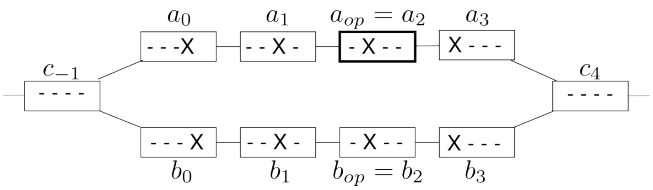


Fig. 2. A mouth with $k = 4$. The symbol $'-'$ stands for a match between two k-mers positions while the symbol $'X'$ stands for a mismatch. The k-mer a_2 is the opening k-mer and the k-mer b_2 is the opposite opening k-mer. Here, $p = 1$ as $a_{op}[1] \neq b_{op}[1]$.

Disabling the use of a same opening k-mer more than once Once a k-mer a_{op} was used for opening a mouth, it is flagged and never used anymore either as an opening or as an extending k-mer. The underlying idea is to avoid detecting twice the same mouth. Moreover, we can safely discard this k-mer because, since it was the one with smallest count, it should not belong to another mouth.

4.2 Complexity

The time complexity can then be divided into two main parts as follows.

- **Indexation:** if N is the total number of distinct k-mers, indexing them into a tree can be done in $O(N \log N)$ time using heap sort. This then provides access in time $O(k)$ to any k-mer information.
- **Mouth creation:** Given one opening k-mer a_{op}, at worst $k \times (|\Sigma| - 1)$ k-mers b_{op} may fulfill the opening conditions. Any k-mer may be opening. Thus at worst, $O(N \times k \times |\Sigma|)$ mouth opening pairs must be tried. Given an opening pair of k-mers a_{op} and b_{op}, in the worst case, for each of the $k-1$ steps of the extension, $|\Sigma|$ k-mers muts be tested. Access to a k-mer information being in time $O(k)$, the time complexity for one mouth extension is thus $O(k|\Sigma|^k)$. Thus, at worst, the time complexity for mouth finding is $O\left(N \times k^2 \times |\Sigma|^{2k}\right)$.

Each k-mer being stored in $O(k)$, the space complexity is $O(Nk)$.

4.3 Checking for read coherence

Two k-mers are linked in the de Bruijn graph if they overlap over k-1 characters, without checking whether the created k+1-mer indeed exists in the set of reads. This may lead to false-positive results. In order to remove those k-mers, in a post-treatment step, we check for *read-coherency* of the identified mouths. The upper (resp. lower) lip of a mouth is said to be read-coherent if it is 100% covered by reads from set A (resp. B) and, moreover, if in the upper (resp. lower) lip the SNP position is covered by at least two distinct reads from set A (resp. B). We keep only the mouths for which both lips are read-coherent. The rationale for restricting to mouths covered by at least 2 reads is to minimize the number of sequencing errors that are mistaken for SNPs. Indeed, it is unlikely that a sequencing error occurs at the same nucleotide for 2 distinct reads, as shown in [9].

5 Applications to simulated read data

We developed the algorithm in a program called kisSnp, coded in Java, that was used for testing our approach. kisSnp is available for download at: http://alcovna.genouest.org/kissnp/.

To test our approach on controlled datasets, we applied the following procedure. Given an input sequence s_{ref}, we generated a sequence s_{snp} such that

s_{ref} and s_{snp} differ by $\left\lfloor \frac{|s_{ref}|}{1000} \right\rfloor$ substitutions, each considered as a SNP. The average distance between two virtual SNPs is then 1000 nucleotides, in agreement with [4]. The substitutions are randomly distributed over s_{snp}, and each substitution is introduced following the transition/transversion probabilistic model [5].

The sequence s_{ref} and s_{snp} are then virtually sequenced into a set of reads r_{ref} and r_{snp} using the MetaSim [10] program. Among other parameters, MetaSim enables to tune the sequencing errors model as well as the average coverage. In all our experiments, we generated reads of length 62, in agreement with the Illumina technology. kisSnp is then tested using sets r_{ref} and r_{snp}.

5.1 Finding the SNPs

Human chrX portion We extensively tested the parameters of kisSnp on a small portion of the human chromosome X of length 137897 *bp*, corresponding to a kinesin family member 4A (KIF4A). We applied MetaSim to the pair s_{ref}, s_{snp} distant by 137 simulated SNPs with i) no sequencing errors (Fig. 3(a)) and ii) errors following an empirical Illumina error model (Jean Marc Aury, Genoscope, personal communication – Fig. 3(b) and (c)).

One may start by observing that the quality of the results is relatively robust to the choice of parameters. With a good coverage ($> 4x$), large distinct sets of parameters thus enable to find almost all SNPs with no false positives. The main lessons learnt about such parameters from these results are the following:

- The δ value has not a strong influence for $\delta \geq 20$ (Fig. 3(b) vs Fig. 3(c)). However, for smaller values of δ (data not shown), the specificity decreases rapidly due to sequencing errors and a non uniform coverage.
- As expected, for small values of k (≤ 20), false positives are found. For larger values of k (say, ≥ 30), less SNPs are found as more k-mers involve sequencing errors and/or more positions are not covered by any k-mer.

Concerning the data, coverage is of major importance as a small coverage leads to lower sensitivity (in each case, the lower the coverage, the less sensitive is kisSnp). Illumina sequencing errors have a small influence on the results (Fig. 3(a) vs Fig. 3(b)). One important message is that, using $k = 25$ and $\delta \geq 20$, all experiments obtained 100% specificity (no false positives) for various values of sensitivity, the latter depending in particular on the coverage.

Neisseria meningitidis strain MC58 One main limitation of our mouth model could be the presence of repeats in the genomes considered. To measure the effect of repeats on the performance of kisSnp, we performed experiments on the bacterium *Neisseria meningitidis* (strain MC58) of length 2.27 Mbp. The size of the repeated elements in this genome range from 10 bases to more than 2000 bases, and their number may reach more than 200 copies. We performed tests on the original MC58 sequence introducing 2272 SNPs and simulating reads using MetaSim with an Illumina error model. Moreover, we performed exactly the same tests on a randomised sequence obtained from the MC58 sequence

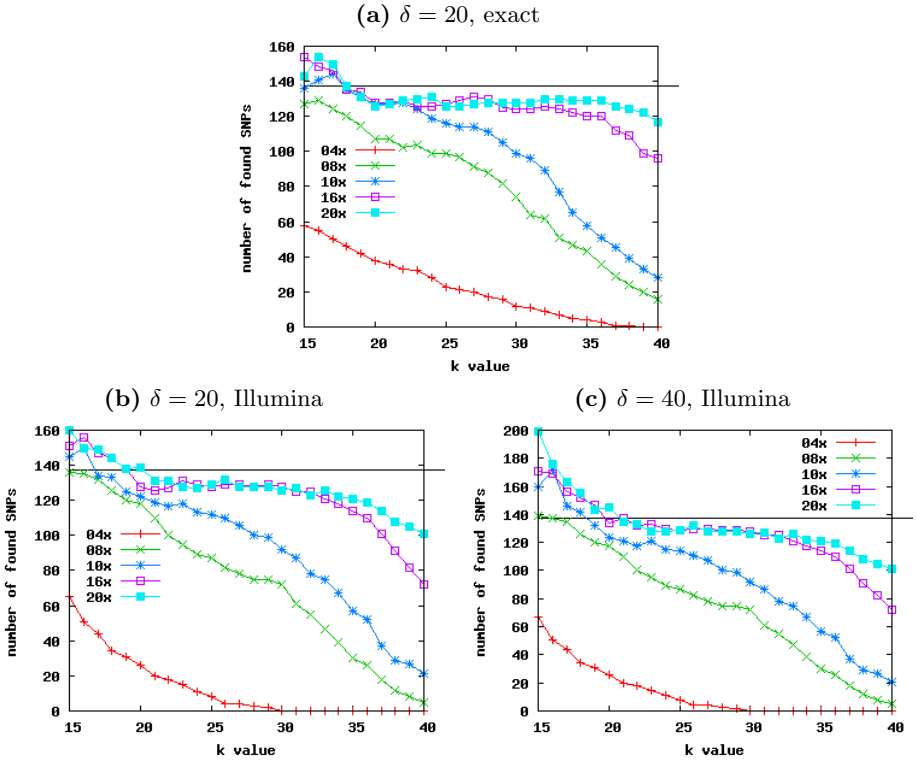


Fig. 3. Number of SNPs (read-coherent mouths) found by kisSnp. Fragment of the human chromosome X, 137 SNPs to find (symbolised by the horizontal line). The "$n\times$" values indicate the coverage used while simulating the reads.

(same length and nucleotide frequencies, distribution of nucleotides following a Bernoulli model). Results for both MC58 and the randomised MC58 are presented in Fig. 4.

One may observe that even on a difficult dataset like MC58, a large part of the SNPs are identified (26%, 86% and 97% respectively with coverage 4x, 10x, and 20x with $k = 20$). Another important remark is that the difference between the results obtained on MC58 and the randomised MC58 is small, showing that the algorithm is robust to repeats.

5.2 Execution time

The kisSnp program, coded in Java, is a prototype and is not yet time optimised. The performance results below enable only to give an idea of the evolution of the running time with different parameters. All tests were done on a DELL laptop, quad-core 2.67GHz with 4Gb memory running under Fedora Core 12.

We started by testing on the human KIF4A dataset (simulating reads with an Illumina error model), the influence of the δ parameter on the execution time. We observed (data not shown) that this has no influence whatsoever for $\delta \geq 20$.

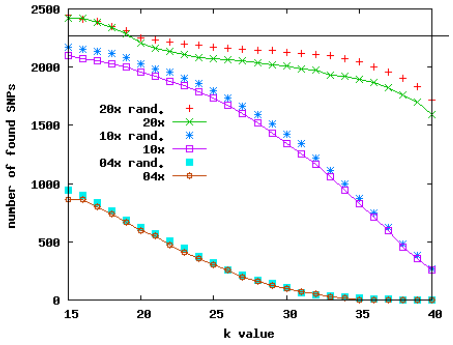


Fig. 4. Results on MC58 and the randomised MC58 sets while looking for 2272 SNPs (horizontal line) with $\delta = 20$. The "$n\times$" values indicate the coverage used while simulating the reads, "*rand.*" stands for results on the randomised MC58 set.

On the same dataset, we fixed $\delta = 30$ and checked the execution time for values of k varying from 0 to 40. The results, presented in Fig. 5, show that two phases may be distinguished. For k from 0 to 6, we observe an exponential time growth that is in agreement with the theoretical complexity. During this phase, the worst-case behaviour is reached, each mouth extension tests $|\Sigma|^k$ lips. The second phase is observed for bigger values of k, when a large number of possible k-mers are no longer present in the data, thus limiting the number of tested lips extensions. This greatly reduces the execution time, which starts decreasing for $k \geq 25$ as less and less mouths are successfully created.

The execution time also highly depends on the number N of distinct k-mers we have to deal with. We thus performed experiments on random sequences of growing size. The results are presented in Fig. 6. The execution time grows linearly with N while N remains below a threshold of around 15 million reads. Above this threshold, one observes an exponential growth that could be explained by the fact that the kisSnp prototype uses a hash table instead of a tree for storing and accessing the k-mers information. With a large number of k-mers, the hash table load factor becomes higher than 0.75 increasing the lookup cost, because of an important number of collisions.

6 Applications to real read data

To test our approach on a real dataset now, we used raw reads from the *Escherichia coli Long-term Experimental Evolution Project* [1] whose purpose was to grow *Escherichia coli* during more than 20 years, conserving a sample each 500 generations. The SNPs found over these generations are listed in the Barrick *et. al* paper [2]. An Illumina 1G platform was used for sequencing the samples with reads of length 36 and a high coverage ($50x$). We focused our attention on the raw reads from the first generation sample, and those from the

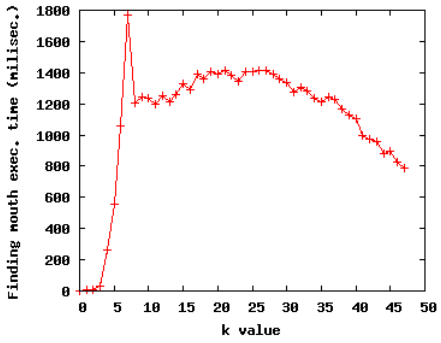


Fig. 5. Influence of k on the execution time. Data: set KIF4A (described Section 5.1), $\delta = 30$.

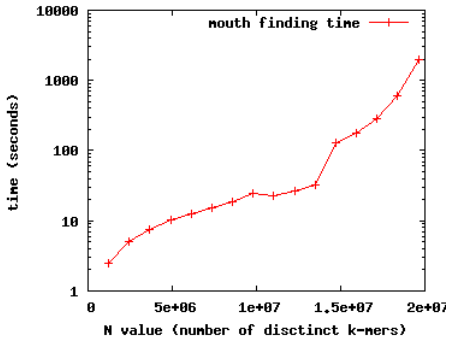


Fig. 6. Execution time with respect to the number N of distinct k-mers of length $k = 25$, with $\delta = 20$, Bernoulli random sequences.

20.000^{th} generation sample. The existence of experimentally validated SNPs is very rare which is the main reason that led us to work on this dataset, for which true positives are known.

Using a custom-made computational pipeline called BRESEQ, Barrick *et. al* identified 28 SNPs by mapping these two generations of reads against the reference genome CP000819. These 28 SNPs were then experimentally validated.

We used kisSnp for comparing these two sets of reads, forgetting the reference genome. Using as parameters $k = 26$ and $\delta = 20$, 88 SNPs were found. Of these, 27 of the 28 SNPs found by Barrick *et. al* were also identified by us, giving a sensitivity of 96%. Our kisSnp method missed one SNP, located position 430835.

To evaluate the potential interest of the remaining 61 SNPs we identified, we mapped them against the reference genome using Maq [6]. Among the 61, 43 correspond to a SNP structure not detected in the Barrick *et. al* project: the two lips map at the same position with one substitution in the 20.000^{th} generation sequence. The remaining 18 correspond to suspicious SNPs. Indeed, the two lips do not map to the same position in the genome. Without experimental validation, one however cannot conclude on those 61 detected putative SNPs.

The results obtained with kisSnp are very good on this real dataset, as without a reference genome it was able to find back 27 of the 28 experimentally verified SNPs, and 41 additional ones that correspond to real SNP structures.

7 Conclusion and future work

We proposed an algorithm for comparing the raw outputs of short reads experiments, typically Illumina ones, with the purpose of finding SNPs between individuals of a same species. This is of particular interest for quickly designing genomic tags without waiting and/or paying for a full genome assembly.

Preliminary results on both simulated and real experimental data are particularly promising. In both cases, kisSnp identifies the SNPs, while not being too

sensitive to the parameters used. On a real dataset, kisSnp enabled to find 96% of the SNPs initially detected by mapping to the reference genome. In addition, we propose new SNPs, which could be tested experimentally.

There is clearly room for improvement. For now, the method does not handle heterozygous SNPs, does not take sequencing qualities into account and is limited to pairwise comparison while sets of more than two individuals may be compared. More generally, the three challenges of SNP identification are that the reads contain errors, the genome contains repeats and the read coverage is not uniform. This last item is usually disregarded whereas we notice that it has a significant impact on the results. We expect that several algorithms in the area of NGS bioinformatics could be improved by taking this observation into account.

Acknowledgments

We are grateful to Jean Marc Aury, who provided the empirical Illumina error model, to Matteo Brilli who pointed out to us the *E.coli* experiment, to the GenOuest platform who supported extensive computations and to the INRIA ARC Alcovna for financial support.

References

1. E. coli long-term experimental evolution project site, http://myxo.css.msu.edu/ecoli/.
2. J. E. Barrick, D. S. Yu, H. Jeong, T. K. Oh, D. Schneider, R. E. Lenski, and J. F. Kim. Genome evolution and adaptation in a long-term experiment with escherichia coli. *Nature*, 461:1243–1247, 2009.
3. C. Cannon, C.-S. Kua, D. Zhang, and J. Harting. Assembly free comparative genomics of short-read sequence data discovers the needles in the haystack. *Molecular Ecology*, 19(Suppl. 1):147–161, 2010.
4. D. N. Cooper, B. A. Smith, H. J. Cooke, S. Niemann, and J. Schmidtke. An estimate of unique DNA sequence heterozygosity in the human genome. *Hum. Genet.*, 69:201–205, 1985.
5. M. Kimura. A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. *J. Mol. Evol.*, 16:111–120, Dec 1980.
6. H. Li, J. Ruan, and R. Durbin. Mapping short DNA sequencing reads and calling variants using mapping quality scores. *Genome Research*, 18(11):1851–1858, 2008.
7. R. Li, H. Zhu, J. Ruan, W. Qian, X. Fang, Z. Shi, Y. Li, S. Li, G. Shan, K. Kristiansen, S. Li, H. Yang, J. Wang, and J. Wang. De novo assembly of human genomes with massively parallel short read sequencing. *Genome Res.*, 20(2):265–272, 2010.
8. P. Pevzner, H. Tang, and M. Waterman. An Eulerian path approach to DNA fragment assembly. *Proc. Natl. Acad. Sci.*, 98:9748–9753, 2001.
9. A. Ratan, Y. Zhang, V. Hayes, S. Schuster, and W. Miller. Calling SNPs without a reference genome. *BMC Bioinformatics*, 11:130, 2010.
10. D. Richter, F. Ott, A. Auch, R. Schmid, and D. Huson. MetaSim – A Sequencing Simulator for Genomics and Metagenomics. *PLoS ONE*, 3(10):e3373, 2008.
11. D. Zerbino and E. Birney. Velvet: Algorithms for de novo short read assembly using de Bruijn graphs. *Genome Res.*, 18(5):821–829, 2008.

Kissplice : de-novo calling alternative splicing events from rna-seq data

Sacomoto et al. BMC Bioinformatics 2012, 13(Suppl 6):S5
http://www.biomedcentral.com/1471-2105/13/S6/S5

BMC Bioinformatics

PROCEEDINGS Open Access

KISSPLICE: de-novo calling alternative splicing events from RNA-seq data

Gustavo AT Sacomoto[1,2], Janice Kielbassa[1,2], Rayan Chikhi[3], Raluca Uricaru[3,4], Pavlos Antoniou[3], Marie-France Sagot[1,2], Pierre Peterlongo[3*], Vincent Lacroix[1,2*]

From Second Annual RECOMB Satellite Workshop on Massively Parallel Sequencing
Barcelona, Spain. 19-20 April 2012

Abstract

Background: In this paper, we address the problem of identifying and quantifying polymorphisms in RNA-seq data when no reference genome is available, without assembling the full transcripts. Based on the fundamental idea that each polymorphism corresponds to a recognisable pattern in a De Bruijn graph constructed from the RNA-seq reads, we propose a general model for all polymorphisms in such graphs. We then introduce an exact algorithm, called KISSPLICE, to extract alternative splicing events.

Results: We show that KISSPLICE enables to identify more correct events than general purpose transcriptome assemblers. Additionally, on a 71 M reads dataset from human brain and liver tissues, KISSPLICE identified 3497 alternative splicing events, out of which 56% are not present in the annotations, which confirms recent estimates showing that the complexity of alternative splicing has been largely underestimated so far.

Conclusions: We propose new models and algorithms for the detection of polymorphism in RNA-seq data. This opens the way to a new kind of studies on large HTS RNA-seq datasets, where the focus is not the global reconstruction of full-length transcripts, but local assembly of polymorphic regions. KISSPLICE is available for download at http://alcovna.genouest.org/kissplice/.

Background

Thanks to recent technological advances, sequencing is no longer restricted to genomes and can now be applied to many new areas, including the study of gene expression and splicing. The so-called RNA-seq protocol consists in applying fragmentation and reverse transcription to a RNA sample followed by sequencing the ends of the resulting cDNA fragments. The short sequencing reads then need to be reassembled in order to get back to the initial RNA molecules. A lot of effort has been put on this assembly task [2], whether in the presence or in the absence of a reference genome but the general goal of identifying and quantifying all RNA molecules initially present in the sample remains hard to reach. The main challenge is certainly that reads are short, and can therefore be ambiguously assigned to multiple transcripts. In particular, in the case of alternative splicing (AS for short), reads stemming from constitutive exons can be assigned to any alternative transcript containing this exon. Finding the correct transcript is often not possible given the data we have, and any choice will be arguable. As pointed out in Martin and Wang's review [2], reference-based and de novo assemblers each have their own limitations. Reference-based assemblers depend on the quality of the reference while only a small number of species currently have a high-quality reference genome available. De novo assemblers implement reconstruction heuristics which may lead them to miss infrequent alternative transcripts while highly similar transcripts are likely to be assembled into a single transcript. We argue here that it is not always necessary to aim at the difficult goal of assembling full-length molecules. Instead, identifying the variable parts

* Correspondence: Pierre.Peterlongo@inria.fr; Vincent.Lacroix@univ-lyon1.fr
[1]INRIA Grenoble Rhône-Alpes, France
[3]Centre de recherche INRIA Rennes - Bretagne Atlantique, IRISA, Campus universitaire de Beaulieu, Rennes, France
Full list of author information is available at the end of the article

between molecules (polymorphic regions) is already very valuable and does not require to solve the problem of assigning a constitutive read to the correct transcript. We therefore focus in this paper on the simpler task of identifying polymorphisms in RNA-seq data. Three kinds of polymorphisms have to be considered: i) AS (alternative splicing) that produces several alternative transcripts for a same gene, ii) SNPs (single nucleotide polymorphism) that may also produce several transcripts for a same gene whenever they affect transcribed regions, and iii) approximate tandem repeats which affect the number of copies of tandem repeats. Our contribution in this paper is double: we first give a general model which captures these three types of polymorphism by linking them to characteristic structural patterns called "bubbles" in the De Bruijn graph (DBG for short) built from a set of RNA-seq reads, and second, we propose a method dedicated to the problem of identifying AS events in a DBG, including read-coverage quantification. We notice here that only splicing events but not transcriptional events, such as alternative start and polyadenylation sites, are covered by our method.

The identification of bubbles or bulges in DBG has been studied before in the context of genome assembly [3-5]. However, the purpose in these works was not to enumerate these patterns, but "only" to remove them from the graph. Additionally, since in these applications, the patterns correspond to SNPs and sequencing errors, the authors only considered paths of length smaller than a constant.

More recently, ad-hoc enumeration methods have been proposed but are restricted to non-branching bubbles [6], *i.e.*, each vertex from the bubble has in-degree and out-degree 1, except for the extremities of the bubble.

Extracting AS events from a splicing graph has been studied before [7] but a significant difference between splicing graphs and De Bruijn graphs is that in the former, nodes are genomically ordered (through the use of a reference annotated genome) therefore leading to a DAG, whereas DBGs are general graphs, that furthermore do not require any additional information to be built.

When no reference genome is available, efforts have focused on assembling the full-length RNA molecules, not the variable parts which are our interest here. Most RNA-seq assemblers [8-10] do rely on the use of a DBG, but, since the primary goal of an assembler is to produce the longest contigs, heuristics are applied, such as tip or bubble removal, in order to linearise the graph. The application of such heuristics results in a loss of information which may in fact be crucial if the goal is to study polymorphism.

To our knowledge, this work is the first attempt to characterise polymorphism in RNA-seq data without assembling full-length transcripts. We stress that it is not a general purpose transcriptome assembler and when we benchmark it against such methods, we only focus on the specific task of AS event calling. Finally, our method can be used with a single or multiple RNA-seq experiments and our quantification module outputs a coverage (reads per nt) for both the shorter and the longer isoform(s) of each AS event, in each experiment.

The paper is organised as follows. We first present the model (Section "*De-Bruijn graph models*") linking structures of the DBG for a set of RNA-seq reads to polymorphism, and then introduce a method, that we call KISSPLICE, for identifying DBG structures associated with AS events (Section "*The* KISSPLICE*algorithm*"). We show in Section "*Results*" the results of using KISSPLICE compared with other methods on simulated and real data.

Methods

De-Bruijn graph models

De-Bruijn graph

DBGs were first used in the context of genome assembly in 2001 by Pevzner *et al.* [11]. In 2007, Medvedev *et al.* [12] modified the definition to better model DNA as a double stranded molecule. In such a context, a DBG is a bidirected multigraph, each node N storing a sequence w and its reverse complement $\bar{w}$. The sequence w, denoted by $F(N)$, is the forward sequence of N, while $\bar{w}$, denoted by $R(N)$, is the reverse complement sequence of N. An arc exists from node N_1 to node N_2 if the suffix of length k - 1 of $F(N_1)$ or $R(N_1)$ overlaps perfectly with the prefix of $F(N_2)$ or $R(N_2)$. Each arc is labelled with a string in {*FF, RR, FR, RF*}. The first letter of the arc label indicates which of $F(N_1)$ or $R(N_1)$ overlaps $F(N_2)$ or $R(N_2)$, this latter choice being indicated by the second letter. Because of reverse complements, there is an even number of arcs in the DBG: if there is an arc from N_1 to N_2 then, necessarily, there is an arc from N_2 to N_1 (*e.g.* if the first arc has label *FF* then the second has label *RR*). Examples of DBGs are presented in Figure 1.

Definition 1 (Valid path) *The traversal of a node is said to be valid if the rightmost label (F or R) of the arc entering the node is equal to the leftmost label of the arc leaving the node.*

A path in the graph is valid if for each node involved in the path, its traversal is valid, that is, each pair of adjacent arcs in the path are labelled, respectively, XY and Y Z with X, Y, Z $\in$ *{R, F}.*

For instance, for any graph shown in Figure 1, the path starting from the leftmost encircled node, going by the upper path to the rightmost encircled node is valid.

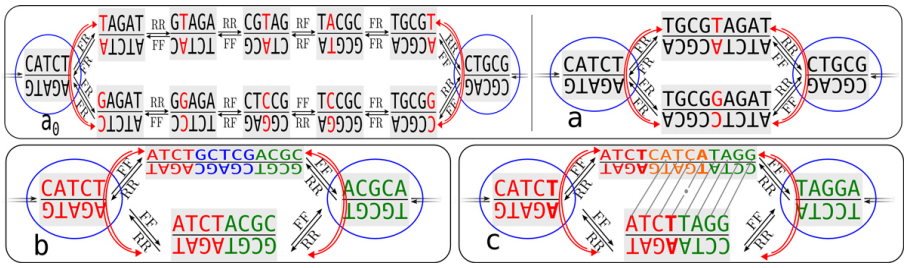


Figure 1 De Bruin graphs. Part of non-compressed **(a_0)** and compressed **(a, b, c)** de Bruijn graphs (k = 5). Each node contains a word (upper text of each node) and its reverse complement (lower text of each node). In the uncompressed graph, the word is a k-mer. Encircled nodes are switching with respect to red paths (pointed out by red arrows). **(a_0, a)** Bubble due to a substitution (red letter). Starting from the forward strand in the leftmost (switching) node would generate the sequences CATCT A CGCAG (upper path) and CATCT C CGCAG (lower path). **(b)** Bubble due to the skipped exon GCTCG (blue sequence). This bubble is generated by the sequences CATCT ACGCA and CATCT GCTCG ACGCA. **(c)** Bubble due to an inexact tandem repeat. This bubble is generated by the sequences CATCT TAGGA and CATCT CATCA TAGGA, where CATC**T** CATC**A** is an inexact tandem repeat.

A DBG can be compressed without loss of information by merging simple nodes. A *simple* node denotes a node linked to at most two other nodes. Two adjacent simple nodes are merged into one by removing the redundant information. A valid path composed by $i > 1$ simple nodes is compressed into one node storing a sequence of length $k + (i - 1)$ as each node adds one new character to the first node. Figure 1.a represents the compressed DBG shown in Figure 1.a0. In the remaining of the paper, we denote by cDBG a compressed DBG.

Bubble patterns in the cDBG

Polymorphisms (*i.e.* variable parts) in a transcriptome or a genome, correspond to recognisable patterns in the cDBG, which we call a **bubble**. Intuitively, the variable parts will correspond to alternative paths and the common parts will correspond to the beginning and end points of these paths. We now formally define the notion of bubble, taking carefully into account the bi-directed and arc labeled nature of the cDBG.

Definition 2 (Node switching with respect to a path) *A node is switching with respect to a path if this path is invalid during the traversal of this node.*

Definition 3 (Bubble) *In the cDBG, a bubble is a simple cycle involving at least three distinct nodes such that exactly two nodes SN_{left} and SN_{right} are switching w.r.t. the path of the cycle. By definition, two valid paths exist between these two switching nodes. In the remaining of the paper, we refer to these two paths as the paths of the bubble. If they differ in length, we refer to, respectively, the longer and the shorter path of the bubble.*

Figure 1 presents four bubbles. For each one, their switching nodes are encircled in blue.

In general, any process generating patterns *asb* and *as'b* in the sequences, with $a, b, s, s' \in \Sigma^*$, $|a| \geq k$, $|b| \geq k$ and s and s' not sharing any k-mer, creates a bubble in the cDBG. Indeed, all k-mers entirely contained in a (resp. b) compose the node SN_{left} (resp. SN_{right}). Since $|a| \geq k$ and $s \neq s'$, there is at least one pair of k-mers, one in as and the other in *as'*, sharing the k - 1 prefix and differing by the last letter, thus creating a branch in SN_{left} from which the two paths in the bubble diverge. The same applies for *sb*, *s'b* and SN_{right}, where the paths merge again. All k-mers contained in s (resp. s') and in the junctions *as* and *sb* (resp. *as'* and *s'b*) compose the paths of the bubble. In the case of AS events and approximate tandem repeats, s is empty and the shorter path is composed of k-mers covering the junction *ab*.

This model is general as it captures SNPs, approximate tandem repeats and AS events, as shown in Figure 1. The main focus of the algorithm we present in this paper is the detection of bubbles generated by AS events.

Bubbles generated by AS events

A single gene may give rise to multiple alternative spliceforms through the process of AS. Alternative spliceforms differ locally from each other by the inclusion or exclusion of subsequences. These subsequences may correspond to exons (exon skipping), exon fragments (alternative donor or acceptor sites) or introns (intron retention) as shown in Figure 2. Observe that alternative start and polyadenylation sites, which are not considered as AS events but as transcriptional events, are not taken into account in this work. A splicing event corresponds to a local variation between two alternative transcripts. It is characterised by two common sites (called a and b in the examples given in Figure 2) and a variable part (called s Figure 2). In the cDBG, the common sites correspond to the switching nodes and the variable part to the longer path. As there are k - 1 k-mers at the junction between the two common sites $a.b$, the shorter path is composed of at most k - 1 k-mers, *i.e.* represents a path of length at most $2k$ - 2 in the cDBG. An example is given in Figure 1.b.

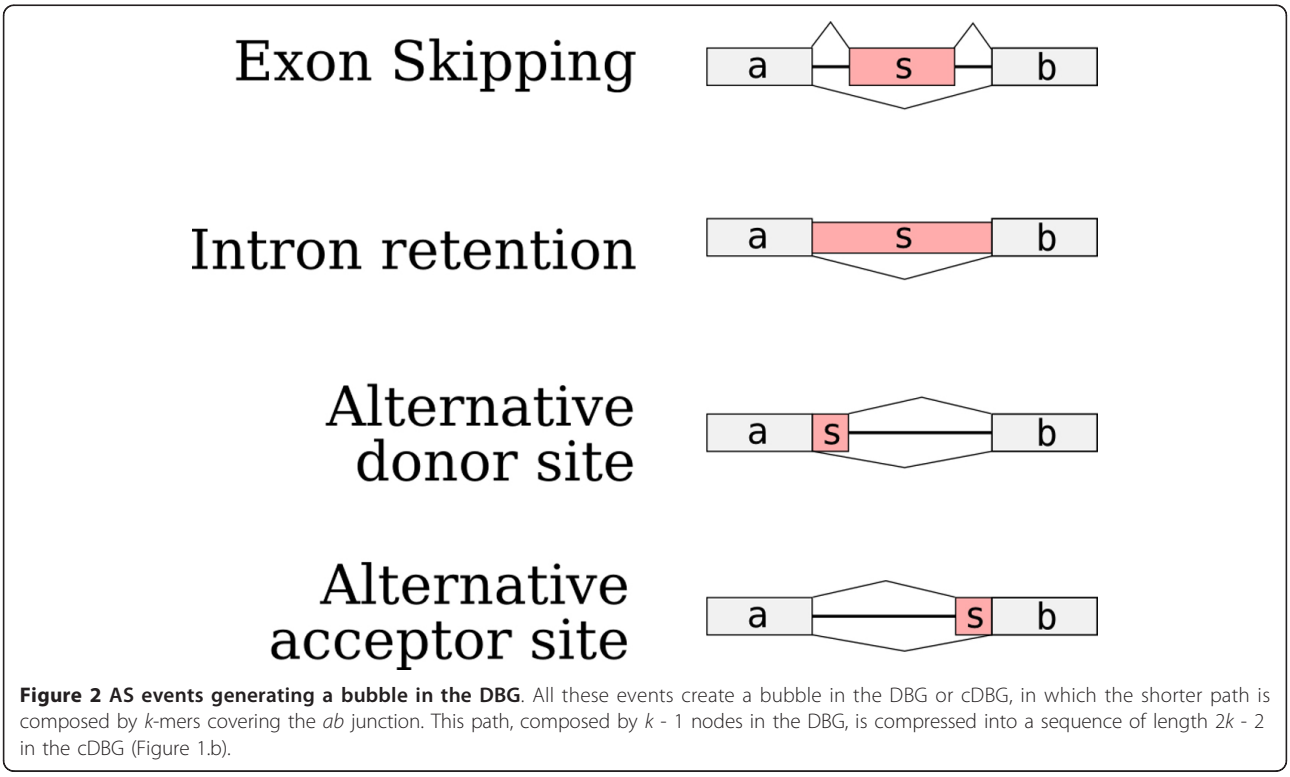


Figure 2 AS events generating a bubble in the DBG. All these events create a bubble in the DBG or cDBG, in which the shorter path is composed by *k*-mers covering the *ab* junction. This path, composed by *k* - 1 nodes in the DBG, is compressed into a sequence of length 2*k* - 2 in the cDBG (Figure 1.b).

The shorter path of a bubble generated by an AS event has length exactly 2*k* - 2 iff (i) the last nucleotide (nt for short) of the variable part is distinct from the last nt of the left switching node, and (ii) the first nt of the variable part is distinct from the first nt of the right switching node. Otherwise, the two alternative paths join (case (i)) or diverge (case (ii)) earlier and the shorter path may be smaller. In human, 99% of the annotated exon skipping events yield a bubble with a shorter path length between 2*k* - 8 and 2*k* - 2.

Bubbles generated by SNPs and approximate tandem repeats

Polymorphism at the genomic level will necessarily also be present at the transcriptomic level whenever it affects transcribed regions. Two major kinds of polymorphism can be observed at the genomic level: SNPs and approximate tandem repeats. As shown in Figure 1, these two types of polymorphism also generate bubbles in the cDBG.

However, these bubbles have characteristics which enable to differentiate them from bubbles generated by AS events. Indeed, bubbles generated by SNPs exhibit two paths of length exactly 2*k* - 1, which is larger than 2*k* - 2, the maximum size of the shorter path in a bubble generated by an AS event.

Approximate tandem repeats may generate bubbles with a similar path length as bubbles generated by splicing events, but the sequences of the paths exhibit a clear pattern which can be easily identified: the longer path contains an inexact repeat. More precisely, as outlined in Figure 1.c, it is sufficient to compare the shorter path with one of the ends of the longer path.

Finally, genomic insertions or deletions (indels for short) may also generate bubbles with similar path lengths as bubbles generated by splicing events. In this case, the difference of length between the two paths is usually smaller (less than 3 nt for 85% of indels in human transcribed regions [13] whereas it is more than 3 nt for 99% of AS events). In our method, when the difference of path lengths is strictly below 3, we classify the bubble as an indel. Otherwise, we do not decide, which means that a fraction of the bubbles we report as AS events will correspond to indels. Note that this classification is a simple suggestion. We encourage users to affine results by considering species specificity and by applying coverage criterion.

In the following, we focus on bubbles generated by AS events. We do provide as a collateral result three additional collections of bubbles: one corresponding to putative SNPs, one to short indels, and one to putative approximate tandem repeats. The post-treatment of these collections to discard false positives caused by sequencing errors is beyond the scope of this paper.

The KISSPLICE algorithm

The KISSPLICE algorithm detects in the cDBG all the bubble patterns generated by AS events, *i.e.* the bubbles

having a shorter path of length at most $2k$ - 2. Essentially, the algorithm enumerates all the cycles verifying the two following criteria: i) the path obtained by following all the nodes of the cycle contains exactly two nodes that are switching for this path, and ii) the length of the shorter path linking the two switching nodes must be no longer than $2k$ - 2. Further criteria are applied to make the algorithm more efficient without loss of information, and to eliminate polymorphism events that do not correspond to AS. Since the number of cycles in a graph may be exponential with the size of the graph, the naive approach of enumerating all cycles of the cDBG and verifying which of them satisfy our conditions is only viable for very small cases.

Nonetheless, KISSPLICE is able to enumerate a potentially exponential number of bubbles for real-sized data in very reasonable time and memory. This is in part due to the fact that, previous to cycle enumeration, the graph is pre-processed in a way that, along with the pruning criteria of Step 4 (see below), is responsible for a good performance in practice.

KISSPLICE is indeed composed of six main steps which are described next. The pre-processing just mentioned corresponds to Step 2. As far as we know, it is the first time it is used in conjunction with cycle enumeration.

Step 1. Construction of the cDBG of the reads of one or several RNA-seq experiments. Each node contains the coverage of the corresponding k-mer in each experiment. In order to get rid of most of the sequencing errors, nodes with a minimal coverage of 1 may be removed.

Step 2. Biconnected component (BCC for short) decomposition. A connected undirected graph is *biconnected* if it remains connected after the removal of any vertex. A BCC of an undirected graph is a maximal biconnected subgraph. Moreover, it is possible to show that the BCCs of an undirected graph form a partition of the edges with two important properties: every cycle is contained in exactly one BCC, and every edge not contained in a cycle forms a singleton BCC. Applying on the underlying undirected graph of the cDBG Tarjan's lowpoint method [14] which performs a modified depth-first search traversal of the graph, Step 2 detects all BCCs, and discards all singleton ones that could not contain any bubble. Without modifying the results, this considerably reduces the memory footprint and the computation time of the whole process. To give an idea of the effectiveness of this step, the cDBG of a 5 M dataset had 1.7 M nodes, but the largest BCC only 2961 nodes.

Step 3. Four-nodes compression. Single substitution events (SNPs, sequencing errors) generate a large number of cycles themselves included into bigger ones, creating a combinatorial explosion of the number of possible bubbles. This step of KISSPLICE detects and compresses all bubbles composed by just four nodes: two switching nodes and two *non-branching internal nodes* each storing equal length sequences differing by just one position. Figure 1.a shows an example of a four-nodes bubble. Four-nodes bubbles are output as potential SNPs and then reduced to a three-nodes path. The two non-branching internal nodes are merged into one, storing a consensus sequence where the unique substitution is replaced by N.

Step 4. Bubbles enumeration. The cycles are detected in the cDBG using a backtracking procedure [15] augmented with two pruning criteria. The exploration of one cycle is stopped if the path contains more than two nodes that are switching relative to the path that is being followed, or the length of the shorter path is bigger than $2k$ - 2. This approach has the same theoretical time complexity of Tiernan's algorithm for cycle enumeration [15], which is worse than Tarjan's [16] polynomial delay algorithm but it appears to be not immediate how to use the pruning criteria with the latter while preserving its theoretical complexity. We however were able to show that in practice, the pruning criteria are very effective for the type of instances we are dealing with. Indeed, we compared the three following implementations on a 1 M reads dataset: i) Tiernan ii) Tarjan iii) Tiernan with prunings (our method). The results clearly showed that, while Tarjan (22 min) outperforms Tiernan (32 min), both are clearly outperformed when the prunings are used (4 s).

Step 5. Results filtration and classification. The two paths of each bubble are aligned. If the whole of the shorter path aligns with high similarity to the longer path, we decide that the bubble is due to an approximate tandem repeat (see Section "*Bubbles generated by SNPs and approximate tandem repeats*"). After this alignment, a bubble is classified either as an AS event, an approximate tandem repeat, or a small indel (less than 3 nt).

Step 6. Read coherence and coverage computation. Reads from each input dataset are mapped to each path of the bubble. If at least one nucleotide of a path is covered by no read, the bubble is said to be not read-coherent and is discarded. The coverage of each position of the bubble corresponds to the number of reads overlapping this position. Border effects are handled in the following way: reads mapping to the extremity of a path with less than k bases are discarded. This results in a systematic under-estimation of the coverage of the extremities of the path. Under a simple assumption of locally uniform coverage, this can be counter-balanced by multiplying the coverage of each of the k - 1 external positions by a correction factor of $\frac{L}{L-i}$, with L the read

size and i the distance to the first non biased position. This correction is possible because the paths considered correspond to internal transcript sequences, not to a transcription start or end.

Results

Simulated data

In order to assess the sensitivity and specificity of our approach, we simulated the sequencing of genes for which we are able to control the number of alternative transcripts. We show that the method is indeed able to recover AS events whenever the alternative transcripts are sufficiently expressed. For our sensitivity tests, we used simulated RNA-seq single end reads (75 bp) with sequencing errors. We first tested a pair of transcripts with a 200 nt skipped exon. Simulated reads were obtained with MetaSim [17] which is a reference software for simulating sequencing experiments. As in real experiments, it produces heterogeneous coverage and authorises to use realistic error models.

In order to find the minimum coverage for which we are able to work, we created datasets for several coverages (from 4× to 20×, which corresponds to 60 to 300 Reads Per Kilobase or RPK for short), with 3 repetitions for each coverage, and tested them with different values of k (k = 13, ...41). The purpose of using 3 repetitions for each coverage was to obtain results which did not depend on irreproducible coverage biases. For coverages below 8× (120 RPK), KISSPLICE found the correct event in some but not all of the 3 tested samples. The failure to detect the event was due to the heterogenous and thus locally very low coverage around the skipped exon, *e.g.* some nt were not covered by any read or the overlap between the reads was smaller than k-1. Above 8× (120 RPK), KISSPLICE detected the correct exon skipping event in all samples.

For each successful test, there was a maximal value k_{max} for k above which the event was not found, and a minimal value k_{min} below which KISSPLICE also reported false positive events. Indeed, if k is too small, then the pattern *ab, as 'b*, with $|a| \geq k$, $|b| \geq k$ is more likely to occur by chance in the transcripts, therefore generating a bubble in the DBG. Between these two thresholds, KISSPLICE found only one event: the correct one. The values of k_{min} and k_{max} are clearly dependent on the coverage of the gene. At 8× (120 RPK), the 200 nucleotides exon was found between k_{min} = 17 and k_{max} = 29. At 20× (300 RPK), it was found for k_{min} = 17 and k_{max} = 39. We performed similar tests on other datasets, varying the length of the skipped exon. As expected, if the skipped exon is shorter (longer), KISSPLICE needed a lower (higher) coverage to recover it.

Since KISSPLICE is, to our knowledge, the first method able to call AS events without a reference genome, it cannot be easily benchmarked against other programs. Here, we compare it to a general purpose transcriptome assembler, Trinity [8]. Both methods are compared only on the specific task of AS event calling. The current version of Trinity being restricted to a fixed value of k = 25, we systematically verified that this value was included in $[k_{min}, k_{max}]$.

We found out that Trinity was able to recover the AS event in all 3 samples only when the coverage was above 18× (270 RPK), which clearly shows that KISSPLICE is more sensitive for this task. This can be explained by the fact that TRINITY uses heuristics which consist in discarding a k-mer in the DBG whenever it is 20 times less frequent than an alternative k-mer branching at the same location in the DBG.

All these results were obtained using a minimal k-mer coverage (mkC for short) of 1. We also tested with mkC = 2 (*i.e.* k-mers present only once in the dataset are discarded), leading to the same main behaviour. We noticed however a loss in sensitivity for both methods, but a significant gain in the running time. KISSPLICE found the event in all 3 samples for a coverage of 12× (180 RPK) which remains better than the sensitivity of Trinity for mkC = 1.

Real data

We further tested our method on RNA-seq data from human. Even though we do not use any reference genome in our method, we applied it to cases where an annotated reference genome is indeed available in order to be able to assess if our predictions are correct.

We ran KISSPLICE with k = 25 and mkC = 2 on a dataset which consists of 32 M reads from human brain and 39 M reads from liver from the Illumina Body Map 2.0 Project (downloaded from the Sequence Read Archive, accession number ERP000546). As in all DBG-based assemblers, the most memory consuming step was the DBG construction which we performed on a cluster. The memory requirement is directly dependent on the number of unique k-mers in the dataset. Despite the fact that we do not use any heuristic to discard k-mers from our index, our memory performances are very similar to the ones of Inchworm, the first step of Trinity, as indicated in Figure 3. In addition, for the specific task of calling AS events, KISSPLICE is faster than TRINITY as shown in Figure 4.

KISSPLICE identified 5923 biconnected components which contained at least one bubble, 664 of which consisted of bubbles generated by approximate tandem repeats and 1160 which consisted of bubbles generated by short indels (less than 3 nt). Noticeably, the BCCs which generated most cycles and were most time consuming were associated to approximate tandem repeats. As these bubbles are not of interest for KISSPLICE, this

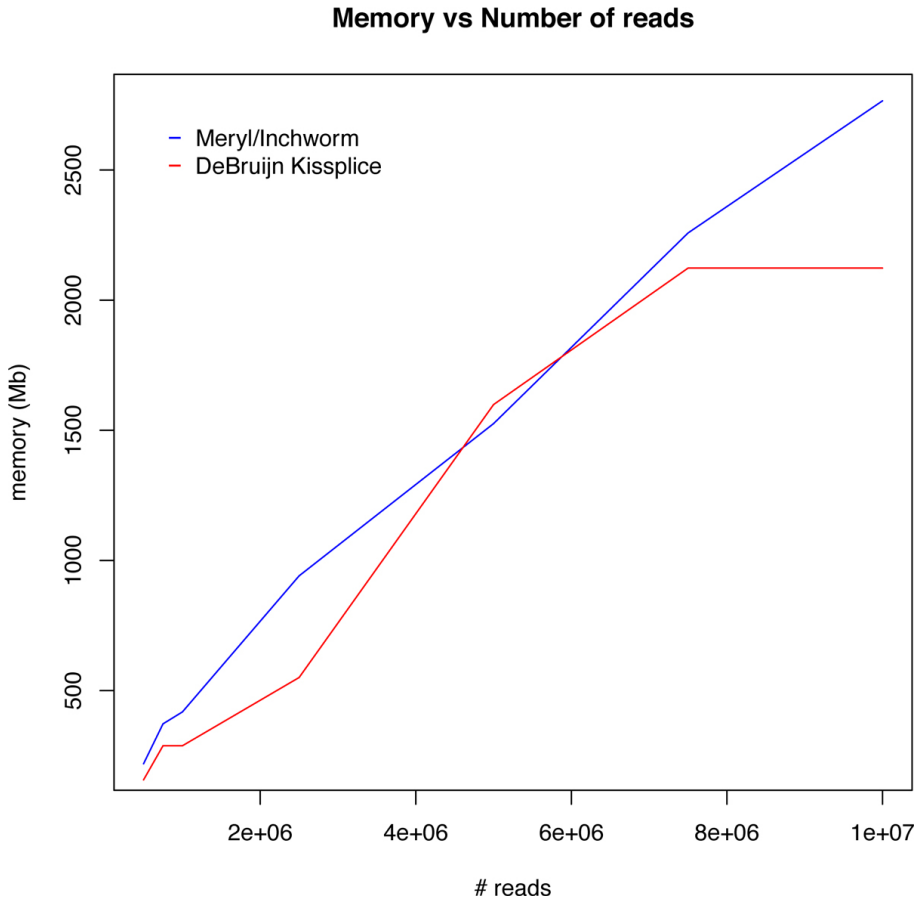


Figure 3 Memory usage. Memory usage of KISSPLICE and Inchworm as a function of input size.

observation prompted us to introduce an additional parameter in KISSPLICE to stop the computation in a BCC if the number of cycles being enumerated reaches a threshold. This enabled us to have a significant gain of time. We however advise not to use this threshold if the purpose is to identify AS events associated to approximate tandem repeats, which we did not address here.

Out of the 4099 remaining BCCs, we found that 3657 were read-coherent (*i.e.* each nucleotide is covered by at least one read) and we next focused on this set. For each of the 3657 cases, we tried to align the two paths of each bubble to the reference genome using Blat [18]. If the two paths align with the same initial and final coordinates, then we consider that the bubble is a real AS event. If they align with different initial and final coordinates, then we consider that it is a false positive. Out of the 3657 BCCs, 3497 (95%) corresponded to real AS events, while the remaining corresponded to false positives. A first inspection of these false positives led to the conclusion that the majority of them correspond to chimeric transcripts. Indeed, the shorter path and the longer path both map in two blocks within the same gene, but the second block is either upstream of the first block, or on the reverse strand, in both cases contradicting the annotations and therefore suggesting that the transcripts are chimeric and could have been generated by a genomic rearrangement or a trans-splicing mechanism.

For each of the 3497 real cases, we further tried to establish if they corresponded to annotated splicing events. We therefore first computed all annotated AS events using AStalavista [19] and the UCSC Known Genes annotation [20]. Then, for each aligned bubble, we checked if the coordinates of the aligned blocks matched the splice sites of the annotated AS events. If the answer was positive, then we considered that the AS event we found was known, otherwise we considered it was novel. Out of a total of 3497 cases, we find that only 1538 are known while 1959 are novel. This clearly shows that current annotations largely underestimate the number of alternative transcripts per multi-exon genes as was also reported recently [1].

Additionally, we noticed that 719 BCCs contained more than one AS event, which all mapped to the same

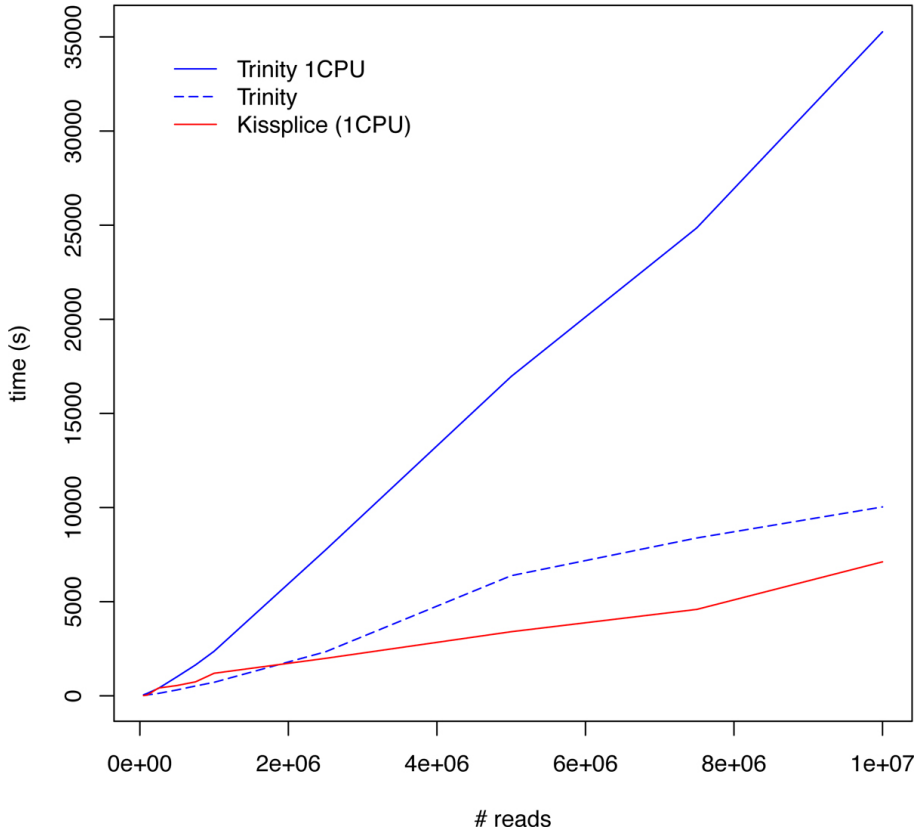


Figure 4 Time performances. Time performances of KISSPLICE and TRINITY as a function of input size.

gene. This corresponds to complex splicing events which involve more than 2 transcripts. Such events have been described in Sammeth et al. [7]. Their existence suggests that more complex models could be established to characterise them as one single event, and not as a collection of simple pairwise events. An example of novel complex AS event is given in Figure 5.

We also found the case where the same AS event maps to multiple locations on the reference genome (423 cases). We think these correspond to families of paralogous genes, which are "collectively" alternatively spliced. We were able to verify this hypothesis on all tested instances. In this case, we are unable to decide which of the genes of the family are producing the alternative transcripts, but we do detect an AS event.

Characterisation of novel AS events

In order to further characterise the 1959 novel AS events we found, we compared them with annotated events considering their abundance, length of the variable region and use of splice sites. For each AS event, we have 4 abundances, one for each spliceform (i.e. path of the bubble), and one for each condition. We computed the abundance of an event as the abundance of the minor spliceform. As outlined in Figure 6, we show that novel events are less abundant than annotated events. This in itself could be one of the reasons why they had not been annotated so far. Interestingly, we also found that while annotated events are clearly more expressed in brain than liver (median coverage of 3.4 Vs 1.2), this trend was weaker for novel events (2.4 Vs 1.2). This may reflect the fact that, since tissue-specific splicing in brain has been intensely studied, annotations may be biased in their favour.

We then computed the length of each event as the difference of the length between the two paths of the bubble. We found that for annotated events, there is a clear preference (59%) for lengths that are a multiple of 3, which is expected if the event affects a coding region. However, although still very different from random, this preference is less strong for novel events (45%), which, in addition are particularly enriched in short lengths as shown in Figure 7.

Finally, we computed the splice sites of annotated and novel events, and we found that a vast majority (99.5%)

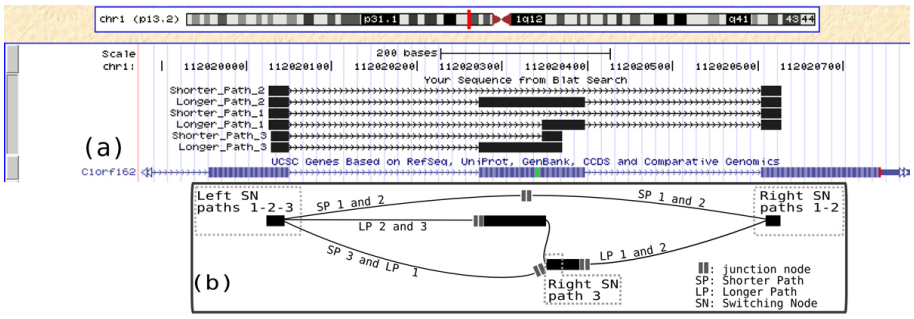


Figure 5 Complex AS event. BCC corresponding to a novel complex AS event. The intermediate annotated exon is either present, partially present, or skipped. (a) The annotations (blue track) report only the version where it is present while black tracks report all events found by KISSPLICE. (b) The cDBG associated to this complex event where the junction nodes are composed by $2k$ - 2 nucleotides.

of known events exhibit canonical splice sites, while this is again less strong for novel events (75.3%). Out of the non canonical cases, 13 correspond to U12 introns, but most correspond to short events.

Altogether, while we cannot discard that short non canonical events do occur and have been under-annotated so far, we think that the observations we make on the length and splice site features can be explained by the presence of genomic indels in our results. We had indeed already stated in Section "*De-Bruijn graph models*" that while most annotated genomic indels are below 3 nt, some may still be above. In practice, if the purpose is to strictly study AS events and not indels, then we recommend to focus on events longer than 10 nt, which have canonical splice sites in 95.2% of the cases. More generally, we wish to stress that this confusion between genomic indels and AS events is currently being made by all transcriptome assemblers.

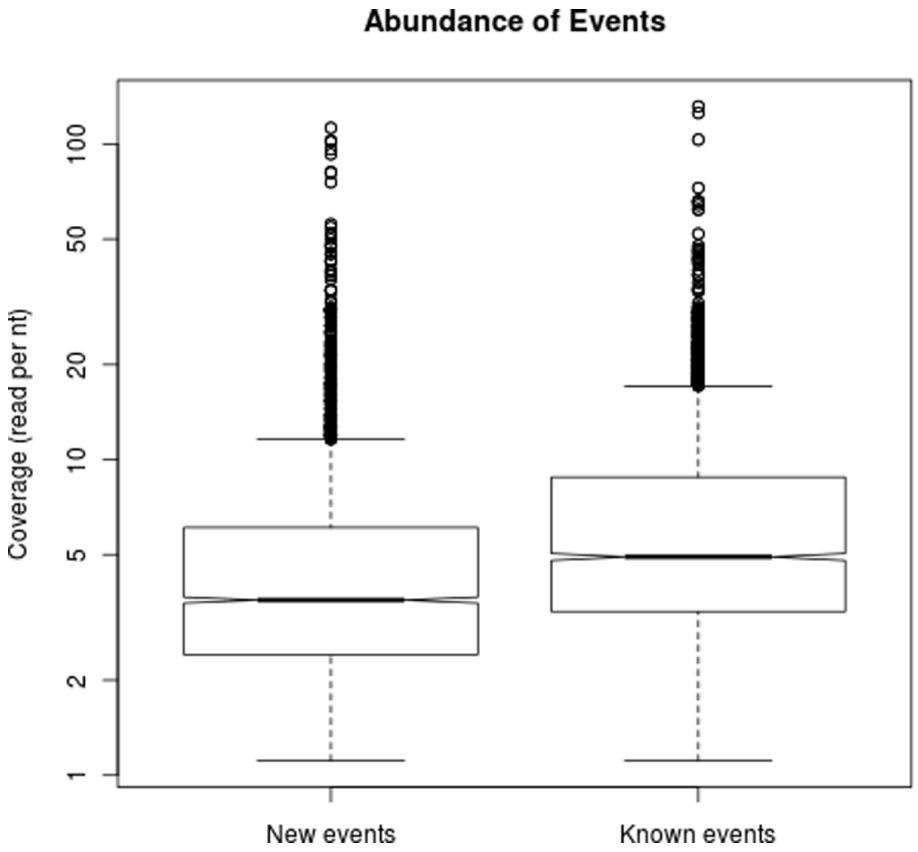


Figure 6 Event abundances. Abundance of known and novel events.

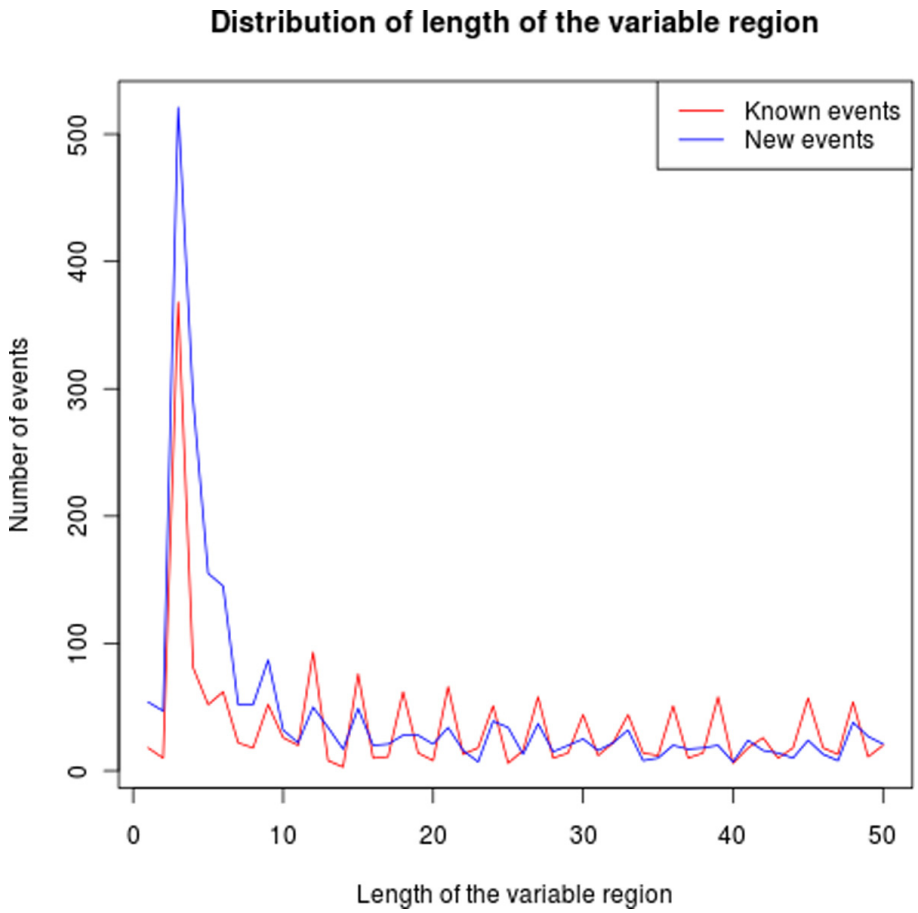


Figure 7 Event lengths. Distribution of lengths of the variable regions for known and novel events. Only the initial part of the distribution is given.

Comparison with Trinity

Finally, in order to further discuss the sensitivity of our method on real data, we compared our results with TRINITY. Although TRINITY is not tailored to find AS events, we managed to retrieve this information from the output. Whenever TRINITY found several alternative transcripts for one gene, we selected this gene. We further focused on cases which contained a cycle in the splicing graph reconstructed from this gene and we compared them with the events found by KISSPLICE. Whenever we found that both the longer and the shorter path of a bubble were mapping to the transcripts of a TRINITY gene, we decided that both methods had found the same event. In total, KISSPLICE found 4099 cases, TRINITY found 1123 out of which 553 were common. While the sensitivity is overall larger for KISSPLICE, we see that 570 cases are found by Trinity and not by KISSPLICE. We then mapped these transcripts to the human genome using blat. In many instances (348 cases), the transcripts did not align on their entire length, or to different chromosomes, indicating that they corresponded to chimeras. A first inspection of the remaining 222 cases revealed that they correspond to the complex BCCs we chose to neglect at an early stage of the computation, because they contain a very large number of approximate tandem repeats. A first simple way to deal with this issue is to increase the value of k. The effect of this is to break the large BCCs into computable cases, enabling to recover a good proportion of the missed events. For instance, for $k = 35$, we found back 84 cases. More generally, this shows that more work on the model and on the algorithms is still required to characterise better AS events which are intricated with approximate tandem repeats. We think that TRINITY manages to identify some of them because it uses heuristics, which enables it to simplify these complex graph structures.

Conclusions

This paper presents two main contributions. First, we introduced a general model for detecting polymorphisms in De Bruijn graphs, and second, we developed an

algorithm, KISSPLICE, to detect AS events in such graphs. This approach enables to tackle the problem of finding AS events without assembling the full-length transcripts, which may be time consuming and uses heuristics that may lead to a loss of information. To our knowledge, this approach is new and should constitute a useful complement to general purpose transcriptome assemblers.

Results on human data show that this approach enables de-novo calling of AS events with a higher sensitivity than obtained by the approaches based on a full assembly of the reads, while using similar memory requirements and less time. 5% of the extracted events correspond to false positives, while the 95% remaining can be separated into known (44%) and novel events (56%). Novel events exhibit similar sequence features as known events as long as we focus on events longer than 10 nt. Below this, novel events seem to be enriched in genomic indels.

KISSPLICE is available for download at http://alcovna.genouest.org/kissplice/ and can already be used to establish a more complete catalog of AS events in many species, whether they have a reference genome or not. Despite the fact that more and more genomes are now being sequenced, the new genome assemblies obtained usually do not reach the level of quality of the ones we have for model organisms. Hence, we think that methods which do not rely on a reference genome are not going to be easily replaced in the near future. There is of course room for future work. The KISSPLICE algorithm could be improved in several ways. The coverage could be used for distinguishing SNPs from sequencing errors. Moreover, the sequences surrounding the bubbles could be locally assembled using a third party tool [21]. This would allow to output their context or the full contig they belong to.

Last, the complex structure of BCCs associated to approximate tandem repeats seems to indicate that more work on the model and on the algorithms is required to efficiently deal with the identification of real approximate tandem repeat events, which may be highly intertwined with other events.

Acknowledgements

We would like to thank the reviewers for helpful comments on earlier versions of this manuscript. We also thank Nadia Pisanti for helpful discussions on the KisSplice model and Claire Lemaitre for insightful comments on the quantification step. This work was supported by the following grants: the Inria Alcovna ARC, the french ANR MIRI BLAN08-1335497 Project, the ERC Advanced Grant Sisyphe held by Marie-France Sagot, and by the french ANR-2010-COSI-004 MAPPI Project.

This article has been published as part of *BMC Bioinformatics* Volume 13 Supplement 6, 2012: Proceedings of the Second Annual RECOMB Satellite Workshop on Massively Parallel Sequencing (RECOMB-seq 2012). The full contents of the supplement are available online at http://www.biomedcentral.com/bmcbioinformatics/supplements/13/S6.

Author details

[1]INRIA Grenoble Rhône-Alpes, France. [2]Université de Lyon, F-69000, Lyon; Université Lyon 1; CNRS, UMR5558, Laboratoire de Biométrie et Biologie Evolutive, F-69622, Villeurbanne, France. [3]Centre de recherche INRIA Rennes - Bretagne Atlantique, IRISA, Campus universitaire de Beaulieu, Rennes, France. [4]INRA UMR118, Amélioration des Plantes et Biotech. Végétales, Rennes, France.

Authors' contributions

VL proposed the models, PP GS and MFS developed the algorithms, RC implemented the DBG construction, PA and GS implemented the main algorithms, PP supervised the implementations, JK ran the tests on simulated data, RU ran the tests on real data, VL supervised the tests. All authors contributed to the writing of the paper.

Competing interests

The authors declare that they have no competing interests.

Published: 19 April 2012

References

1. Wang ET, Sandberg R, Luo S, Khrebtukova I, Zhang L, Mayr C, Kingsmore SF, Schroth GP, Burge CB: **Alternative isoform regulation in human tissue transcriptomes.** *Nature* 2008, **456(7221)**:470-476.
2. Martin JA, Wang Z: **Next-generation transcriptome assembly.** *Nature Reviews Genetics* 2011, 1-12.
3. Zerbino DR, Birney E: **Velvet: Algorithms for de novo short read assembly using de Bruijn Graphs.** *genome research* 2008, **18(5)**:821-9.
4. Simpson J, Wong K, Jackman S, Schein J, Jones S, Birol I: **ABySS: A parallel assembler for short read sequence data.** *Genome Research* 2009, **19(6)**:1117-1123.
5. Pevzner P, Tang H, Tesler G: ***De novo* repeat classification and fragment assembly.** *RECOMB* 2004, 213-222.
6. Iqbal Z, Caccamo M, Turner I, Flicek P, McVean G: **De novo assembly and genotyping of variants using colored de Bruijn graphs.** *Nature genetics* 2012.
7. Sammeth M: **Complete alternative splicing events are bubbles in splicing graphs.** *Journal of Computational Biology* 2009, **16(8)**:1117-1140.
8. Grabherr MG, Haas BJ, Yassour M, Levin JZ, D AT, *et al*: **Full-length transcriptome assembly from RNA-Seq data without a reference genome.** *Nat Biotechnol* 2011, **29(7)**:644-652.
9. Robertson G, Schein J, Chiu R, Corbett R, Field M, Jackman SD, Mungall K, Lee S, Okada HM, Qian JQ, Griffith M, Raymond A, Thiessen N, Cezard T, Butterfield YS, Newsome R, Chan SK, She R, Varhol R, Kamoh B, Prabhu AL, Tam A, Zhao Y, Moore RA, Hirst M, Marra MA, Jones SJM, Hoodless PA, Birol I: **De novo assembly and analysis of RNA-seq data.** *Nature Methods* 2010, **7(11)**:909-912.
10. Schulz MH: **Data Structures and Algorithms for Analysis of Alternative Splicing with RNA-Seq Data.** *PhD thesis* Free University of Berlin; 2010.
11. Pevzner PA, Tang H, Waterman MS: **An Eulerian path approach to DNA fragment assembly.** *Proceedings of the National Academy of Sciences of the USA* 2001, **98(17)**:9748-53.
12. Medvedev P, Georgiou K, Myers G, Brudno M: **Computability of Models for Sequence Assembly.** *WABI* 2007, 289-301.
13. Sherry ST, Ward MH, Kholodov M, Baker J, Phan L, Smigielski EM, Sirotkin K: **dbSNP: the NCBI database of genetic variation.** *Nucleic Acids Research* 2001, **29**:308-311.
14. Tarjan RE: **Depth-First Search and Linear Graph Algorithms.** *SIAM Journal on Computing* 1972, **1(2)**:146-160.
15. Tiernan JC: **An efficient search algorithm to find the elementary circuits of a graph.** *Communonications ACM* 1970, **13**:722-726.
16. Tarjan RE: **Enumeration of the Elementary Circuits of a Directed Graph.** *SIAM Journal on Computing* 1973, **2(3)**:211-216.
17. Richter DC, Ott F, Auch AF, Schmid R, Huson DH: **MetaSim: a sequencing simulator for genomics and metagenomics.** *PLoS One* 2008, **3(10)**:e3373.
18. Kent WJ: **BLAT-the BLAST-like alignment tool.** *Genome Research* 2002, **12(4)**:656-664.
19. Sammeth M, Foissac S, Guigó R: **A general definition and nomenclature for alternative splicing events.** *PLoS Computational Biology* 2008, **4(8)**: e1000147.

20. Kuhn RM, Karolchik D, Zweig AS, Wang T, K ES, *et al*: **The UCSC Genome Browser Database: update 2009.** *Nucleic Acids Research* 2009, , **37** Database: D755-D761.
21. Peterlongo P, Chikhi R: **Mapsembler, targeted assembly of larges genomes on a desktop computer.** *Research report RR-7565, INRIA* 2011.

doi:10.1186/1471-2105-13-S6-S5
Cite this article as: Sacomoto *et al.*: KISSPLICE: de-novo calling alternative splicing events from RNA-seq data. *BMC Bioinformatics* 2012 **13**(Suppl 6):S5.

Reference-free detection of isolated snps

Published online 17 November 2014

Nucleic Acids Research, 2015, Vol. 43, No. 2 ***e11***
doi: 10.1093/nar/gku1187

Reference-free detection of isolated SNPs

Raluca Uricaru[1,2,3,*], Guillaume Rizk[4], Vincent Lacroix[5,6], Elsa Quillery[7,8], Olivier Plantard[7,8], Rayan Chikhi[9], Claire Lemaitre[4,*] and Pierre Peterlongo[4,*]

[1]University of Bordeaux, CNRS/LaBRI, F-33405 Talence, France, [2]University of Bordeaux, CBiB, F-33000 Bordeaux, France, [3]INRA, UMR1349 IGEPP, Le Rheu, France, [4]GenScale, INRIA Rennes Bretagne-Atlantique, IRISA, Rennes, France, [5]BAMBOO, INRIA Grenoble Rhone-Alpes, Lyon, France, [6]Laboratoire de Biométrie et Biologie Évolutive, Université Lyon 1 UMR CNRS 5558, Lyon, France, [7]INRA, UMR1300 Biology, Epidemiology and Risk Analysis in Animal Health, Nantes, France, [8]LUNAM University, Oniris, Nantes Atlantic College of Veterinary Medicine and Food Sciences and Engineering, UMR BioEpAR, Nantes, France and [9]Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA, 16802, USA

Received February 14, 2013; Revised October 17, 2014; Accepted November 03, 2014

ABSTRACT

Detecting single nucleotide polymorphisms (SNPs) between genomes is becoming a routine task with next-generation sequencing. Generally, SNP detection methods use a reference genome. As non-model organisms are increasingly investigated, the need for reference-free methods has been amplified. Most of the existing reference-free methods have fundamental limitations: they can only call SNPs between exactly two datasets, and/or they require a prohibitive amount of computational resources. The method we propose, DISCOSNP, detects both heterozygous and homozygous *isolated* SNPs from any number of read datasets, without a reference genome, and with very low memory and time footprints (billions of reads can be analyzed with a standard desktop computer). To facilitate downstream genotyping analyses, DISCOSNP ranks predictions and outputs quality and coverage per allele. Compared to finding isolated SNPs using a state-of-the-art assembly and mapping approach, DISCOSNP requires significantly less computational resources, shows similar precision/recall values, and highly ranked predictions are less likely to be false positives. An experimental validation was conducted on an arthropod species (the tick *Ixodes ricinus*) on which *de novo* sequencing was performed. Among the predicted SNPs that were tested, 96% were successfully genotyped and truly exhibited polymorphism.

INTRODUCTION

Assessing the genetic differences between individuals within a species or between chromosomes of an individual is a fundamental task in many aspects of biology. This is increasingly feasible with next-generation sequencing technologies, as individuals from virtually any species can be sequenced at a modest cost. Of specific interest, single nucleotide polymorphisms (SNPs) are variations of a single base, either between two homologous chromosomes within a single individual, or between two individuals. Finding biallelic or Mendelian SNPs is often done in many biological applications involving SNP genotyping, e.g. population genomics, health, ecology or agronomy research (1,2). To be easily amplified by polymerase chain reaction (PCR), such SNPs must not be surrounded by other polymorphism sources, i.e. other SNPs, indels or structural variants. We call such SNPs *isolated*. Formally, *isolated* SNPs must be distant to the left and to the right by at least k nucleotides from any other polymorphism, k being one of the main parameters of a SNP detection tool.

State-of-the-art methods to detect SNPs between individuals or strains, generally map sequenced reads (GATK (3), SAMtools (4)) or partial assemblies (DISCOVAR, FERMI (5)) on a reference genome. These *reference-based* methods clearly cannot be applied when there is no reference genome. In fact, even when there exists a reference genome, the behavior of these methods is highly dependent on the quality of the assembly. The reality today is that with sequencing costs falling, sequencing efforts are no longer limited to the main species of interest (human and other primates, mouse, rat, drosophila, yeast, *Escherichia coli*, etc.). In fact, biologists are increasingly working on data for which they do not have any close reference genome. Unfortunately, while sequencing with NGS (Next Generation Sequencers) is becoming routine, assembling genomes remains a very

*To whom correspondence should be addressed. Tel: +33(0)2 99 84 74 59; Fax: +33(0)2 99 84 71 71; Email: pierre.peterlongo@inria.fr
Correspondence may also be addressed to Raluca Uricaru. Tel: +33 (0)5 40 00 60 95; Fax: +33(0)5 40 00 66 69; Email: ruricaru@labri.fr
Correspondence may also be addressed to Claire Lemaitre. Tel: +33(0)2 99 84 71 16; Fax: +33(0)2 99 84 71 71; Email: claire.lemaitre@inria.fr

complicated task, for which no single software performs consistently well (6), thus producing reference sequences of poor quality. For these reasons, there is a strong need for *reference-free* methods able to detect SNPs, in particular those which are isolated, without relying on a reference genome (reference-free methods).

Reference-free methods that detect SNPs can be broadly divided into two categories. The first category methods perform *de novo* assembly to build a reference sequence. Then, these methods include, as a sub-module, a reference-based method to map back the reads of each individual to this reference sequence (7). We refer to such methods as *hybrid*, as they use both *de novo* assembly and mapping techniques to call SNPs.

A major limitation of these methods is that they tend to cumulate problems raised in assembly and problems raised in mapping. An interesting alternative is to directly focus on the assembly of SNPs, without trying to assemble a full reference. We refer to such methods as *de novo*.

Several methods that fall into the *de novo* category have been developed recently (8–12). These methods are based on a *de Bruijn graph*, i.e. a directed graph where the set of vertices corresponds to the set of words of length k (k-mers) contained in the reads, and there is an edge between two k-mers if they overlap on $k-1$ nucleotides. KISSNP (8) is a software that takes as input two sets of reads, and detects SNPs using k-mers that are differentially abundant between the two datasets. CORTEX_VAR (9) (here after called CORTEX) detects and genotypes SNPs (as well as indels and larger events), between n datasets. BUBBLEPARSE (10), which is based on the same graph structure as CORTEX, detects SNPs that it furthermore classifies into homozygous and heterozygous groups. NIKS (11) finds homozygous SNPs between two datasets by performing local *de novo* assembly around sample-specific k-mers. KSNP V2 (13) proposes yet another approach that is rather dedicated to SNP phylogeny on microbial species. Even if the SNP detection is performed *de-novo*, this method is strongly based on a reference sequence (on which putative selected k-mers witnessing a SNP are mapped).

These methods all entail unpractical computational costs (in particular in terms of memory) for large and complex genomes. Even with substantial computational resources, most of them still cannot detect variants in mammalian-size datasets. Therefore, there is still a strong need for robust tools that can detect isolated SNPs without a reference genome.

For these reasons, we introduce a new *de novo* method called DISCOSNP. This method is designed to call isolated SNPs directly from sequenced reads, without a reference genome. As shown in this paper, DISCOSNP opens the way to the discovery of SNPs from large-scale studies, even on a standard desktop computer. DISCOSNP finds and ranks high quality isolated heterozygous or homozygous SNPs from any number of read sets, from 1 to n. It introduces new features to distinguish SNPs from sequencing errors or false positives due to approximate repeats. DISCOSNP can be used for finding high-quality isolated SNPs, either heterozygous, e.g. to build databases of high-quality markers within and across populations, or homozygous between individuals/strains, e.g. to create discriminant markers. The genotyping can be performed as a downstream analysis, based on the read coverage information it provides.

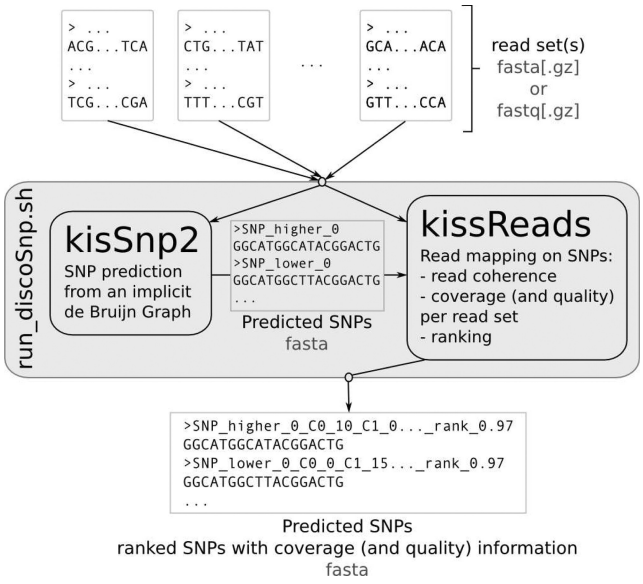


Figure 1. DISCOSNP method diagram. DISCOSNP is composed of two modules, KISSNP2 and KISSREADS that are called by the *run_disco.sh* script.

Results presented in this paper show that DISCOSNP outperforms other reference-free SNP detection methods in terms of resources (faster and using at least two orders of magnitude less memory), type and number of input dataset(s) (ability to find both homozygous and heterozygous SNPs from more than two datasets), and quality of the ranking of predicted isolated SNPs. On two whole-genome mouse datasets, DISCOSNP detected 2 million isolated SNPs, of which 84% were also found by a manually tuned pipeline in a previous study. On a *S. cerevisiae* dataset DISCOSNP predicted 90% of the validated SNPs. On an arthropod dataset, DISCOSNP found 0.3 million isolated SNPs; furthermore, an experimental validation carried over a sample of these SNPs confirmed 96% of them. Finally, DISCOSNP is designed to reach a wide audience, as it aims to be easy to use, regardless of the size and the complexity of the input data.

MATERIALS AND METHODS

Algorithms

As depicted in Figure 1 DISCOSNP is composed of two modules, KISSNP2 and KISSREADS. KISSNP2 detects putative SNPs from one or more sets of reads. KISSREADS evaluates the coverage and base quality of the SNPs per read set and ranks them accordingly. These two modules are independent, however, a script automatically pipelines them, so their calls are transparent to the user.

KISSNP2 *module.* Similarly to CORTEX and KISSNP, the KISSNP2 module is based on a *de Bruijn graph*. A *de Bruijn graph* is a directed graph that contains all the k-mers present in the read dataset as vertices, and all the possible $(k-1)$-overlaps between k-mers as edges. Such graphs have been widely used in *de novo* assembly (14). The idea is the following: if a dataset contains two sequences that are identical,

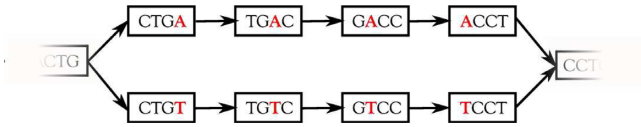


Figure 2. Toy example of a *bubble* in the *de Bruijn Graph* ($k = 4$). Bubble generated by a single nucleotide polymorphism. The two polymorphic sequences are $\ldots CTGACCT\ldots$ and $\ldots CTGTCCT\ldots$

except for one character, then these sequences generate a *bubble* in the graph (see an example in Figure 2). Formally, a *bubble* in a *de Bruijn graph* is composed of two distinct paths of $k + 2$ nodes, having the start and the end nodes in common. Precisely, KISSNP2 detects couples of paths of length $2k - 1$, denoted by p_1 and p_2, that correspond to polymorphic sequences, i.e. sequences of bubbles excepting the two extreme nodes. Formally, these two paths can be written as $p_1 = p\alpha q$ and $p_2 = p\beta q$, with p, q being $(k - 1)$-mers, and α and β are single nucleotides such that $\alpha \neq \beta$.

In KISSNP2, this idea is exploited by generating the *de Bruijn graph* of all input dataset(s) pooled together, and by searching the previously described bubbles in the graph. To avoid k-mers generated by sequencing errors, only k-mers whose support (coverage) is above or equal to a user-defined threshold, c, are taken into account in the construction of the *de Bruijn graph*. The *de Bruijn graph* is built with the MINIA data structure (15,16). The nodes of the graph are stored in a cascade of Bloom filters. As the edges of the graph can be inferred from the nodes, they do not need to be stored. Overall, the graph representation requires around 1 byte per k-mer. This data structure supports efficient and exact enumeration of the neighbors of any node in the graph. Thus, it enables to efficiently traverse the *de Bruijn graph* starting from any node, on both forward and reverse strands.

In the following, we say that a k-mer ω can be *right extended* with a nucleotide α if the k-mer obtained by concatenating the suffix of length $k - 1$ of ω with α, exists in the data structure. Symmetrically, we say that a k-mer ω can be *left extended* with a nucleotide α, if the concatenation of α with the prefix of length $k - 1$ of ω forms a k-mer that can be found in data structure.

The KISSNP2 algorithm detects all k-mers that can be right extended with at least two distinct nucleotides. Such k-mers are the starting nodes of the bubbles they generate, as in the example depicted in Figure 2, where the k-mers obtained after the extension are *CTGA* and *CTGT*. Then, for each such couple of k-mers starting with the same $k - 1$ length prefix, KISSNP2 tries to complete the bubble by performing successively $k - 1$ right extensions on both paths with the same nucleotide (using successively nucleotides *C*, *C* and *T* for the example in Figure 2). If, at one step, both paths cannot be right extended with the same nucleotide, then the bubble is discarded. Based on this pattern, only *isolated* SNPs can be detected, thus avoiding SNPs that are closer than k bases to other polymorphisms.

Branching bubbles. High copy number repeats in genomes typically yield complex bubbles, which may combinatorially increase the number of false positives. To limit this effect,

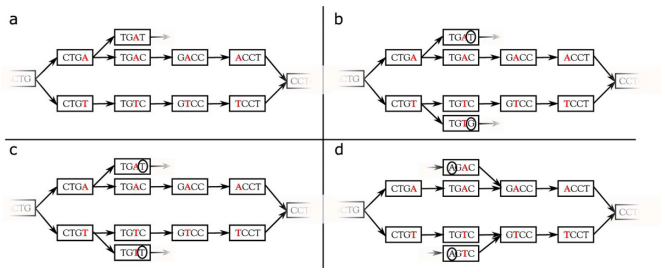


Figure 3. Examples of non-symmetrically branching bubbles (**a** and **b**) and symmetrically branching bubbles (**c** and **d**). Path divergences in bubbles a and b create branching bubbles, but the branching is not symmetric. Divergence is only present in one path (**a**) or in both paths but with distinct (circled) characters (**b**). Path divergences in bubbles c and d create symmetrically branching bubbles. Both paths of these bubbles can be right extended (**c**) or left extended (**d**) with the same two (circled) characters. With option −b 0 (default), none of these bubbles would be considered as a SNP, with option −b 1 bubbles **a** and **b** would have been considered as SNP while with option −b 2 all of them would have been output.

a classic filter consists in removing bubbles that contain at least one branching node; this is the default behavior of DISCOSNP (−b 0). The counterpart of this filter is that it may discard true bubbles that contain branchings only due to repeats or to some isolated non-filtered sequencing errors. To achieve a better compromise between false positive and false negative rates, especially in complex genomes, we introduce a novel concept of symmetrically branching bubbles.

More precisely, a bubble is called *non-branching* if during its construction, for every extension step, there is only one k-mer that can be used (only one possible extension for each of the two paths). On the other hand, if at some point during the extension of the bubble, there is a choice between two k-mers (for at least one of the paths), the bubble is said to be *branching*. Now, if at some point during the extension, both paths can be extended with the same (at least) two nucleotides, then the bubble is called *symmetrically branching* (see examples in Figure 3). Finally, bubbles that are branching, but not symmetrically branching, are called *simply branching*. Note that branchings are checked in both directions (left and right).

The rationale for keeping simply branching bubbles is that they may correspond to true SNPs, where the branching is simply due to a sequencing error that accidentally passed the coverage filter. As shown in the 'Results' section, this enables to increase the recall especially in complex genomes, with high copy number repeats, while being highly efficient in removing the numerous false positive branching bubbles. DISCOSNP can be run with any of these three filtering strategies: filtering out all branching bubbles (default behavior, option −b 0), or filtering out symmetrically branching bubbles only (−b 1), or no filtering based on the branching criteria (−b 2).

Retrieving sequence contexts. Each sequence in the multi-fasta file output by KISSNP2 is composed of the $2k - 1$ nucleotides, which correspond to the bubble, together with its left and right contigs that are extending the $2k - 1$ sequences. For every such contig, the length of the longest unambiguous context (longest non-branching path in the graph starting from the bubble) is also indicated. Recon-

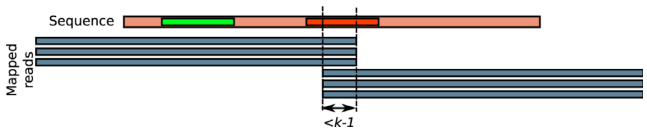


Figure 4. Read-coherency and *k*-read-coherency example. With coverage threshold = 2, schematic example where a sequence is *read-coherent* but not *k-read-coherent*. The leftmost represented *k*-mer (green) on the sequence is an example where the *k*-mer starting at this position is covered with three mapped reads. On the other hand, the rightmost represented *k*-mer (red) is covered by no read, thus illustrating why the sequence is not *k-read-coherent*.

structing these contexts is done with the MINIA assembler (15) algorithm. Its contiging algorithm was evaluated within the Assemblathon2 challenge (17) (team '*Symbiose*').

KISSREADS *module.* One should note that it is not possible to compute read coverages, nor read quality information, based solely on a raw *de Bruijn graph*. Moreover, all *de Bruijn graph*-based methods may build chimeric sequences due to overlapping *k*-mers, which are never present in the same read. These sequences are said to be *non-read-coherent*. The KISSREADS module is meant to filter out bubbles composed of *non-read-coherent* sequences, to add coverage and quality information on the remaining ones, and finally to rank SNPs (see the section below).

Given a sequence s and a set of reads $\mathcal{R}$ mapped on s, we say that s is *read-coherent* if, for each position of s, at least c reads are mapped, with c a user-defined threshold. Note that reads are mapped in a semi-global manner: mapped reads may have a prefix starting before the first position of s and/or may have a suffix ending after the last position of s. By default, one substitution is authorized which roughly corresponds to current error rates. Moreover, knowing that in the DISCOSNP framework the sequence s represents one of the two alleles of a SNP, no substitution is authorized on the polymorphic position during the mapping.

Moreover, it may appear that a position of s is mapped only by the *end* of reads (last k positions), and/or only by the *beginning* of reads (first k positions). This reveals a situation where part of the sequence s was generated by k-mers not truly belonging to the mapped reads, i.e. due to a repeat of length bigger or equal to $2k$ and smaller than the read size. Such a sequence is thus chimeric. An example of this situation is illustrated in Figure 4. To overcome this problem, we define the *k-read-coherency*. Given a sequence s and a set of reads $\mathcal{R}$ that can be mapped on s, we consider s as *k-read-coherent* if, for each position i of s except the last $k-1$ ones, there exists at least c reads that fully map on the k-mer starting on position i. Note that this condition is symmetrical, whether s is read on both forward or reverse complement strand.

Given the set $\mathcal{S}$ of pairs of $2k-1$ sequences generated by KISSNP2 and the initial sets of reads, the KISSREADS algorithm maps the reads (using a classic seed-and-extend approach) on the sequences of $\mathcal{S}$. Once all reads are mapped on all sequences of $\mathcal{S}$, the *k-read-coherency* is computed for each read set and for each sequence of $\mathcal{S}$. Sequences of $\mathcal{S}$ for which at least one read set makes them *k-read-coherent*, are conserved. KISSREADS outputs such coherent sequences together with their read depth per read set and with the average PHRED quality of the polymorphic nucleotide per read set.

Ranking SNPs

DISCOSNP scores each SNP according to the coverage repartition of its alternative paths between the conditions. The aim is to rank best the SNPs that are the most discriminant between the samples. For a given SNP, the score is the *Phi coefficient* of the table of read counts for each path and each dataset, computed as follows: $\sqrt{\frac{\chi^2}{n}}$. It can be seen as a normalized Chi-squared statistics that varies between 0 and 1. A high score, close to 1, is obtained if the frequencies of the paths are very different between datasets, the best case being for homozygous SNPs between two datasets, where each path is strictly specific to one dataset. Notably, this score ranks poorly bubbles that are due to sequencing errors or inexact repeats as they are likely to have similar repartitions in all datasets (small frequency for an error, and equal frequency for repeats). Moreover, the normalization prevents from over-scoring highly covered bubbles which are often due to repeats. Since this ranking favors SNPs for which one variant is enriched in one read set, it is not well suited to select SNPs that are heterozygous in all read sets. It is also not usable with only one dataset, one diploid individual or a pooled sample. As shown in the 'Results' section, other rankings can be used in these cases.

DISCOSNP input and output

In summary, DISCOSNP takes as input any number of read sets, and has two main parameters k and c, respectively the k-mer size used to build the *de Bruijn graph* and the minimal coverage a k-mer should reach to be inserted in the graph. When applied to two read datasets or more, DISCOSNP is executed only once as all datasets are pooled together. All isolated SNPs shared by any number of samples are output as single calls.

Finally, the DISCOSNP output is a sorted *multi-fasta* file, in which every consecutive couple of sequences corresponds to the two $2k-1$ paths of a SNP, surrounded by its left and right contigs. Headers of the sequences give information about read coverage and average PHRED quality for each input dataset, lengths of unambiguous left and right extensions, and Phi coefficient of each SNP.

Simulated datasets and evaluation

We propose here an overview of the evaluation approach. Supplementary details on the protocols for generating data and for evaluating the results are provided in Supplementary Additional File 1.

All tools were run with at least $k = 31$ and a minimal coverage of 4 (k-mers that were seen less than four times were considered to be sequencing errors) or based on the developers' advices. When several parameter sets were tested, presented results were obtained with those giving the best results. Full details on parameters and full results are proposed in the Supplementary Additional File 1.

Simulation and evaluation for one or two diploids. In order to evaluate the behavior of DISCOSNP on large, complex genome sequences, and to compare it to the other reference-free SNP discovery tools, we propose an experiment simulating the sequencing of two diploid, human chromosome 1 individuals.

To obtain a realistic distribution of SNPs and genotypes, we used SNPs predicted from real human individuals. Two vcf files were retrieved from the '1000 genome project' (phase 1 release), corresponding to the human chromosome 1 of two individuals: HG00096 and HG00100. We then generated the genome sequences for the two diploids, i.e. two sequences per individual, by placing the substitutions listed in the vcf files on the human reference sequence (GRCh37/hg19 reference assembly version). In the case of a homozygous SNP, the same nucleotide was placed on the two sequences, while for a heterozygous SNP, one sequence was randomly chosen for each of the two nucleotides.

A total of 316 502 positions were mutated, with 131 263 positions that were mutated in the same time in both individuals (representing an average ratio of 0.5 SNPs by kb in each individual). 29 038 SNPs (9%) have a homozygous genotype in both individuals (homozygous-homozygous), 218 556 (69%) are heterozygous in only one individual (homozygous–heterozygous) and the remaining 68 908 (22%) are heterozygous in both individuals.

We then simulated a 40x coverage sequencing of these two individuals, with 100-bp reads and 1% error rate, thus generating 89 753 907 reads. The sequencing procedure is further explained in Supplementary Additional File 1.

Among the simulated SNPs, only those distant by more than k nucleotides from one another were selected (isolated SNPs) and were recorded as $2k-1$ bubbles in a reference file (same format as the output of DISCOSNP). This gave 150 348 heterozygous isolated SNPs for individual HG00096 (84% of all heterozygous mutations inserted in this individual) and 260 539 isolated SNPs when considering both individuals (82% of all simulated SNPs). We produced two SNP reference files, *ref_snps.fa* (as described in 'Precision and recall computation', in Supplementary Additional File 1): one for the HG00096 individual, and a second one when considering the two individuals. Predicted SNPs were compared to one of these files (depending on the dataset(s) on which DISCOSNP was applied) in order to compute the number of true and false positives. A predicted SNP for which the two paths match exactly the two paths of a simulated SNP present in this reference file is a true positive (TP), else it is a false positive (FP). In the same way, a SNP from the reference file not matched by any predicted SNP is a false negative (FN). The recall is given by the number of TP divided by the number of TP plus the number of FN. The precision is given by the number of TP divided by the number of TP plus the number of FP. When allowing branching bubbles (option −b 1), precision was computed by considering as false positives only bubbles that do not correspond to any simulated SNP (isolated or not). Full details on the evaluation protocol can be found in the Supplementary Additional File 1. Isolated SNPs detected with other tools than DISCOSNP were transformed into the DISCOSNP output format, in order to be evaluated with the same protocol.

Simulation and evaluation for two and more haploids. We propose an experiment simulating more than two haploid bacterial individuals. For doing this, we created 30 copies (that we call *individuals*) of the *E. coli* K-12 MG1655 strain. We then simulated SNPs with a uniform distribution such that ≈4200 SNPs (≈0.1% of the genome length) are common to any pair of individuals, half this number is common to any trio of individuals, a third to any quadruplet, and so on. With this strategy, while considering all the 30 individuals together, 69 600 SNP sites were generated, covering ≈1.5% of the genome. Similarly to the diploid dataset, we simulated a 40x sequencing of each of the 30 individuals, with 100-bp reads and 0.1% error rate. Thus, 1 855 870 reads were generated per read set.

Finally, we created a reference file containing the isolated-SNPs per subsets of individuals: the first two, the first three, and so on. For each of these subsets, a SNP was considered as isolated if no other SNPs were simulated in the $k-1$ positions before and after the SNP's locus, inside this subset. In the presented results, for two individuals, 4164 simulated SNPs are isolated while 268 SNPs (≈6%) are not. For 30 individuals, 25 993 SNPs are isolated while 333 930 (≈92%) are not. For the isolated SNPs, the two corresponding sequences of length $2k-1$ were used as a reference to assess the precision/recall of all tested tools, based on the same protocol as the one used for two diploids.

Mouse dataset

DISCOSNP was applied on a real dataset to compare two mouse inbred strains, *FVB/NJ* and *C57BL/6NJ*, the latter corresponding to the mouse reference genome (NCBIM37). Reads were retrieved from the European Nucleotide Archive (www.ebi.ac.uk/ena/), 987 million reads for the *C57BL/6NJ* strain (ERP000041) and 1888 million reads for the *FVB/NJ* strain (ERP000687).

To compare with DISCOSNP results, we retrieved the set of predicted SNPs obtained with the same data by Wong *et al.* (18) (www.sanger.ac.uk/resources/mouse/genomes/, file mgp.v3.snps.rsIDdbSNPv137.vcf). We selected SNPs having distinct genotypes in *FVB/NJ* and *C57BL/6NJ* strains. This provided 5 020 489 SNPs, among which 2 472 456 (≈49.2%) are isolated. We evaluated the number of isolated SNPs found by the two methods and those found specifically by one method or the other, by comparing the DISCOSNP output to the Wong *et al.* one. The set of SNPs that was predicted by Wong *et al.* was transformed in DISCOSNP format, as for the other experiments, in order to allow the comparisons.

Saccharomyces cerevisiae dataset

DISCOSNP was applied on a real *S. cerevisiae* dataset, for which several SNPs have been biologically validated in a recent study (19). In this study, three glucose-limited, chemostat-evolved populations of haploid S288c, named E1, E2 and E3, were sequenced every ≈70 generations, giving eight samples per population. Using a reference-based mapping approach, 110 mutations were discovered, among which only 33 have a minor allele frequency (MAF) >10% and were confirmed by Sanger sequencing. Among these 33

SNPs, 30 are isolated (with $k = 31$). Therefore, in order to assess the performance of DISCOSNP on this dataset in terms of recall, these 30 SNPs were used as a reference. The 24 read sets were downloaded from the NCBI Sequence Read Archive (with the accession number SRA054922), and processed to remove barcode and adapter sequences as in the initial study.

DISCOSNP was run independently on populations E1, E2 and E3. For each population, DISCOSNP was applied on the eight read sets corresponding to the eight time points, with the default parameters and $c = 11$.

Details about datasets and applied commands are provided in Supplementary Additional File 1.

Tick dataset

DISCOSNP was applied on real sets of reads as part of a population genetic study of the tick *Ixodes ricinus* (2). DNA was extracted from two tick pools, one composed of ten individuals from Gardouch (close to Toulouse), France, and another composed of 20 individuals from Malville (close to Nantes), France. A genomic reduction was applied on these two pools by selecting the piece of an agarose gel containing DNA fragments, with a length of 500–600 nucleotides within the smear obtained following the enzymatic digestion by MseI of genomic DNA. This reduction corresponded to 3.8% of the initial genome. The DNA was sequenced by 454 Roche pyrosequencing, leading to the generation of 1 389 201 reads in two libraries (730 482 for one and 658 719 for the second). After quality trimming (where reads or ends of reads with a PHRED quality score inferior to 20 or that did not contain the expected restriction site were deleted), a total of 996 508 reads (536 061 for the first pool and 460 447 for the second) with an average length of 529 bp were used for analysis with DISCOSNP.

In order to be able to design efficient primers, detected SNPs were then selected for experimental validation using the following criteria : (i) SNPs with a coverage between 4 and 10 (126 567 SNPs) were selected to avoid sequencing errors and repeated sequences, highly frequent in ticks (66% of the genome is repeated (20)), (ii) SNPs had to be distant from homopolymers (the bubble sequences should not contain any window of 8 nucleotides with at least six identical nucleotides), and they must not be closely located (less than k bp) to any other variants such as indels or other SNPs (reads were mapped to the bubble sequences using GASSST (21) with a similarity threshold set at 80%, bubbles with at least one read mapped with differences were excluded) and (iii) SNPs with PHRED sequence quality <30 were filtered out. As in this study framework one is not interested by SNPs discriminating the two datasets, we did not use the Phi coefficient for selecting SNPs for experimental validation. Among the 1768 SNPs meeting these criteria, 384 were randomly picked for genotyping validation.

Primers were designed for each selected SNP. To validate them, we performed a genotyping run using *Fluidigm* technology, where primers are combined with fluorochrome (VIC or FAM for each allele) (22). Reading the fluorescence allows to determine the genotype of the individual typed at each locus (homozygous XX or YY, heterozygous XY).

RESULTS

We propose experiments that aim at (i) assessing the quality of DISCOSNP results on simulated datasets, in comparison with state-of-the-art reference-free SNP detection methods (including a *hybrid* approach); (ii) showing how DISCOSNP performs on real data, with biological validation. In addition, the computational resources (time and memory) required by DISCOSNP are compared with those required by the other tools.

Results were obtained with DISCOSNP, version 1.2.1 available online, as well as with the latest versions of NIKS, BUBBLEPARSE (released on April 2013) and CORTEX (CORTEX_VAR v1.0.5.21). The tests were performed on the *GenOuest* (genouest.org) cluster, composed by Intel Xeon© core processors with speed varying between 2 and 2.8 GHz.

Detection of isolated SNPs from two to 30 haploid datasets simulated from *E. coli*

We simulated 30 read sets from *E. coli* genomes, in which SNPs were inserted in order to mimic a realistic allele frequency spectrum of several individuals (see 'Materials and Methods' section for details). Below, we present the results obtained by different tools on (i) two haploids, and on (ii) three to 30 haploids, with respect to the set of isolated SNPs.

Results on two haploids. Results of DISCOSNP applied with default parameters on 2 of the 30 haploid genomes serve as proof of concept of the approach. Results reach high precision and recall: 98.81% of predicted SNPs are true positives, while 97.31% of simulated isolated SNPs were recovered. Moreover, these results were obtained in 3 min 49 s, and needed no more than 7MB of RAM memory. Results presented in Table 1 show that, on a simple dataset composed by two bacterial genomes, all tools give similar results in terms of precision/recall (except for KISSNP and NIKS, all results have precision $\approx 99.19 \pm 0.66\%$ and recall $\approx 96.56 \pm 1.41\%$). However, DISCOSNP runs faster than other tools while being by far the one needing the smallest amount of memory, by at least three orders of magnitude.

Results on more than two haploids. When applied on more than two datasets, DISCOSNP still produces high quality results, with precision reaching 99.83%, and recall decreasing only slightly (94.29%) for 30 individuals (Figure 5a). Notably this shows that DISCOSNP results remain constant regardless of the number of input datasets, unlike CORTEX whose recall and precision drop for more than two datasets. Except CORTEX, none of the other *de novo* tools could be run on more than two datasets. As shown in Figure 5b, DISCOSNP runs faster than CORTEX, while needing much less memory: applied on 30 individuals, CORTEX requires 3 h 38 s and 9658MB of memory while DISCOSNP runs in 37 min and requires 9MB. For all methods, running time grows linearly with respect to the number of read sets, but with a smaller coefficient for DISCOSNP (see Supplementary Figure S3 of Supplementary Additional File 1).

Furthermore, in Figure 5a and in b, we present the results obtained by the hybrid approach when assembling one of the 30 read sets for creating a reference sequence (with SOAPdenovo2 (23)) and then mapping the read sets to this

Table 1. Comparative results of several tools discovering SNPs between two haploid bacterial datasets

Tool	Precision	Recall	Time (s)	Memory (GB)
KISSNP	98.03	90.58	1234	50.6
NIKS	77.50	72.11	67217	86
BUBBLEPARSE	98.53	97.98	980	12.6
CORTEX	99.85	95.15	289	9.5
DISCOSNP	98.81	97.31	229	0.007
hybrid assembly	97.79	98.36	430	6.3

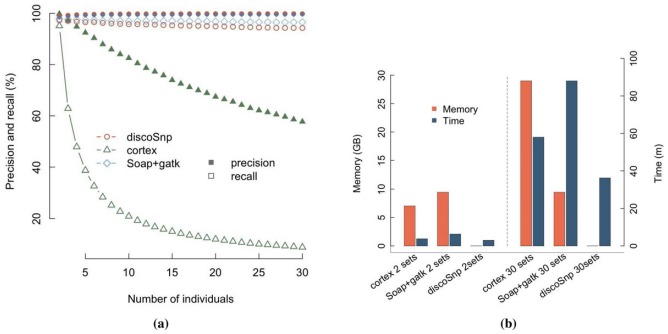


Figure 5. DISCOSNP, CORTEX and hybrid strategy (soap + gatk) results, depending on the number of input haploid individuals. Soap and gatk were launched with default parameters. For DISCOSNP and CORTEX, *k*-mers having three or fewer occurrences in all datasets were removed. (**a**) Precision and recall: filled symbols represent the precision and empty symbols represent the recall. (**b**) Time and memory performances for two (left part) and 30 (right part) individuals.

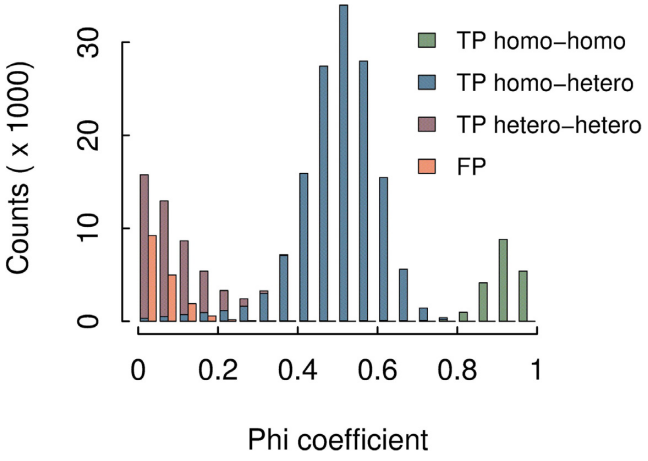


Figure 6. Repartition of SNPs detected by DISCOSNP depending of their phi coefficient. FP are false positives and TP are true positives. Homo (resp. hetero) stands for homozygous (resp. heterozygous) SNPs. True positive SNPs are then classified according to their genotype in the two individuals.

reference (with Bowtie 2 (24)). Finally, SNPs were called with GATK (25). The results are similar to DISCOSNP ones, with a slightly better recall and a slightly worse precision. In this particular case, for which the reference dataset is small, complete, well covered and easy to assemble, a hybrid method can be preferred to the *de novo* ones, as long as the time and the memory footprints are not limiting.

Detection of isolated SNPs from diploids, simulated from human chromosome 1

In this section, we present various results obtained by DISCOSNP on a more complex genome, namely diploid read datasets simulated from two human chromosome 1 individuals, to discover both heterozygous and homozygous SNPs. As expected, when the complexity and the repeat content of the input genome increase, DISCOSNP makes more erroneous calls and misses more real SNPs. Whereas precision remains reasonable, with $<3\%$ of false positive calls, recall is more impacted with 72% of the isolated SNPs that are recovered.

Among the 72 518 SNPs that are missed, 86% fall in repeated regions of human chromosome 1 (as masked by RepeatMasker from UCSC Genome Browser). In order to increase the recall and detect some of the SNPs located in repeated regions, we can change the filtering parameter that controls the branching features of the detected bubbles. By default, only clean bubbles without any branching are kept. Allowing all kinds of branching bubbles (option −b 2) would lead to recover almost all simulated SNPs but at a cost of millions of false positive calls and a precision close to zero. As a compromise, when allowing only some of the branching bubbles, those without any symmetrical branchings (option −b 1), recall reaches 79.22%, allowing to detect some of the SNPs that were simulated inside repeated regions of the chromosome. This shows that symmetrically branching bubbles are the most common type of branching bubbles and are mainly false positives. Interestingly, among the few symmetrically branching bubbles that correspond to real SNPs, 90% are located in highly repeated regions such as transposable elements, and specifically SINE elements (59%).

The precision obtained when using the −b 1 parameter is slightly lower, 92.33%, with 17 153 false positives. We observed that a large majority (78.8%) of these false calls are inexact repeats of size at least $2k-1$ contained in the chromosome 1, with both paths of the bubble mapping exactly to the non-mutated chromosome. Consequently, these bubbles would appear in any individual and could be easily filtered out when comparing coverage values between individuals. This is, in fact, the purpose of the phi coefficient, to rank the predictions according to how discriminant their coverages are between individuals, leaving almost all false positives with the lowest ranks. As shown in Figure 6, 99.7% of SNPs ranked with a phi value >0.2 are real SNPs. This filter will remove however most of the SNPs that are heterozygous in both individuals, but for SNPs that are discriminant between the individuals, i.e. for which at least one individual is homozygous, the recall stays high (above 77.8 % over such SNPs).

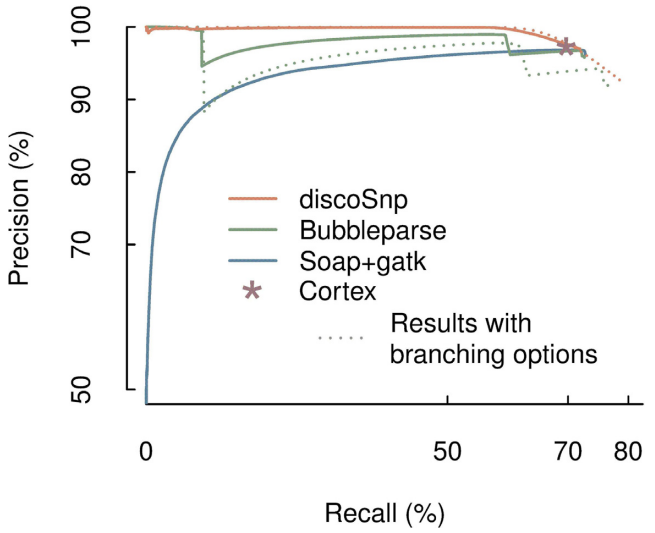


Figure 7. Comparative results of DISCOSNP, CORTEX, BUBBLEPARSE and the *hybrid* SOAPdenovo2 + Bowtie2 + GATK approaches on the two diploid human chromosome 1 dataset. Precision versus recall curves are obtained by ranking the predicted SNPs. Each data point is obtained at a given rank threshold, where precision and recall values are computed for all SNPs with better ranks than this threshold. In this framework CORTEX does not rank the predicted SNPs, its results are thus represented by a single point. Plain lines for DISCOSNP and BUBBLEPARSE were obtained while discarding all branching bubbles (options −b 0 and depth = 0 respectively), whereas dotted lines were obtained when allowing for some branchings (options −b 1 and depth = 1 respectively).

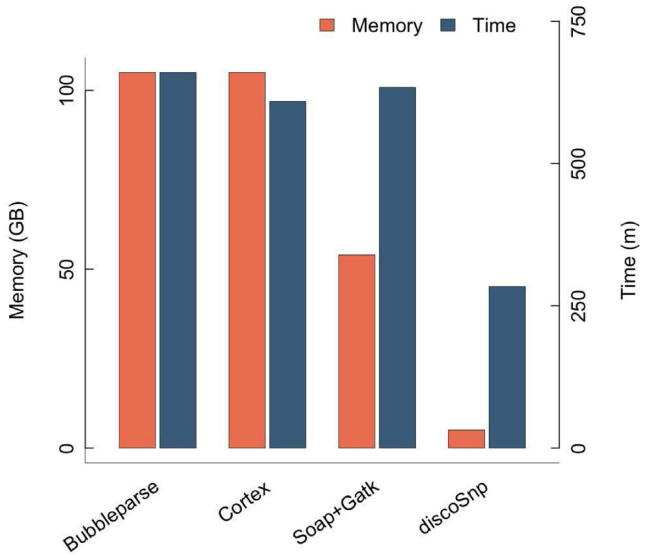


Figure 8. Comparative memory and time performances on the human chromosome 1 dataset. Time values are given with options *depth* = 1 for BUBBLEPARSE and −*b* 1 for DISCOSNP.

If the user is interested specifically in non discriminant polymorphisms, or in the case when only one individual is analyzed, we recommend to use an alternative filter based on the left and right unambiguous extension lengths (see 'Materials and Methods'). Long unambiguous extensions reveal SNPs that are isolated from other polymorphisms, while short ones reveal repeated regions or regions with high densities of polymorphism. Consequently, false positives have smaller unambiguous extension sizes (median size of 24 bp versus 527 bp for true positives).

In addition to state-of-the-art *de novo* SNP detection tools (CORTEX and BUBBLEPARSE), we compared DISCOSNP to a *hybrid* strategy consisting of three steps: *de novo* assembly using SOAPdenovo2 (23), mapping with Bowtie 2 (24) and SNP calling using GATK (25). Qualitatively, the precision and recall values remain comparable between all tested methods on this dataset. Table 2 shows that, with *stringent* parameters ($d = 0$ for BUBBLEPARSE and $b = 0$ for DISCOSNP), all tools have a precision of $96.50 \pm 0.72\%$ and a recall of $71.20 \pm 1.50\%$. With more sensitive parameters ($d = 1$ for BUBBLEPARSE and $b = 1$ for DISCOSNP), DISCOSNP achieves the best precision/recall compromise, notably with the highest recall value (79.22%).

Figure 7 shows a precision–recall curve for each software, according to its own ranking method. Unlike DISCOSNP and Bubbleparse, the hybrid approach gives high ranks to many false positive predictions. We therefore conclude that ranks are not a viable indicator to filter out the false positives for this method. The DISCOSNP ranking approach enables to reach a precision of ∼100% for a high (≈60%) recall range, slightly outperforming Bubbleparse.

As shown in Figure 8, an important benefit of DISCOSNP stands in the computational resources that are needed. DISCOSNP is at least twice faster than any other compared tool, while its memory needs are lower by several orders of magnitude than all other approaches.

Finally, it can be of interest to detect SNPs from one unique set of reads representing an individual or a pool of individuals, as this is the case, for instance, in the tick study presented below. Therefore, we performed similar tests while searching for heterozygous SNPs from one simulated set of reads sequenced from a single diploid individual. Conclusions of this study are similar to previous ones as, with the default and most stringent filtering parameters, 67% of isolated SNPs are recovered, with roughly the same amount of false positive calls (precision of 94.1%).

Results on real data: detection of SNPs from two mouse strains

To analyze the behavior of DISCOSNP on real data, we detected SNPs between two publicly available mouse datasets produced by an Illumina GA2 sequencer. Taken together, these datasets contain 2.88 billion of 100-bp reads. The first one was generated from the *FVB/NJ* mouse inbred strain, while the second one was generated from the *C57BL/6NJ* reference line. A previous study by Wong *et al.* (18) mapped the reads from the *FVB/NJ* strain to the *C57BL/6J* reference sequence, and detected ≈5 million homozygous SNPs, among which 2 472 456 are *isolated* SNPs. In the following, we refer to their set of isolated SNPs as *IS*.

DISCOSNP was run with $k = 31$ and $c = 5$ on the *FVB/NJ* and *C57BL/6NJ* read sets, and discovered 2 065 833 isolated SNPs. As presented in Figure 9 (left), 84.3% of these SNPs are also present in *IS*, while 70.1% of the *IS* set is detected by DISCOSNP. As the study was performed on inbred strains, only homozygous SNPs are expected. Therefore, we can take advantage of the Phi coefficient to further

Table 2. Results obtained by several tools with several parameter configurations on two human chromosome 1 diploid datasets

Tool	Precision (%)	Recall (%)	Time	Memory (GB)
DISCOSNP*b*0	96.98	71.99	**4h24**	**5**
CORTEX	**97.22**	69.70	10h09	105
BUBBLEPARSE*d*0	95.78	72.71	11h24	105
DISCOSNP*b*1	92.33	**79.22**	4h44	**5**
BUBBLEPARSE*d*1	91.66	76.60	11h00	105
hybrid strategy	96.18	72.86	10h34	54

The upper part of the table shows results obtained by *de novo* tools with stringent parameters (no branching allowed in bubbles). Below are the results obtained with the same tools but with more sensitive parameters, namely allowing for some branchings in bubbles. The bottom line shows results obtained with a hybrid strategy (SOAPdenovo2 + Bowtie2 + GATK), filtering out low covered SNPs (with the same parameters as those applied for all the other tools, by removing SNPs whose both alleles have coverage below four). As presented in Supplementary Additional File 1, results are worse without this filter.

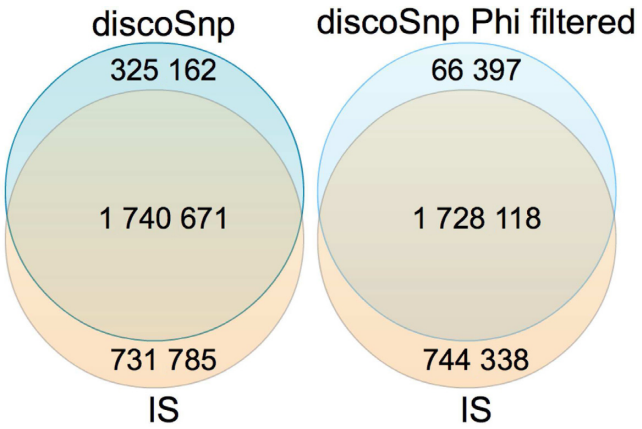


Figure 9. Venn diagrams of isolated SNPs detected by Wong *et al.* (18) (*IS* set) and by DISCOSNP. **Left**: Raw DISCOSNP results. **Right**: Filtered DISCOSNP results, i.e. SNPs with Phi ≥0.2.

filter DISCOSNP predictions. By removing SNPs for which the Phi coefficient is ≤0.2, the number of predicted SNPs drops to 1 794 515. As depicted in Figure 9 (right), 96.3% of the filtered SNPs are also in *IS*, while roughly the same proportion of *IS* SNPs are found, compared to the unfiltered results (70.0%). Therefore, by keeping only SNPs with a Phi coefficient greater than 0.2, we discard almost exclusively SNPs that are found only by DISCOSNP, and which might be false positives.

Note that in this experiment, we can not use the terms 'precision' and 'recall' for describing results quality. Indeed, the *IS* dataset was obtained via a mapping approach, and thus it can neither be considered as an exhaustive, nor a perfect list of isolated SNPs.

It is worth stressing that the results of Wong *et al.* were obtained by running a complex pipeline, involving a high-quality reference genome, 6 distinct tools, followed by a filtering step composed of 14 non-automated filters (coverage, quality, genotype, etc.). On the other hand, DISCOSNP did not require any third-party tool, or any manual tuning.

Finally, this experiment showed that DISCOSNP scales remarkably well to large amounts of data. The KISSNP2 module required 34 h and 4.5GB of memory. The KISSREADS module, which assesses the average quality and coverage of the results, took 78.5 h and 5.7GB memory. In comparison, NIKS, KISSNP, CORTEX and BUBBLEPARSE exceeded the memory limit on a server with 512GB RAM.

Evaluating DISCOSNP recall on a validated *Saccharomyces cerevisiae* SNP set

Using a set of biologically validated SNPs predicted from an artificial evolution study on *S. cerevisiae* (19), DISCOSNP recall could be evaluated on real read datasets. Among the 30 reference validated isolated SNPs, 28 were predicted by DISCOSNP, thus giving an estimated recall of 93.3%.

The two SNPs were correctly predicted by using the −b 2 DISCOSNP option that reports all bubbles including branching ones. This suggests that these SNPs may be located in complex regions of the yeast genome.

Overall, this experiment demonstrates the good performances of DISCOSNP at discovering SNPs from pooled samples, even those with low allelic frequencies: most of the reference SNPs were reported in the initial study with a MAF <20%. Note that no SNP with a MAF <10% was experimentally validated, so we could not assess the recall on these very low frequency SNPs.

Use case example with experimental validation on SNPs in the *Ixodes ricinus* genome

We conducted a study on real data, including an experimental validation on SNPs selected from DISCOSNP predictions. This was part of a population genetic study on the tick species *I. ricinus*, the main vector species of human and animal vector-borne diseases in Europe. Given the stake of tick-borne diseases in public health (26,27), it is necessary to have an accurate description of the genetic variability within and among populations of ticks, with the aim of developing efficient control methods against this vector. To this end, highly resolutive genetic markers, like SNPs, provide particularly valuable information to estimate genetic variability and also to estimate the dispersal and genetic structure of tick populations.

No genomic resources, nor a reference genome, were available until now for this species. This study fits a case in which DISCOSNP is useful as (i) sequenced material exists but no reference genome is available and (ii) one is interested in detecting a small set of highly confident heterozygous SNPs. Therefore, DISCOSNP was applied on a 454 read set obtained from pooling and sequencing of several tick individuals isolated from natural populations (2).

DISCOSNP detected 321 088 SNPs of which 384 were selected, according to their minimal and maximal coverage and context sequences for experimental validation (see 'Ma-

terials and Methods' section). Note that as in this context there is no need to discriminate SNPs between conditions, the Phi coefficient was not used. Primers were designed for each selected SNP and 464 individuals were then genotyped for these 384 SNPs using the *Fluidigm* technology. Among them, 368 SNPs (95.8%) were retrieved with a minor allele frequency varying between 0.04 and 0.5, with a mean value of 0.23. Of the remaining 16 SNPs, 5 SNPs were not amplified and 11 presented only one of the two alleles.

DISCUSSION

DISCOSNP is robust with respect to input parameters, and easy to use. The input of the software is an ordered list of any number of raw read dataset(s) (`fasta` or `fastq`, `gzip` compressed or not), and two parameters (*k*-mer size and minimal coverage threshold). These two parameters have limited impact on results quality (see Supplementary Additional File 1, Figures S1 and S2), as long as they are coherent with respect to the input data (read length and approximate coverage). The output is a `fasta` file composed of sequences containing ranked SNPs, together with their coverage and average `PHRED` quality.

As there exists no exhaustive and perfect list of the isolated SNPs present in a real dataset, a large part of the results discussed in this paper were obtained on simulated data. However, we paid special attention to the simulations being realistic. The use of simulated data allowed us to evaluate the correctness of our method in a controlled environment, and to analyze false positive and false negative calls. This highlighted that the main source of wrong calls is the presence of repeated sequences in the genomes.

Two distinct strategies may be adopted for distinguishing SNPs from approximate repeats. On complex genomes, which are repeat-prone, the choice of the strategy influences the results. Removing all branching bubbles (−b 0 option) is a good way to obtain high confident coverage (precision of ≈97% on human datasets) at the expense of missing some SNPs located into repeats. On the other hand, accepting simply branching bubbles, i.e. non-symmetrical (−b 1 option), enables to detect more SNPs that are located in repeated regions, but leads to an increase of the number of false positives and to the detection of non-isolated SNPs. Therefore, the choice between these two strategies is determined by the level of complexity of the genome and by the specific needs of the user.

Importantly, SNPs can be ranked according to the Phi coefficient, enabling to highlight true SNPs that discriminate conditions and to discard putative false positives due to repeats. Such feature can be precious for many biological studies, where one is often interested in finding a subset of high confidence SNPs that have opposite or only different allele frequencies between individuals or pooled samples. This was the case, for instance, when comparing two inbred mouse strains. Moreover, our simulations on a human chromosome show that, by removing low-ranked SNPs, the recall of discriminant SNPs drops only by 1.3%, while the precision increases from 92.3 to 99.7%.

A distinctive advantage of our method is its extremely low memory usage. For instance, in the mouse study cited above, nearly 3 billion reads (100 bp) were analyzed by DISCOSNP, using at most 5.7GB of memory. This is not at the expense of prohibitive running times, as DISCOSNP stays faster than all other known *de novo* SNP detection tools. DISCOSNP is usable on the standard desktop computer of any biological lab, enabling studies that were not possible with other available *de novo* SNP detection tools (that require at least two orders of magnitude more memory).

One limitation of DISCOSNP is that it cannot find the genomic locations of SNPs. However, in numerous biological applications, the localization of the polymorphisms is not required. For instance, DISCOSNP can be applied to identify SNPs associated to some phenotypic traits or diseases. Another limitation of DISCOSNP is induced by the fact that it focuses on isolated SNPs. If this kind of SNPs are well suited for designing markers, they do not represent the full SNP diversity and thus, they cannot be used directly for statistical downstream studies as phylogeny reconstruction, or estimation/comparison of the genetic diversity among or between natural populations. However, sequences obtained around those SNPs of interest can be genotyped at larger population scales with standard genotyping technologies or used for diagnostic assays. This was actually the case for the tick study, where natural populations were genotyped to characterize their reproductive mode (level of inbreeding) and to estimate the gene flow within and among populations at various spatial scales. Moreover, SNPs discovered by DISCOSNP and selected with respect to primer hybridization features, are currently used to build a genetic map based on the analysis of the segregation of parental alleles in the offspring of several controlled crosses.

A future development will consist in integrating *de novo* SNP, and possibly other polymorphisms such as indels and structural variants detection tools (28), with *de novo* assembly. This solution would unite the power of both approaches, facilitating the assembly by tackling the polymorphism problem and by conserving the recall and precision performances of methods such as DISCOSNP. This idea would lead to assembled genomes represented no more as 'simple' linear sequences but as graphs such as suggested by the `fastg` format http://fastg.sourceforge.net/, in which polymorphisms are conserved. Such a change would open the way to new possibilities of storage and use of polymorphisms.

CONCLUSION

DISCOSNP is an integrated reference-free software designed to find SNPs that can be subsequently used as high-quality genetic markers. DISCOSNP combines several advantages: (i) robust detection and ranking of isolated SNPs, (ii) support for any number of read datasets, (iii) scaling to big data studies thanks to a highly memory efficient data structure, (iv) lower running times than other reference-free tools and (v) an easy to use software, capable of processing billions of reads from a mammalian genome with a single command.

The experiments on a dataset composed of two simulated diploids show that DISCOSNP provides similar results, both in terms of precision and recall, to those of other state-of-the-art reference-free SNP detection methods, while being faster and needing at least two orders of magnitude less memory. Experiments on simulated haploids show that,

when analyzing together more than two individuals, DISCOSNP outperforms the other tools, both in terms of computational resources and results quality. Applied on real datasets, results confirm the capacity of DISCOSNP to scale-up to large volumes of data (less than 6GB memory on 3 billion Illumina reads), as well as its high precision, i.e. an experimental validation conducted on an arthropod species (the tick *I. ricinus*) on which *de novo* sequencing was performed, confirmed 96% of the predicted SNPs that were tested.

The DISCOSNP source code, available under CeCILL license, can be downloaded from http://colibread.inria.fr/discoSnp/. Moreover, this web page shows how to integrate DISCOSNP in any Galaxy instance using the *GenOuest Tool Shed*.

SUPPLEMENTARY DATA

Supplementary Data are available at NAR Online.

ACKNOWLEDGEMENTS

The authors warmly thank Zamin Iqbal and Richard Leggett for interesting discussions and valuable help on using CORTEX and BUBBLEPARSE tools. We also thank the *GenOuest* (genouest.org) cluster team, as well as the *Bordeaux Bioinformatics Center* (http://www.cbib.u-bordeaux2.fr/en) team, who allowed us to perform all the tests.

FUNDING

French ANR-12-BS02-0008 Colib'read project; SOFIPROTEOL under the FASO (Le Fonds dAction Stratégique des Oléoprotéagineux) project 'PEAPOL'. Funding for open access charge: Inria.
Conflict of interest statement. None declared.

REFERENCES

1. Xu,X., Liu,X., Ge,S., Jensen,J. D., Hu,F., Li,X., Dong,Y., Gutenkunst,R. N., Fang,L., Huang,L. *et al.* (2012) Resequencing 50 accessions of cultivated and wild rice yields markers for identifying agronomically important genes. *Nat. Biotechnol.*, **30**, 105–111.
2. Quillery,E., Quenez,O., Peterlongo,P. and Plantard,O. (2014) Development of genomic resources for the tick Ixodes ricinus: isolation and characterization of single nucleotide polymorphisms. *Mol. Ecol. Resour.*, **14**, 393–400.
3. DePristo,M.A., Banks,E., Poplin,R., Garimella,K.V., Maguire,J.R., Hartl,C., Philippakis,A.A., del Angel,G., Rivas,M.A., Hanna,M. *et al.* (2011) A framework for variation discovery and genotyping using next-generation DNA sequencing data. *Nat. Genet.*, **43**, 491–498.
4. Li,H., Handsaker,B., Wysoker,A., Fennell,T., Ruan,J., Homer,N., Marth,G., Abecasis,G. and Durbin,R. (2009) The sequence alignment/map format and SAMtools. *Bioinformatics*, **25**, 2078–2079.
5. Li,H. (2012) Exploring single-sample snp and indel calling with whole-genome de novo assembly. *Bioinformatics*, **28**, 1838–1844.
6. Bradnam,K., Fass,J., Alexandrov,A., Baranay,P., Bechner,M., Birol,I., Boisvert,S., Chapman,J., Chapuis,G., Chikhi,R. *et al.* (2013) Assemblathon 2: evaluating de novo methods of genome assembly in three vertebrate species. *GigaScience*, **2**, 10.
7. Willing,E.-M., Hoffmann,M., Klein,J.D., Weigel,D. and Dreyer,C. (2011) Paired-end RAD-seq for de novo assembly and marker design without available reference. *Bioinformatics*, **27**, 2187–2193.
8. Peterlongo,P., Schnel,N., Pisanti,N., Sagot,M.-F. and Lacroix,V. (2010) Identifying SNPs without a reference genome by comparing raw reads. In: *String Processing and Information Retrieval*. Springer, pp. 147–158.
9. Iqbal,Z., Caccamo,M., Turner,I., Flicek,P. and McVean,G. (2012) De novo assembly and genotyping of variants using colored de Bruijn graphs. *Nat. Genet.*, **44**, 226–232.
10. Leggett,R.M., Ramirez-Gonzalez,R.H., Verweij,W., Kawashima,C.G., Iqbal,Z., Jones,J.D.G., Caccamo,M. and MacLean,D. (2013) Identifying and classifying trait linked polymorphisms in non-reference species by walking coloured de Bruijn graphs. *PLoS ONE*, **8**, e60058.
11. Nordström,K.J.V., Albani,M.C., James,G.V., Gutjahr,C., Hartwig,B., Turck,F., Paszkowski,U., Coupland,G. and Schneeberger,K. (2013) Mutation identification by direct comparison of whole-genome sequencing data from mutant and wild-type individuals using k-mers. *Nat. Biotechnol.*, **31**, 325–330.
12. Sacomoto,G.A., Kielbassa,J., Chikhi,R., Uricaru,R., Antoniou,P., Sagot,M.-F., Peterlongo,P. and Lacroix,V. (2012) KISSPLICE: de-novo calling alternative splicing events from RNA-seq data. *BMC Bioinformatics*, **13**, S5.
13. Gardner,S.N. and Hall,B.G. (2013) When whole-genome alignments just won't work: kSNP v2 software for alignment-free SNP discovery and phylogenetics of hundreds of microbial genomes. *PLoS ONE*, **8**, e81760.
14. Zerbino,D.R. and Birney,E. (2008) Velvet: algorithms for de novo short read assembly using de Bruijn graphs. *Genome Res.*, **18**, 821–829.
15. Chikhi,R. and Rizk,G. (2012) Space-efficient and exact de Bruijn graph representation based on a Bloom filter. In: *Lecture Notes in Computer Science* , Vol. **7534**. WABI , pp. 236–248.
16. Salikhov,K., Sacomoto,G. and Kucherov,G. (2013) Using cascading Bloom filters to improve the memory usage for de Brujin graphs. In: *Algorithms in Bioinformatics*. Springer, pp. 364–376.
17. The Assemblathon 2 Consortium. (2013) Assemblathon 2: evaluating de novo methods of genome assembly in three vertebrate species. *GigaScience*, **2**, 10.
18. Wong,K., Bumpstead,S., Van Der Weyden,L., Reinholdt,L.G., Wilming,L.G., Adams,D.J. and Keane,T.M. (2012) Sequencing and characterization of the FVB/NJ mouse genome. *Genome Biol.*, **13**, R72.
19. Kvitek,D.J. and Sherlock,G. (2013) Whole genome, whole population sequencing reveals that loss of signaling networks is the major adaptive strategy in a constant environment. *PLoS Genet.*, **9**, e1003972.
20. Ullmann,A., Lima,C., Guerrero,F., Piesman,J. and Black,W. (2005) Genome size and organization in the blacklegged tick, Ixodes scapularis and the Southern cattle tick, Boophilus microplus. *Insect Mol. Biol.*, **14**, 217–222.
21. Rizk,G. and Lavenier,D. (2010) GASSST: global alignment short sequence search tool. *Bioinformatics*, **26**, 2534–2540.
22. Wang,J., Lin,M., Crenshaw,A., Hutchinson,A., Hicks,B., Yeager,M., Berndt,S., Huang,W.-Y., Hayes,R., Chanock,S. *et al.* (2009) High-throughput single nucleotide polymorphism genotyping using nanofluidic dynamic arrays. *BMC Genomics*, **10**, 561.
23. Luo,R., Liu,B., Xie,Y., Li,Z., Huang,W., Yuan,J., He,G., Chen,Y., Pan,Q., Liu,Y. *et al.* (2012) SOAPdenovo2: an empirically improved memory-efficient short-read de novo assembler. *GigaScience*, **1**, 18.
24. Langmead,B. and Salzberg,S.L. (2012) Fast gapped-read alignment with Bowtie 2. *Nat. Methods*, **9**, 357–359.
25. McKenna,A., Hanna,M., Banks,E., Sivachenko,A., Cibulskis,K., Kernytsky,A., Garimella,K., Altshuler,D., Gabriel,S., Daly,M. *et al.* (2010) The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Res.*, **20**, 1297–1303.
26. Gubler,D.J. (1998) Resurgent vector-borne diseases as a global health problem. *Emerg. infect. Dis.*, **4**, 442.
27. Parola,P. and Raoult,D. (2001) Tick-borne bacterial diseases emerging in Europe. *Clin. Microbiol. Infect.*, **7**, 80–83.
28. Lemaitre,C., Ciortuz,L. and Peterlongo,P. (2014) Mapping-free and assembly-free discovery of inversion breakpoints from raw NGS reads. In: *Lecture Notes in Computer Science* , Vol. **8542**. AlCoB , pp. 119–130.

Development of genomic resources for the tick Ixodes ricinus : isolation and characterization of single nucleotide polymorphisms

MOLECULAR ECOLOGY
RESOURCES
Molecular Ecology Resources (2013) doi: 10.1111/1755-0998.12179

Development of genomic resources for the tick *Ixodes ricinus*: isolation and characterization of single nucleotide polymorphisms

E. QUILLERY,*† O. QUENEZ,‡ P. PETERLONGO§ and O. PLANTARD*†

*INRA, UMR1300 Biology, Epidemiology and Risk Analysis in animal health, BP 40706, F-44307 Nantes, France, †LUNAM Université, Oniris, Ecole nationale vétérinaire, agroalimentaire et de l'alimentation Nantes Atlantique, UMR BioEpAR, Nantes 44307, France, ‡GenOuest Core Facility, INRIA Rennes Bretagne-Atlantique, IRISA, Rennes 35042, France, §GenScale, INRIA Rennes Bretagne-Atlantique, IRISA, Rennes 35042, France

Abstract

Assessing the genetic variability of the tick *Ixodes ricinus*—an important vector of pathogens in Europe—is an essential step for setting up antitick control methods. Here, we report the first identification of a set of SNPs isolated from the genome of *I. ricinus*, by applying a reduction in genomic complexity, pyrosequencing and new bioinformatics tools. Almost 1.4 million of reads (average length: 528 nt) were generated with a full Roche 454 GS FLX run on two reduced representation libraries of *I. ricinus*. A newly developed bioinformatics tool (DiscoSnp), which isolates SNPs without requiring any reference genome, was used to obtain 321 088 putative SNPs. Stringent selection criteria were applied in a bioinformatics pipeline to select 1768 SNPs for the development of specific primers. Among 384 randomly SNPs tested by Fluidigm genotyping technology on 464 individuals ticks, 368 SNPs loci (96%) exhibited the presence of the two expected alleles. Hardy–Weinberg equilibrium tests conducted on six natural populations of ticks have shown that from 26 to 46 of the 384 loci exhibited significant heterozygote deficiency.

Keywords: 454, *de novo* SNP calling, *Ixodes ricinus*, nonmodel organism, reduced representation library, SNP

Received 31 May 2013; revision received 17 September 2013; accepted 20 September 2013

Introduction

Ticks (Acari, Ixodidae) are haematophagous vectors of numerous pathogens. In Europe, *Ixodes ricinus* is the most widespread species and is responsible for both human and animal diseases (Gubler 1998; Parola & Raoult 2001). This tick transmits pathogenic agents responsible for zoonotic diseases such as Lyme disease, tick-borne-encephalitis, babesioses or rickettsioses (Gray 2002).

An accurate knowledge of tick dispersal and genetic diversity is required to set-up efficient vector control methods, such as acaricides or antitick vaccines, but is still lacking (Philipp *et al.* 1997; Gillet *et al.* 2009). However, all three sets of microsatellite markers developed independently to date (6 by Delaye *et al.* 1998; 17 by Røed *et al.* 2006 and 9 by Noel *et al.* 2012) exhibit high heterozygote deficiency (De Meeûs *et al.* 2004; Røed *et al.* 2006; Kempf *et al.* 2009, 2011; Noel *et al.* 2012) and, for several loci, nonmendelian transmission patterns that could not be fully explained by the high frequency of null alleles (De Meeûs *et al.* 2004; Røed *et al.* 2006). Moreover, the large number of alleles—sometimes differing by a single nucleotide, stutter bands or short allele dominance- makes genotype assignation difficult (De Meeûs *et al.* 2004). To circumvent the difficulties associated with microsatellites, single nucleotide polymorphisms may constitute markers of choice for the investigation of genetic variability in *I. ricinus* (Noureddine *et al.* 2011; Porretta *et al.* 2013; Van Zee *et al.* 2013). SNPs have been successfully developed in other nonmodel organisms (Ekblom & Galindo 2011; Helyar *et al.* 2011). SNPs are the most abundant markers within genomes and exhibit a uniform distribution across chromosomes (Schlötterer 2004). Because of their lower mutation rates (compared with microsatellites; Estoup *et al.* 2002), SNPs exhibit less homoplasy (Morin *et al.* 2004). As they are biallelic, a lower error rate in genotyping and allele assignation can be expected. Moreover, due to their large numbers, they are very useful and informative for the investigation of genetic polymorphism (Lao *et al.* 2006; Paschou *et al.*

Correspondence: Elsa Quillery and Olivier Plantard, fax: +33(0)240 687 751; E-mails: elsa.quillery@oniris-nantes.fr and olivier.plantard@oniris-nantes.fr

2007). Fifty biallelic SNPs are considered to be equivalent to 20 highly polymorphic microsatellites (Smouse 2010).

Several methodological constraints have to be overcome during the development of SNPs in *I. ricinus*. First, because of the large estimated genome size (about 2.1 Gb based on the closely allied north American species *I. scapularis*), it is more difficult to get a higher sequencing depth. However, when looking for SNPs, a minimum depth is necessary to discriminate between true polymorphism and sequencing errors. Second, no reference genome is available for *I. ricinus*. Indeed, most of the available softwares developed to date and designed to call SNPs from NGS data sets make use of a reference genome (Li *et al.* 2009; McKenna *et al.* 2010). They map the reads on this reference genome to look for differences between this sequence and the reads (Nielsen *et al.* 2011).

In the case of *I. ricinus*, the *de novo* assembly from NGS data sets would thus be especially difficult to build because of the large size of the genome and the high proportion of repeated elements within this genome (Meyer *et al.* 2010).

We overcame the first difficulty using a method employing reduced representation libraries (RRL) to reduce genome complexity (Altshuler *et al.* 2000; Van Tassell *et al.* 2008). This method is based on the use of restriction enzymes and the selection of digested DNA fragments of a given range size. Finally, we developed an original method based on the DiscoSnp tool (R. Uricaru, G. Rizk, V. Lacroix, E. Quillery, O. Plantard, R. Chikhi, C. Lemaitre & P. Peterlongo, in prep) that can identify SNPs by comparing their reads in raw NGS data sets using a de-Bruijn graph and without any genome assembly.

Materials and methods

Tick collections and DNA extraction

The DNA libraries were constructed by sampling two *I. ricinus* populations. Only adult females (the lifecycle stage with the largest amount of DNA) were used. The T population corresponds to 10 partially engorged females that were collected on roe deer in southwest France ('Gardouch'; 43° 23′ 27.88″N, 1° 41′ 1.67″E) during the winter of 2010 and kept at −80°C until DNA extraction. The M population corresponds to 20 nonengorged females that were collected in spring 2011 (and kept alive at 4°C in the laboratory until DNA extraction) as they were questing for a host on vegetation in northwest France ('Malville'; 47° 21′ 30.10″ N, 1° 51′ 41.59″W). Each individual was extracted, and the DNA concentration measured separately.

The ticks were frozen in liquid nitrogen and crushed with a pestle in individual tubes. DNA extraction was conducted according to manufacturer instructions using the NucleoSpin Tissue XS kit (Macherey-Nagel).

For the genotyping and SNP validation, *I. ricinus* nymphs have been collected by the drag method (Schulze *et al.* 1997) in 83 different sampling sites covering an area of 130 km^2 of the « Zone Atelier Armorique» (Bretagne, France; 48° 28′ 28.25″N, 1° 33′ 49.29″W). Between 1 to 10 *I. ricinus* individuals, nymphs were extracted in each sampling sites, resulting in a total of 464 individuals. Sixteen controls were added in the genotyping assay including 'no template controls' (NTC) used for Fluidigm technology, Whole-Genome Amplification controls and DNA extraction controls.

As the amount of DNA obtained from a single individual was insufficient for genotyping of all the SNPs isolated, a Whole-Genome Amplification (WGA) using the primer extension preamplification method (PEP-PCR) (LGC-Kbio) was performed on each individual DNA extract prior to genotyping.

Genome reduction and sequencing

After testing several restriction enzymes, *MseI* (T ^ AAT; New England Biolabs) was selected because a high concentration of DNA fragments of the target size (*id est* 500–600b, to maximize the efficiency of 454 pyrosequencing relative to reads length) could be obtained, and no discrete band (that would reveal the presence of repeat elements) was observed in the target range of DNA fragments selected. Each sample was digested for 8 hours (2.5 U/μg of DNA) according to manufacturer's instructions. The digested DNA was then separated on a 1% agarose gel (4 h, 80v). A gel piece of each sample, containing DNA fragments from 500 to 600 bp (according to the 100 pb marker size ladder; Eurobio), was sheared under a UV lamp. The DNA was then extracted and cleaned with the gel clean-up kit (Macherey-Nagel), eluted in 40 μL of EB Buffer (3.33 mM Tris, pH 8.5), then quantified using Qubit™ with the dsDNA HS Assay™ kit (Invitrogen). Each sample was then sent to the Biogenouest Genomic platform (Rennes, France) where the 454 sequencing was conducted. The samples from each of the 2 populations (T and M) were pooled separately. Pools were prepared from an equimolar quantity of DNA from each of the 10 to 20 samples, respectively, corresponding to 500 ng of DNA in 10 μl, tagged with a unique barcode (multiplex identifier MID) and sequenced using the Roche 454 GS FLX and Titanium chemistry.

The reads obtained with the 454 sequencer were trimmed by default filters. Another filter was performed directly by the sequencing platform (the Biogenouest Genomic platform; scripts are available upon request on website galaxy: galaxy.genouest.org) to delete reads that (i) did not contain the four nucleotides A, C, T, G; (ii)

contained a high frequency of undetermined base (>7.0%); (iii) was less than 150 bp or greater than 950 bp in size; (iv) contained repeat motifs (cf. 'passed 1' reads in Table 1). Finally, reads (or ends of reads) with a quality inferior to 20 (PHRED Quality score) or that did not contain the expected restriction site were deleted (script available upon request from INRA MIGALE bioinformatics platform: http://migale.jouy.inra.fr), as well as the MID tags used for the pyrosequencing (cf. 'passed 2' reads in Table 1).

Identification and checking of SNPs

A pipeline for SNP calling was developed based on the DiscoSnp tool (R. Uricaru, G. Rizk, V. Lacroix, E. Quillery, O. Plantard, R. Chikhi, C. Lemaitre & P. Peterlongo, in prep). The DiscoSnp source code is available under CeCILL licence, and it can be downloaded from colibread.inria.fr/discosnp/. This tool calls SNPs from one or several reads sets without using any reference genome. The DiscoSnp tool is composed of two main modules. The first module, KisSnp2, constructs a de-Bruijn graph by extracting all words of length k (k-mers) from the reads. The de-Bruijn graph organizes the k-mers as follows: a node stores a k-mer and an oriented edge connects two nodes if the suffix of length k-1 of the source node is equal to the prefix of length k-1 of the target node. KisSnp2 then detects patterns in the graph that reveal the presence of a SNP in the read set(s). This first module output is a FASTA file containing sets of pairs of heterozygous sequences of length 2k-1. Because sequences are constructed by assembling k-mers from reads, it is necessary to map the initial reads on them. This step is performed by the second DiscoSnp module called KissReads. This step (i) removes spurious sequences not existing in reads but only in k-mers and (ii) quantifies the average PHRED quality and the read sequencing depth of each position of each sequence and for each read set.

A draft assembly was conducted with MIRA3 (Chevreux *et al.* 1999) to estimate the coverage of the *I. ricinus* genome by the sequencing of our two reduced representation libraries.

SNP assay design and genotyping

For each SNP, read alignments containing the isolated SNPs were checked visually with the Tablet software (Milne *et al.* 2013). Only biallelic SNPs were considered. SNP loci located in the vicinity of microsatellite loci were excluded because such loci are known to exhibit high mutation rates that would prevent hybridization of the primers designed for genotyping. Among the SNPs satisfying all the above criteria, 384 loci (corresponding to 4 Fluidigm chips 96 × 96) were selected at random, for which primers were designed with the Perl Primer software (Marshall 2004) with length, annealing temperature and GC content as recommended for their use with Fluidigm technology and Kaspar chemistry (Table S1, Supporting information).

The WGA was conducted by the GENTYANE platform (INRA, Clermont-Ferrand, France) like the genotyping, using the Biomark HD system (Fluidigm technology) and Kaspar chemistry (Wang *et al.* 2009).

Minor allele frequencies (MAF) were calculated on the 464 individuals genotyped (Table S1, Supporting information). Expected and observed heterozygote frequencies were calculated, and Hardy–Weinberg exact tests were computed using Genepop 4.2.1 (Rousset 2008). The tests were conducted on the six largest populations of the sample, with 10 individuals for each population.

Results

A workflow illustrating the steps involved in data processing is given in Fig. S1.

Pyrosequencing of the reduced representation libraries

454 GS FLX sequencing generated 1 389 201 reads for the two reduced representation libraries (730482 and 658 719 for populations M and T, respectively) corresponding to 527 Mbp (Fig. S2, Supporting information). The reads had a mean length of 401 nt, with an average quality score of 33.2. Most of the reads (95%) began with 'TAA ', as expected for DNA fragments digested with the *MseI*

Table 1 Summary statistics for 454 pyrosequencing of the 2 reduced representation libraries. The number of reads and their lengths are indicated in the raw data and after the first (Passed 1), and second criteria of trimming (Passed 2)

Population	Number of Individuals	Number of reads			Length of reads (Passed 2 reads)			
		Raw	Passed 1	Passed 2	Mean	Maximum	Minimum	Total nt
M	20	730 482	638 228	536 061	528	914	167	283 554 541
T	10	658 719	563 986	460 447	530	825	30	244 272 285
Total	30	1 389 201	1 202 214	996 508	529	914	30	527 826 826

enzyme. After application of the initial trimming steps, 392 693 reads were excluded due to insufficient quality score, presence of repeat sequences or absence of the restriction site.

SNPs discovery with DiscoSnp

A total of 996 508 reads (536 061 and 460 447 for populations T and M, respectively) with a mean length of 529 nt were used for the isolation of SNPs (Table 1). DiscoSnp was run on two read sets corresponding to the M and T populations. The main parameter in DiscoSnp is the k value. The best k value was determined by plotting the k-mer counting histograms (Fig. S3) for distinct k values. Finally, k = 29 was used, corresponding to the beginning of the plateau of the curve with more than 80% of unique k-mer and giving the optimal sequence size as output for DiscoSnp.

After applying the KisSnp2 module of the DiscoSnp software using k = 29, 791 803 SNPs were obtained. The KissReads module was then used to check whether the reconstructed sequences of length (2k-1) really corresponded to reads existing in the original data set. At this stage, 321 088 SNPs (corresponding to 40% of the initial number of SNPs found) were identified as true positive SNPs.

The SNPs generated by DiscoSnp were then sorted according to their sequencing depth (Fig. 1).

This curve of distribution of sequencing depth for the SNPs was similar to that observed for the sequencing depth of contigs obtained during the assembly with MIRA3 (Fig. S4, Supporting information). Only those SNPs for which the sequencing depth was between 4 and 10 were retained for subsequent analysis, corresponding to 126 567 SNPs (i.e. 39.4% of the 321 088 SNPs previously isolated). This selection was conducted to eliminate SNPs that could be due to sequencing errors (considered to be present in SNPs with a sequencing depth inferior to 4) or located in duplicated loci or repeat elements (considered to be present in SNPs with a sequencing depth superior to 10). The SNPs sequence vicinity was used to filter out SNPs close to homopolymers as these concentrate sequencing errors in 454 data and can therefore affect primer hybridization. For this, we used a sliding window of 8 nucleotides and eliminated the sequence if 6 identical nucleotides were

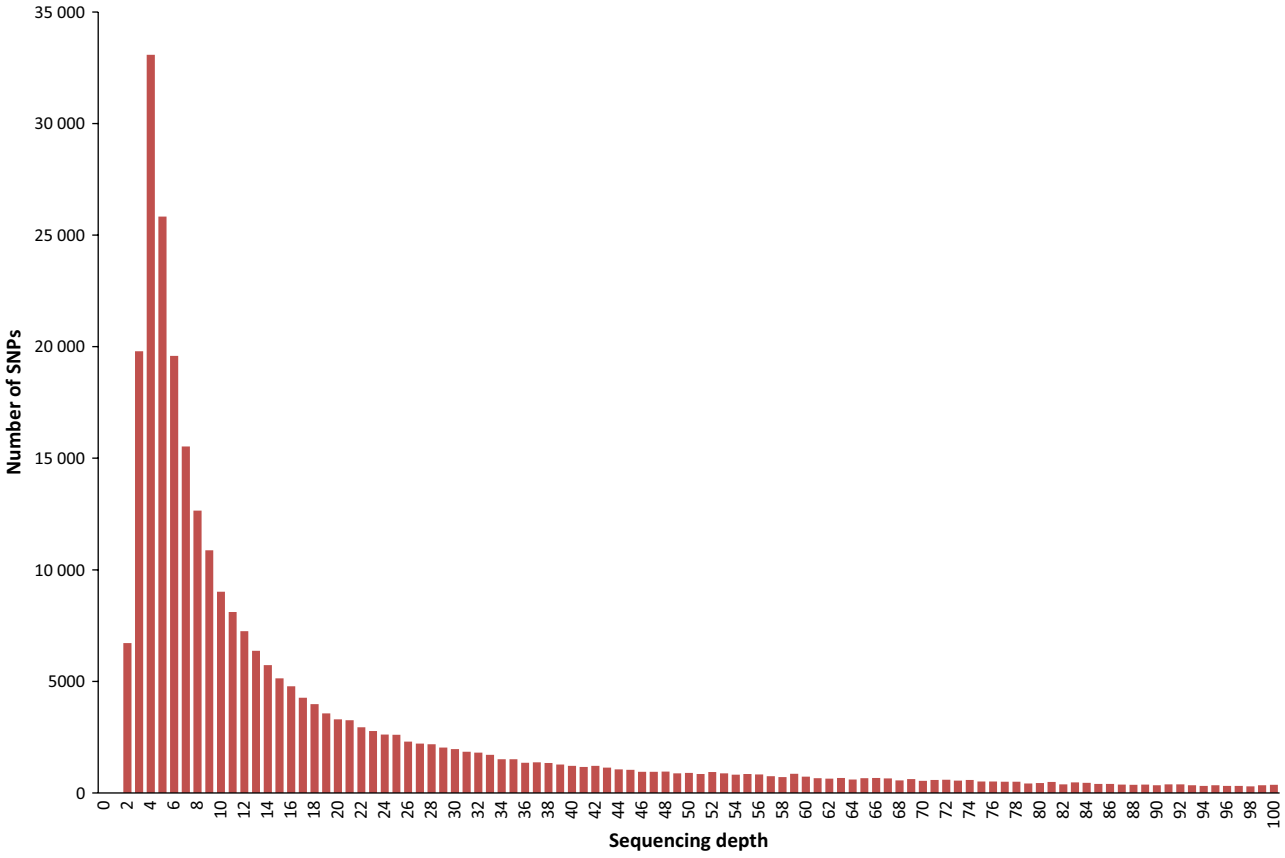


Fig. 1 Number of SNPs identified by DiscoSnp according to sequencing depth (the distribution was truncated; sequencing depths above 100 are not illustrated).

detected within this window. We also filtered out SNPs with PHRED sequence quality lower than 30. Only 9537 SNPs with a quality score superior to 30 (PHRED) and without any homopolymer in the sequence of 2k-1 were kept. We additionally filtered out SNPs in sequences that could have been generated by erroneous reads. To do this, we mapped back reads on all SNPs sequences (of length 2k-1) using GASSST (Global Alignment Short Sequence Search Tool; Rizk & Lavenier 2010) with a similarity threshold set at 80%. In this final step, SNPs for which the alignment between sequence and reads included gaps or substitutions could be detected and were filtered out. As a result, 1768 SNPs were retained.

Description of the 384 SNPs isolated

From those 1768 SNPs, 384 SNPs were selected at random. The number of SNPs selected in each of the seven sequencing depth classes (4 to 10) was proportional to the initial sequencing depth distribution observed for the 1768 SNPs. Among those 384 SNPs, 254 SNPs (66%) corresponded to transitions while 130 (34%) corresponded to transversions. Twenty-two SNPs showed some polymorphism only in the M population and 62 only in the T population, while 300 SNPs exhibited the two alleles in each of the two RRL analysed. Homology of the 384 contigs (corresponding to the consensus of reads for a given SNP locus) was investigated by BLAST using sequences already deposited in GenBank. 56.51% of the retained sequences exhibited homology with sequences from *I. scapularis* (% identity > 80% and coverage > 10%) and 0.78% with another tick species (*Rhipicephalus* (*Boophilus*) *microplus*). No other significant match with any other organism was observed.

Genotyping results

Because of the small amount of DNA available for each individual tick nymph (~17 ng), a Whole-Genome Amplification method (WGA) was applied to multiply the available DNA by 30.

Twenty Fluidigm chips (« dynamic array » 96.96 BioMark™) were used for the genotyping. One of them was not used in subsequent analyses for technical reasons. Finally, 168 378 genotyping points were analysed, of which 35 363 (21%) were not interpretable (lack of amplification or ambiguous genotype assignation) and were therefore considered as missing data.

Among the 384 SNPs, five loci showed no amplification (corresponding to 5.4% of the 35 363 missing data). Eleven SNPs among the 484 individuals investigated were only found in the homozygous state. The remaining 368 SNPs provided suitable amplification results and exhibited the two alleles. The minor allele frequency (MAF) varied between 0.04 and 0.5, with a mean value of 0.23 (Table S2, Supporting information). In the six populations investigated for Hardy–Weinberg equilibrium (those with a population size of ten individuals), from 6.77% to 11.97% of the 384 loci were found with a significant heterozygote deficiency (Table S3, Supporting information).

Discussion

Due to the numerous difficulties encountered with the three sets of microsatellite loci developed in *I. ricinus* to date (Delaye *et al.* 1998; Røed *et al.* 2006; Noel *et al.* 2012), it was urgent to produce a new set of highly resolutive genetic markers for the analysis of genetic variability in this important tick species.

The SNP discovery workflow described here permitted the identification of 321 088 putative SNPs in the *I. ricinus* genome, the successful design of primers for 384 SNP loci and polymorphism was observed in 96% of the loci.

The development of these SNPs involved the following four steps: pyrosequencing of two reduced representation libraries from a pool of *I. ricinus* individuals, isolation of SNPs from this NGS data set with the DiscoSnp pipeline, design of SNPs primers and SNPs genotyping.

To our knowledge, no inbred homozygous *I. ricinus* strain is currently available. Thus, within-individual polymorphism is expected even if a single individual is sequenced. Due to this nonreducible polymorphism, the difficulties encountered in the genome assemblage and the avoidance of sequencing errors are therefore increased. Moreover, as the amount of DNA available from a single individual is limited, several individuals must be pooled together for pyrosequencing.

The sequencing depth must be tuned according to the trade-off between the minimum number of reads for a given sequenced locus required to detect polymorphism (with a minimum of two reads) and the wastage of sequencing efforts due to sequencing of nonvariable genomic regions or regions that are already represented by a sufficient number of reads in the data set. We made use of sequencing depth to exclude sequencing errors (expected to be found in sequences represented by a single or a few reads). We also used the sequencing depth to avoid loci in repetitive DNA or duplicated loci (that are not suitable for SNP design).

An RRL strategy was employed to maximize the sequencing depth of the genomic region sequenced. After a draft assembly of our data set using MIRA3, we estimated that our libraries represented 78.3 Mpb, corresponding to a coverage of about 3.8% of the *I. ricinus* genome (considering a genome size of 2.1 Gb), with a

mean sequencing depth of 2.58 and a median of 2.3. Although this sequencing depth could be considered as weak, SNPs were subsequently only selected in reads with a sequencing depth between 4 and 10 to avoid sequencing errors and repeat loci. Moreover, the choice of restriction enzyme and the size of the DNA fragments used for the library avoided the sequencing of repeat loci (such as transposable elements), which are known to be common in the *I. scapularis* genome (Ullmann *et al.* 2005; Hill *et al.* 2009; Meyer *et al.* 2010) and unsuitable for the isolation of codominant and mendelian molecular markers.

Because of (i) the absence of reference genome, (ii) the large size of the *I. ricinus* genome, (iii) its high density of SNPs, (iv) its large proportion of repeat elements and (v) the frequency of sequencing errors obtained with the 454 technology, the genome assembly required by most SNP calling software would have been especially difficult to conduct. We thus argue that our bioinformatics tool, because it avoids genome assembly, is especially useful and relevant for the isolation of SNPs from the *I. ricinus* genome. This assembly step is in fact precluded using DiscoSnp. The strategy adopted here revealed a large number of SNPs (321 088) in our data set. Using GASSST, we conserved only 18.5% (1768) of the 9537 SNPs meeting all the requested quality criteria (quality of read, sequencing depth, absence of homopolymers). Because of our RRL strategy (and the selection of only a fraction of the whole-genome in our data set, with varying sequencing depths for each locus), the SNP density cannot be directly inferred at the whole-genome scale. Moreover, the final number of SNPs isolated in this pipeline is highly dependent on the stringency of the filters used to avoid SNPs that might correspond to artefacts or be located in unsuitable loci (duplicated loci, occurrence of other SNPs or microsatellite loci in the vicinity of the SNP targeted…). In our case, as we wanted to select only a limited number of SNP loci (384) for population genetics studies, we used parameter values that were highly selective to keep only those SNP loci that were the most suitable for primer design and genotyping. However, numerous additional SNPs isolated could have been used to design new primers in the remaining set of polymorphic sites identified.

Among 1768 SNPs selected as candidates for primer design, 384 SNPs were tested by genotyping 464 individuals of *Ixodes ricinus* from an area (130 km^2) located in Brittany (North of France). These 384 SNPs were successfully amplified at 96%, as 368 displayed both amplification and polymorphism during genotyping. No amplification was observed for five SNPs, while 11 other SNP loci exhibited only one kind of homozygous individuals. The obtained validation rate is higher than in other studies where missing SNPs were reported to represent between 6 and 52% (Sanchez *et al.* 2009; Hyten *et al.* 2010; Fu & Peterson 2012). This reflects the effectiveness and stringency of the pipeline developed to select only those SNPs with the most suitable loci for primer design.

Among the six populations investigated, a deviation from Hardy–Weinberg equilibrium was observed in 6.77 to 11.97% of the 384 loci genotyped. This is in agreement with previous investigations based on microsatellite loci that have all found high heterozygote deficiency (De Meeûs *et al.* 2004; Røed *et al.* 2006; Noel *et al.* 2012). The limited active dispersal of those arthropods, the existence of host-specialized races as well as associative mating may be responsible for this pattern, through a Wahlund effect (Kempf *et al.* 2009, 2011). For a better understanding of *I. ricinus* population structure, complementary investigations conducted at a limited spatial scale and using ticks for which the host used is known will be needed (Quillery *et al.* in prep).

The strategy reported here, which combines high throughput sequencing (HTS), genome reduction with the RRL technique and a powerful bioinformatics pipeline to isolate SNPs without requiring any reference genome, illustrates the feasibility of SNP discovery for nonmodel organisms like *Ixodes ricinus*. The SNP loci described here will be useful for a variety of applications such as the assessment of the genetic structure of *I. ricinus* populations or the building of a genetic map.

Acknowledgements

The authors would like to thank Héléne Verheyden and the CEFS laboratory (Comportement et Ecologie de la Faune Sauvage [CEFS], INRA, UR 035, Castanet-Tolosan, France), for providing engorged ticks collected on roe deer from Gardouch, and all the colleagues who took part in tick sampling at Malville and in the Zone Atelier Armorique. The coordinators of the Zone Atelier Armorique (Jacques Baudry and Cendrine Mony; http://osur.univ-rennes1.fr/zoneatelier-armorique/) are thanked for providing access to the site. Philippe Vanderkoornhuyse, Alexandra Dheilly and Sophie Coudouel from the 'Biogenouest platform' (Rennes) are acknowledged for the 454 pyrosequencing run. Charles Poncet and the 'GENTYANE platform' is acknowledged for Fluidigm genotyping. Members of the 'Ticks and Tick-Borne Diseases' group (Réseau Ecologie des interactions durables) are thanked for useful discussions as well as François Beaudeau (UMR BioEpAR) for his comments on an earlier version of the manuscript. This work was funded by INRA (Animal Health Division; Call of Proposal Epidemiology and Biology/CADIX; AIP Bioressources:

IXOMIC), the Pays de La Loire Region (PhD grant for E.Q.), the French National Research Agency (ANR-11-Agro-001-04; Call for Proposal 'Agrobiosphere', OSCAR project) and the European Commission (FP7 Health: EDENext projet- EDENext publication number 154).

References

Altshuler D, Pollara VJ, Cowles CR *et al.* (2000) An SNP map of the human genome generated by reduced representation shotgun sequencing. *Nature*, **407**, 513–516.

Chevreux B, Wetter T, Suhai S (1999) Genome Sequence Assembly Using Trace Signals and Additional Sequence Information. pp. 45–56.

De Meeûs T, Humair P-F, Grunau C, Delaye C, Renaud F (2004) Non-Mendelian transmission of alleles at microsatellite loci: an example in *Ixodes ricinus*, the vector of Lyme disease. *International Journal for Parasitology*, **34**, 943–950.

Delaye C, Aeschlimann A, Renaud F, Rosenthal B, De Meeûs T (1998) Isolation and characterization of microsatellite markers in the *Ixodes ricinus* complex (Acari: Ixodidae). *Molecular Ecology*, **7**, 360–361.

Ekblom R, Galindo J (2011) Applications of next generation sequencing in molecular ecology of non-model organisms. *Heredity*, **107**, 1–15.

Estoup A, Jarne P, Cornuet J-M (2002) Homoplasy and mutation model at microsatellite loci and their consequences for population genetics analysis. *Molecular Ecology*, **11**, 1591–1604.

Fu Y-B, Peterson GW (2012) Developing genomic resources in two Linum species via 454 pyrosequencing and genomic reduction. *Molecular Ecology Resources*, **12**, 492–500.

Gillet L, Schroeder H, Mast J *et al.* (2009) Anchoring tick salivary anticomplement proteins IRAC I and IRAC II to membrane increases their immunogenicity. *Veterinary Research*, **40**, 51.

Gray JS (2002) Biology of Ixodes species ticks in relation to tick-borne zoonoses. *Wiener Klinische Wochenschrift*, **114**, 473–478.

Gubler DJ (1998) Resurgent vector-borne diseases as a global health problem. *Emerging Infectious Diseases*, **4**, 442–450.

Helyar SJ, Hemmer-Hansen J, Bekkevold D *et al.* (2011) Application of SNPs for population genetics of nonmodel organisms: new opportunities and challenges. *Molecular Ecology Resources*, **11**, 123–136.

Hill CA, Guerrero FD, Zee JPV *et al.* (2009) The position of repetitive DNA sequence in the southern cattle tick genome permits chromosome identification. *Chromosome Research*, **17**, 77–89.

Hyten DL, Song Q, Fickus EW *et al.* (2010) High-throughput SNP discovery and assay development in common bean. *BMC Genomics*, **11**, 475.

Kempf F, De Meeus T, Arnathau C, Degeilh B, McCoy KD (2009) Assortative Pairing in *Ixodes ricinus* (Acari: Ixodidae), the European Vector of Lyme Borreliosis. *Journal of Medical Entomology*, **46**, 471–474.

Kempf F, De Meeus T, Vaumourin E *et al.* (2011) Host races in *Ixodes ricinus*, the European vector of Lyme borreliosis. *Infection, Genetics and Evolution*, **11**, 2043–2048.

Lao O, van Duijn K, Kersbergen P, de Knijff P, Kayser M (2006) Proportioning Whole-Genome Single-Nucleotide–Polymorphism Diversity for the Identification of Geographic Population Structure and Genetic Ancestry. *The American Journal of Human Genetics*, **78**, 680–690.

Li H, Handsaker B, Wysoker A *et al.* (2009) The Sequence Alignment/Map format and SAMtools. *Bioinformatics (Oxford, England)*, **25**, 2078–2079.

Marshall OJ (2004) PerlPrimer: cross-platform, graphical primer design for standard, bisulphite and real-time PCR. *Bioinformatics*, **20**, 2471–2472.

McKenna A, Hanna M, Banks E *et al.* (2010) The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Research*, **20**, 1297–1303.

Meyer JM, Kurtti TJ, Zee JPV, Hill CA (2010) Genome organization of major tandem repeats in the hard tick, *Ixodes scapularis*. *Chromosome Research*, **18**, 357–370.

Milne I, Stephen G, Bayer M *et al.* (2013) Using Tablet for visual exploration of second-generation sequencing data. *Briefings in Bioinformatics*, **14**, 193–202.

Morin PA, Luikart G, Wayne RK, the SNP workshop group, (2004) SNPs in ecology, evolution and conservation. *Trends in Ecology & Evolution*, **19**, 208–216.

Nielsen R, Paul JS, Albrechtsen A, Song YS (2011) Genotype and SNP calling from next-generation sequencing data. *Nature Reviews Genetics*, **12**, 443–451.

Noel V, Léger E, Gómez-Díaz E, Risterucci A-M, McCoy KD (2012) Isolation and characterization of new polymorphic microsatellite markers for the tick *Ixodes ricinus* (Acari, Ixodidae). *Acarologia*, **52**, 123–128.

Noureddine R, Chauvin A, Plantard O (2011) Lack of genetic structure among Eurasian populations of the tick *Ixodes ricinus* contrasts with marked divergence from north-African populations. *International Journal for Parasitology*, **41**, 183–192.

Parola P, Raoult D (2001) Tick-borne bacterial diseases emerging in Europe. *Clinical microbiology and infection: the official publication of the European Society of Clinical Microbiology and Infectious Diseases*, **7**, 80–83.

Paschou P, Ziv E, Burchard EG *et al.* (2007) PCA-Correlated SNPs for Structure Identification in Worldwide Human Populations. *PLoS Genetics*, **3**, e160.

Philipp MT, Lobet Y, Bohm RP Jr *et al.* (1997) The outer surface protein A (OspA) vaccine against Lyme disease: efficacy in the rhesus monkey. *Vaccine*, **15**, 1872–1887.

Porretta D, Mastrantonio V, Mona S *et al.* (2013) The integration of multiple independent data reveals an unusual response to Pleistocene climatic changes in the hard tick *Ixodes ricinus*. *Molecular Ecology*, **22**, 1666–1682.

Rizk G, Lavenier D (2010) GASSST: global alignment short sequence search tool. *Bioinformatics*, **26**, 2534–2540.

Røed KH, Hasle G, Midthjell V, Skretting G, Leinaas HP (2006) Identification and characterization of 17 microsatellite primers for the tick, *Ixodes ricinus*, using enriched genomic libraries. *Molecular Ecology Notes*, **6**, 1165–1167.

Rousset F (2008) genepop'007: a complete re-implementation of the genepop software for Windows and Linux. *Molecular Ecology Resources*, **8**, 103–106.

Sanchez CC, Smith TP, Wiedmann RT *et al.* (2009) Single nucleotide polymorphism discovery in rainbow trout by deep sequencing of a reduced representation library. *BMC Genomics*, **10**, 559.

Schlötterer C (2004) The evolution of molecular markers — just a matter of fashion? *Nature Reviews Genetics*, **5**, 63–69.

Schulze TL, Jordan RA, Hung RW (1997) Biases associated with several sampling methods used to estimate abundance of *Ixodes scapularis* and Amblyomma americanum (Acari: Ixodidae). *Journal of Medical Entomology*, **34**, 615–623.

Smouse PE (2010) How many SNPs are enough? *Molecular Ecology*, **19**, 1265–1266.

Ullmann AJ, Lima CMR, Guerrero FD, Piesman J, Black WC IV (2005) Genome size and organization in the blacklegged tick, *Ixodes scapularis* and the Southern cattle tick *Boophilus microplus*. *Insect Molecular Biology*, **14**, 217–222.

Van Tassell CP, Smith TPL, Matukumalli LK *et al.* (2008) SNP discovery and allele frequency estimation by deep sequencing of reduced representation libraries. *Nature Methods*, **5**, 247–252.

Van Zee J, Black WC IV, Levin M *et al.* (2013) High SNP density in the blacklegged tick, *Ixodes scapularis*, the principal vector of Lyme disease spirochetes. *Ticks and Tick-borne Diseases*, **4**, 63–71.

Wang J, Lin M, Crenshaw A *et al.* (2009) High-throughput single nucleotide polymorphism genotyping using nanofluidic Dynamic Arrays. *BMC Genomics*, **10**, 561.

E.Q. conceived and designed the project and performed experiments, collected samples, analysed data and wrote the manuscript. O.Q. conceived and performed the bioinformatics analysis and helped correct the text. P.P. conceived and developed the bioinformatics tool and helped correct the text. O.P. conceived and designed the project, collected samples and wrote the manuscript.

Data Accessibility

For SNPs sequences, please see the online supplementary materials. For DNA sequences, see NCBI SRA: SRX327489 For draft DNA sequence assembly (.ace), output files from DiscoSnp (.fasta) and Bioinformatics scripts see DRYAD entry doi:10.5061/dryad.1h1f2.

Supporting Information

Additional Supporting Information may be found in the online version of this article:

Figure S1 Workflow illustrating the application of 454 pyrosequencing in the *de novo* identification, selection and validation of SNPs.

Figure S2 Distribution of the reads (length in bp) for the two PicoTiterPlate regions in the Roche 454 GS FLX run.

Figure S3 Ratio of unique k-mers obtained for different k-mer size (nt).

Figure S4 Number of contigs identified by MIRA3 with a sequencing depth only between 2 and 20.

Table S1 Excel file containing primers sequences for the 384 SNP loci.

Table S2 Excel file containing the 384 sequences containing SNP and with MAF for each locus.

Table S3 Excel file containing Observed (H_o) and expected (H_e) heterozygosity, p-value and standard error of Hardy–Weinberg Equilibrium test in the 6 populations with the largest population size (L041 to L064).

Mapping-free and assembly-free discovery of inversion breakpoints from raw NGS reads

Mapping-free and assembly-free discovery of inversion breakpoints from raw NGS reads

Claire Lemaitre, Liviu Ciortuz, Pierre Peterlongo

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] GenScale, INRIA Rennes Bretagne-Atlantique, IRISA
[2] Faculty of Computer Science Iasi, Romania

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) [1]. 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 [2]. 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 [3]. Noteworthy, copy number variations seem to have focused most attention and efforts, whereas balanced structural variants such as inversions have been less investigated [1], 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,

one can also perform full *de novo* assembly of re-sequenced genomes and compare the resulting assemblies [4], 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, such as the *de Bruijn Graph*. Some of them are dedicated to detect SNPs or small indels [5,6,7,8] or alternative slicing events in RNA-seq data [9], but none for 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 enabled to show that i) the described model detects with high recall and precision inversion breakpoints ii) even if they are problematic, numerous 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 [10], 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 $n_\omega^\rightarrow$ denotes the node storing explicitly or implicitly ω, in the state such that reading $n_\omega^\rightarrow$ provides the k-mer ω. Respectively, $n_\omega^\leftarrow$ denotes the same node in the state such that reading $n_\omega^\leftarrow$ provides the k-mer $\overleftarrow{\omega}$.

Given two nodes $n_{start}^\rightarrow$ and $n_{stop}^\rightarrow$, we say that a path of length l exists from node $n_{start}^\rightarrow$ to node $n_{stop}^\rightarrow$, *iif* node $n_{stop}^\rightarrow$ can be reached using l nodes from node

$n^{\rightarrow}_{start}$ and for any traversed node, it should be entered and leaved in the same state (i.e. explicit or implicit). Let k-path denote a path of length k.

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.

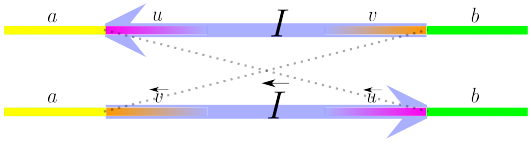


Fig. 1: Sequences aIb and $a\overleftarrow{I}b$, showing the four particular k-mers a, u, v and b at the breakpoints of the inversion.

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 $n^{\rightarrow}_{a}$ that branches to two distinct k-paths that end repsectively in $n^{\rightarrow}_{u}$ and $n^{\leftarrow}_{v}$. Similarly, the other two breakpoints (at the right) are characterized by two k-paths starting from $n^{\leftarrow}_{u}$ and $n^{\rightarrow}_{v}$ that join in $n^{\rightarrow}_{b}$. 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 $n^{\rightarrow}_{a}$ 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 $n^{\rightarrow}_{a}$ to $n^{\rightarrow}_{u}$; one from $n^{\leftarrow}_{u}$ to $n^{\rightarrow}_{b}$, one from $n^{\leftarrow}_{b}$ to $n^{\leftarrow}_{v}$; one from $n^{\rightarrow}_{v}$ to $n^{\leftarrow}_{a}$. 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 $n^{\leftarrow}_{b}$, or $n^{\leftarrow}_{u}$ or

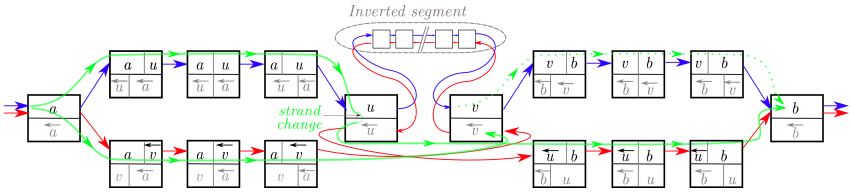


Fig. 2: Schematic example of the inversion pattern generated by sequences aIb (the blue path) and $a\overleftarrow{I}\,b$ (the red path) in a *de Bruijn Graph* with $k = 4$. Nodes are represented as two-stage boxes, with the upper part in black showing the explicit k-mer and the lower part, in grey, 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 shake 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.

$n_v^{\rightarrow}$ 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).

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.

Algorithm 1 Main algorithm to detect the inversion pattern.

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

3.2 Canonical representation of occurences

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 occurences 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 depending both on the starting node $n_a^{\rightarrow}$ and the order of detection between $n_u^{\rightarrow}$ and $n_v^{\leftarrow}$. To avoid outputing 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 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).

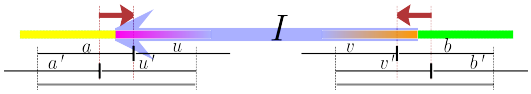


Fig. 3: Example of an inversion with small inverted repeats (red arrows) at the breakpoints. Breakpoint sequences au, vb (resp. $a'u', v'b'$) are obtained starting from nodes $n_a^{\rightarrow}$ or $n_b^{\leftarrow}$ (resp. $n_{u'}^{\leftarrow}$ or $n_v^{\rightarrow}$). The unique canonical representative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

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

sub-parts 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 [11,12]. 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 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).

	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

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.

When varying the main parameter k, results remained almost unchanged: only for $k < 21$ recall and precision slightly decrease to reach 98.6% and 90% respectively. Indeed the smaller k, the more there are genomic repetitions merged in single nodes of the graph, some of the repeats can then form false positive patterns. Also, a side effect is an increased processing time since there are more branching nodes to explore. Conversely when increasing k, precision should be favored, recall can be affected but only if the value of k is unrealistic with respect to the sequencing coverage, leading to uncovered parts of the genome.

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.

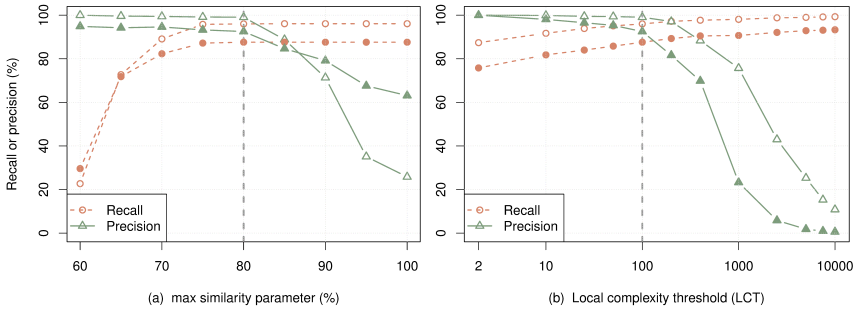


Fig. 4: Effect of the filtering parameters, max_sim (a) and LCT (b), on precision and recall values for the *C. elegans* (open symbols) and human chromosome 22 (solid symbols) datasets. Vertical dashed lines represent the default parameters.

4.3 Time/memory performances

These tests were performed with 2.3 GHz Intel Core i7 processors, with 8GB RAM memory.

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).

	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

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).

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 [6,9], this strongly limits the size of detectable events (at least in complex genomes) and increases the computational time.

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, NHEJ 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 [8]. 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 [13], 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.

6 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 the French ANR-12-BS02-0008 *Colib'read* project.

References

1. Mills, R.E., Walter, K., Stewart, C., Handsaker, R.E., 1000 Genomes Project: Mapping copy number variation by population-scale genome sequencing. Nature **470** (2011) 59–65
2. Medvedev, P., Stanciu, M., Brudno, M.: Computational methods for discovering structural variation with next-generation sequencing. Nat Methods **6** (2009) S13–S20
3. Alkan, C., Coe, B.P., Eichler, E.E.: Genome structural variation discovery and genotyping. Nat Rev Genet **12** (2011) 363–376
4. 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** (2011) 723–730
5. Peterlongo, P., Schnel, N., Pisanti, N., Sagot, M.F., Lacroix, V.: Identifying snps without a reference genome by comparing raw reads. In: SPIRE. Volume 6393 of Lecture Notes in Computer Science. (2010) 147–158
6. 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** (2012) 226–232
7. 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** (2013) 325–330
8. Uricaru, R., et al.: discoSnp Software. `http://colibread.inria.fr/discosnp/` (Manuscript in prep. 2014)
9. 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** (2012) S5
10. Zerbino, D.R., Birney, E.: Velvet: algorithms for de novo short read assembly using de bruijn graphs. Genome research **18** (2008) 821–829
11. Chikhi, R., Rizk, G.: Space-efficient and exact de bruijn graph representation based on a bloom filter. Algorithms for Molecular Biology **8** (2013) 22
12. Salikhov, K., Sacomoto, G., Kucherov, G.: Using cascading Bloom filters to improve the memory usage for de Brujin graphs. In: WABI 2013. Volume 8126 of Lecture Notes in Computer Science., Springer (2013)
13. Lemaitre, C., et al.: MindTheGap Software. `http://mindthegap.genouest.org/` (Manuscript in prep. 2014)

Publications associées au chapitre 3

Compareads : comparing huge metagenomic experiments

Maillet et al. BMC Bioinformatics 2012, 13(Suppl 19):S10
http://www.biomedcentral.com/1471-2105/13/S19/S10

BMC Bioinformatics

PROCEEDINGS Open Access

Compareads: comparing huge metagenomic experiments

Nicolas Maillet[1*], Claire Lemaitre[1], Rayan Chikhi[2], Dominique Lavenier[1], Pierre Peterlongo[1*]

From Tenth Annual Research in Computational Molecular Biology (RECOMB) Satellite Workshop on Comparative Genomics
Niteroi, Brazil. 17-19 October 2012

Abstract

Background: Nowadays, metagenomic sample analyses are mainly achieved by comparing them with *a priori* knowledge stored in data banks. While powerful, such approaches do not allow to exploit unknown and/or "unculturable" species, for instance estimated at 99% for Bacteria.

Methods: This work introduces Compareads, a *de novo* comparative metagenomic approach that returns the reads that are similar between two possibly metagenomic datasets generated by High Throughput Sequencers. One originality of this work consists in its ability to deal with huge datasets. The second main contribution presented in this paper is the design of a probabilistic data structure based on Bloom filters enabling to index millions of reads with a limited memory footprint and a controlled error rate.

Results: We show that Compareads enables to retrieve biological information while being able to scale to huge datasets. Its time and memory features make Compareads usable on read sets each composed of more than 100 million Illumina reads in a few hours and consuming 4 GB of memory, and thus usable on today's personal computers.

Conclusion: Using a new data structure, Compareads is a practical solution for comparing *de novo* huge metagenomic samples. Compareads is released under the CeCILL license and can be freely downloaded from http://alcovna.genouest.org/compareads/.

Introduction

The past five years have seen the arrival of High Throughput Sequencing (HTS), also known as Next-Generation Sequencing (NGS). These technologies drastically lowered sequencing costs and increased sequencing throughput. They radically changed molecular biology and computational biology, as data generation is no longer a bottleneck. In fact, nowadays a major challenge is the analysis and interpretation of sequencing data [1]. HTS democratized access to sequencing to almost all biological labs over the world. It also opened the doors to new techniques such as ChipSeq [2], ClipSeq [3], RadSeq [4] and the topic of this work, metagenomics [5].

Metagenomics, also known as "environmental genomics", provides an alternative to traditional single- genome studies for exploring the microbial world. Most microorganisms (up to 99% of Bacteria [6]) are unknown and possibly "unculturable". Even if traditional genomics sequencing methods are well studied, they are not suited for environmental samples, because of the need to cultivate clones. By sequencing uncultured genomes directly from environmental samples, metagenomics offers new ways to study this unexplored diversity.

HTS technologies provide fragments of sequences (called reads) of length a few hundred base pairs without any information about the locus nor the orientation on the molecule they come from. In the metagenomic context, an additional difficulty comes from the fact that each read may belong to any species.

* Correspondence: nicolas.maillet@inria.fr; pierre.peterlongo@inria.fr
[1]INRIA Rennes - Bretagne Atlantique/IRISA, EPI GenScale, Rennes, France
Full list of author information is available at the end of the article

Nowadays, it is difficult to assemble complex metagenomes (such as soil or water metagenomes) into longer consensus sequences, because reads from different species may be merged into one chimeric sequence. Mende and colleagues [7] showed that for a 400-genomes metagenome, using simulated Illumina reads, 37% of the assembled sequences were chimeric. Thus currently, reads from metagenomes are used to estimate the biodiversity [8] or may be compared to known databases, providing information with respect to the current scientific knowledge [9,10]. Another way to exploit two or more metagenomic datasets is to compare them together, enabling to understand how genomic differences are related to environmental ones (biotopes localizations and/or time spent after an event).

Comparative metagenomics usually deals with many aspects, such as sequence composition, *i.e.* GC content [11], and genome size [12], taxonomic diversity [13], functional content [14], etc. Several methods are currently developed for comparative metagenomics analyses. Some are based on statistical methods with a large number of descriptive variables, *e.g.* principal component analysis (PCA).

To the best of our knowledge, there is no software designed to compare two or more metagenomic samples at the read level, *i.e.* to identify reads that are shared or similar between samples. This can be simply used to compute a similarity measure between samples such as the number or percentage of similar reads between pairs of samples. When dealing with more than two samples, this would enable among others to classify metagenomics samples based on their raw reads content. One could use the popular tool BLAST to align reads in an all-vs-all way, however it is not designed specifically to this task, and more importantly, it cannot cope in time and memory with the size of nowadays metagenomic samples obtained with current sequencing technologies. For instance, with the aim of exploring the diversity of small eukaryotes in the oceans all over the world, the expedition "Tara Ocean" [15] is generating more than 400 metagenomic samples containing each around 100 million short reads, that will need to be compared to each other.

Here, we introduce a time and memory-efficient method for extracting similar reads between two metagenomic datasets. The similarity is based on shared k-mers (words of length k). In order to fit with current memory capacities, the data structure we use is a modified version of a Bloom filter [16]. Bloom filters have recently been used in bioinformatics, notably for assembly graph partitioning [17], which enabled to perform metagenomic *de novo* assembly using 30x less memory.

This manuscript presents two main contributions: (**I**) a new algorithm, called Compareads, which computes the similarity measure between two metagenomics datasets; (**II**) a new simple but extremely efficient data structure based on the Bloom filter for storing the presence/absence of k-mers in huge datasets. The manuscript is organized as follows: in Section "*Methods*", we depict the Compareads algorithm and the new data structure. In Section "*Results*" we provide results both about the data structure and about Compareads, showing the efficiency of our approach in term of computation time, memory and biological accuracy.

Methods

Preliminaries and definitions A *sequence* is composed by zero or more symbols from an alphabet Σ. In this work, as we are dealing with DNA, $\Sigma = \{A, C, G, T\}$. A sequence s of length n on Σ is denoted also by $s[0]s[1] \ldots s[n-1]$, where $s[i] \in \Sigma$ for $0 \leq i < n$. We denote by $s[i, j]$ the *substring* $s[i]s[i+1] \ldots s[j]$ of s. In this case, we say that the substring $s[i, j]$ occurs at position i in s. We call *k-mer* a sequence of length k, and $s[i, i+k-1]$ is a k-mer occurring at position i in s.

Overview of Compareads Compareads is designed for finding similar sequences between two read sets. This basic operation may appear extremely simple. However, it has to be highly efficient, in term of computation time and memory footprint, in order to scale with huge metagenomics datasets.

In order to perform efficiently this operation, Compareads indexes k-mers and uses a rough but efficient notion of "*similar sequences*" defined as follows:

Definition 1 (shared *k*-mer) *Two sequences s_1 and s_2 share a k-mer if and only if $\exists(i_1, i_2)$ such that $s_1[i_1, i_1 + k - 1] = s_2[i_2, i_2 + k - 1]$.*

Definition 2 (Similar sequences) *Given integers k and t, two sequences s_1 and s_2 are said similar if and only if they share at least t non overlapping k-mers.*

In a few words, given two read sets A and B, the goal of the Compareads algorithm is to find the subset of reads from A which are similar to a read in B such set being denoted by $(A \vec{\cap} B)$ As it is a heuristic (see Section "*Dealing with false positives*"), our algorithm outputs an over-approximation of set $(A \vec{\cap} B)$ denoted by $(A \vec{\vec{\cap}} B)$.

Computing $(A \vec{\vec{\cap}} B)$

Compareads computes $(A \vec{\vec{\cap}} B)$ in two steps. **The indexing** step consists in storing in memory all k-mers having at least one occurrence in the set B. The **query** step processes reads from set $\boldsymbol{A}$ one by one. For a read $r \in A$, the index is used to test the presence in the set B of each k-mer of r. If at least t non-overlapping k-mers are returned as present, then the read r is inserted in $(A \vec{\vec{\cap}} B)$. The main practical challenge faced by Compareads is to index the possibly huge volume of k-mers

contained in B. The data structure must therefore fulfill three criteria: it must be quick to build, have a low memory footprint and be quick to request. Section "*The Bloom Data Structure index*" describes the chosen probabilistic data structure, based on a Bloom filter.

Limiting the indexing space To control the approximation error (see Section "*Dealing with false positives*"), the indexing phase is interrupted whenever the volume of k-mers in the first reads of B exceeds a fixed value n. The query phase is then performed on the whole A dataset. This phase returns a partial intersection between A and a first chunk of reads from B. The remaining partial intersections between A and the next chunks of reads from B (each representing a volume of n k-mers or less) are sequentially computed, until all the reads from B have been indexed. Eventually, Compareads returns the union of all partial intersections. Note that, in terms of results, this partitioning approach is strictly equivalent to performing a complete indexing of B then a query of all the reads from A. To avoid redundant computations, reads from A considered as "similar" in one of the partial intersections are tagged using a bitvector and are not queried further.

Time complexity Let n_A and n_B be the number of k-mers respectively in set A and set B. Computing $(A \tilde{\cap} B)$ is done in time $O(n_B)$ (indexing) + $O\left(n_A \times \frac{n_B}{n}\right)$ (query). The $\frac{n_B}{n}$ term is due to the limitation of the indexing space.

Ad hoc data structure

The index data structure we use is based on a Bloom filter, specially designed for the task of storing efficiently a huge set of k-mers, while being fast to build and to query. We shortly recall in Section "*Bloom filter*" what a Bloom filter is before describing, in Section "*The Bloom Data Structure index*", our data structure called BDS.

Bloom filter

A Bloom filter is a probabilistic data structure designed to test the membership of elements in a set [16]. It consists of an array of m bits, all initialized to zero, and a set of hash functions. Each hash function maps an element to a single position in the array. Each element is associated, through the values of the hash functions, to several positions in the array. To insert an element in the structure, the bits in the array associated to this element are all set to one. The structure answers membership queries by checking whether all the bits in the array associated to an element are set to one.

This data structure is probabilistic in nature, as false positives are possible. Even if an element is not in the set, its bits in the array may still be all set to one. This is because the bits associated to an element may independently be associated to other elements. Hence, the Bloom filter returns a wrong answer with non-zero probability. This probability is the *false positive rate*. An asymptotic approximation of the false positive rate is $0.6185^{m/n}$, assuming n elements are inserted in the m-bits array, and $(\ln 2 \cdot (m/n))$ hash functions are used [18]. False negatives never occur: if an element belongs to the set, the Bloom filter always answers positively. Bloom filters are space-efficient: only $(n \log_2 e \cdot \log_2(1/\varepsilon))$ bits are required to support membership queries for n elements with a false positive rate of ε [18].

The Bloom Data Structure index

In this article, we consider a slightly different variation of Bloom filters: instead of using a single array of bits, each hash function corresponds to a distinct array, disjoint from all other functions. In terms of performance, with uniform hash functions, this variation is asymptotically equivalent to the original definition [18]. To avoid confusion with classical Bloom filters, we refer to this variation as BDS, standing for Bloom Data Structure.

Particular hash functions The hash functions used in this framework are a specific family of functions, which can be efficiently computed on consecutive k-mers. We consider the set of functions which map a k-mer to a bit sequence of length k, where each nucleotide is associated to a bit set to 0 or 1, depending only on its type (A, C, G or T). An exhaustive enumeration, in equations 1 and 2, shows that there exists only 7 functions in this set. We can distinguished two types, the first three, f_1, f_2 and f_3, are said to be *balanced* (equation 1), whereas the other four are said to be *unbalanced* (equation 2)

$$f_j : \Sigma^k \to \{0,1\}^k : \forall i \in [1,\ k] \begin{cases} f_1(s)[i] = 0 & if \quad s[i] = A \text{ or } C \quad f_1(s)[i] = 1 \quad \text{otherwise} \\ f_2(s)[i] = 0 & if \quad s[i] = A \text{ or } G \quad f_2(s)[i] = 1 \quad \text{otherwise} \\ f_3(s)[i] = 0 & if \quad s[i] = A \text{ or } T \quad f_3(s)[i] = 1 \quad \text{otherwise} \end{cases} \tag{1}$$

$$f_j : \Sigma^k \to \{0,1\}^k : \forall i \in [1,\ k] \begin{cases} f_4(s)[i] = 0 & \text{if} \quad s[i] = A \quad f_4(s)[i] = 1 \quad \text{otherwise} \\ f_5(s)[i] = 0 & \text{if} \quad s[i] = C \quad f_5(s)[i] = 1 \quad \text{otherwise} \\ f_6(s)[i] = 0 & \text{if} \quad s[i] = G \quad f_6(s)[i] = 1 \quad \text{otherwise} \\ f_7(s)[i] = 0 & \text{if} \quad s[i] = T \quad f_7(s)[i] = 1 \quad \text{otherwise} \end{cases} \tag{2}$$

One important property of these functions is that there is a simple relationship between the hash values of two consecutive k-mers in a read. One can see that the hash value of the next k-mer can be quickly computed, by left-shifting the binary sequence of the previous hash value and appending an extra bit. These functions are not classical hash functions, yet we show that they exhibit good hashing properties when applied to k-mers. In Section "*Practical performance of the* BDS, *comparison with other data structures*", the performance of these functions is compared with that of a classical hash function in terms of computation time, and false positive rate in the BDS.

The Compareads pipeline

Computing $(A \tilde{\cap} B)$ is asymmetrical. Indeed $(A \tilde{\cap} B)$ does not contain the reads from B which are similar to reads in A. For doing this, one needs to compute also $(B \tilde{\cap} A)$. In practice, for fully and symmetrically comparing two

sets A and B we apply a pipeline slightly more complicated than simply $(A \widetilde{\cap} B)$ followed by $(B \widetilde{\cap} A)$. This whole pipeline, designed for reducing a heuristic effect is described in Section "*False positives due to k-mer shared between a read and a dataset*".

Similarity measure While comparing read sets A and B, the result provided by Compareads is composed of two sets: $(A \widetilde{\cap} B)$ and $(B \widetilde{\cap} A)$. Then, a similarity measure between the two datasets is computed as follows:

$Sim(A,B) = \frac{\left|A \widetilde{\cap} B\right| + \left|B \widetilde{\cap} A\right|}{|A|+|B|} * 100$ where $|X|$ denotes the cardinality of the set X.

Dealing with false positives

Our approach may generate false positives for two reasons we describe in the two upcoming sections, which also expose solutions for limiting these effects.

False positives due to k-mer shared between a read and a dataset

Using t >1, Compareads algorithm can call similar sequences that do not respect strictly the definition of similarity given in definition 2. Indeed, steps described in Section "*Computing* $(A \widetilde{\cap} B)$" detect reads from A that share at least t k-mers with reads from B. This is less stringent than finding reads from A that share at least t k-mers with at least one read from set B. In fact, the t k-mers found in read A are possibly spread over two or more distinct reads from set B.

This issue can be mitigated by performing the following steps to compute both $(A \widetilde{\cap} B)$ and $(B \widetilde{\cap} A)$:

1. Compute $(A \widetilde{\cap} B)$, storing the results in a set denoted by $(A \widetilde{\cap} B)^*$.
2. Compute $\left(B \widetilde{\cap} (A \widetilde{\cap} B)^*\right)$ storing the results in a set denoted by $(B \widetilde{\cap} A)$.
3. Compute $\left(A \widetilde{\cap} (B \widetilde{\cap} A)\right)$ storing the results in a set denoted by $(A \widetilde{\cap} B)$.

In a few words, the two output datasets $(B \widetilde{\cap} A)$ and $(A \widetilde{\cap} B)$ are obtained by applying the fundamental operation $(\widetilde{\cap})$ between a query and a read set being itself already the result of the asymmetrical $(\widetilde{\cap})$ operation. This enables to remove some false positives due to k-mers spread over several reads.

The example presented in Figure 1 illustrates this issue for the case t = 2. The two first reads of sets A and B are similar. They are classically output by Compareads respectively in $(A \widetilde{\cap} B)$ and $(B \widetilde{\cap} A)$. The two next reads contain only one shared k-mer (yellow) with reads of set B, they are discarded. The next read of set A contains two (red) shared k-mers with two distinct reads in set B. After a first comparison, $(A \widetilde{\cap} B)^*$ contains this false positive read. However, in step 2, while computing $(B \widetilde{\cap} A)$, these two reads are not conserved in $(B \widetilde{\cap} A)$. Thus, during step 3, the two red k-mers are not present anymore in set $(B \widetilde{\cap} A)$ and thus are not present in $\left(A \widetilde{\cap} (B \widetilde{\cap} A)\right)$. They are thus correctly absent from the final results $(A \widetilde{\cap} B)$. However, the last read from set A is a case of false positive. It contains k-mers spread over distinct reads from B, the latter belonging to $(B \widetilde{\cap} A)$. Thus, even during step 3, these two k-mers remain shared with reads from set $(B \widetilde{\cap} A)$ and are output in $(A \widetilde{\cap} B)$.

Note that in practice, the last set $(A \widetilde{\cap} B)$ is obtained by computing $\left((A \widetilde{\cap} B)^* \widetilde{\cap} (B \widetilde{\cap} A)\right)$ instead of simply $\left(A \widetilde{\cap} (B \widetilde{\cap} A)\right)$ (used here for simplifying the reading). This operation provides the same result but is computed faster as $|(A \widetilde{\cap} B)^*| \leq |A|$.

As outlined in the example Figure 1, this pipeline still yields some false positives. These are characterized by t shared k-mers with at least two distinct reads from the indexed dataset B, themselves considered as similar to reads of set A. Even if this side effect is difficult to assess, we show in Section "*Comparison with a classical approach using* BLAST" that Compareads provides trustworthy results, highly similar to a classical approach, on several real datasets.

Bloom filter false positives

As exposed in Section "*The Bloom Data Structure index*", the BDS index is a probabilistic data structure, that may consider a k-mer as indexed while this is not the case (*i.e.* a false positive or FP). Here, we analyse the variations of the false positive rate for each hash function and their combinations with respect to the parameter k and the number n of distinct indexed k-mers. This enables to determine optimal parameters and appropriate combination of functions, that give the best trade-off between memory usage and false positive rate.

FP probablity for each function Assuming the nucleotide composition of the indexed k-mers and of the query k-mers are unbiaised, we can easily compute the probability, $P_{FP}(f_i, k, n)$, for any query k-mer to be a false positive with one of the seven hash functions, f_i (see Additional file 1 for details). The expression of this probablity is presented in equation 3 for a balanced hash function, and 4 for an unbalanced one.

$$\forall i \in \{1,2,3\} \quad P_{FP}(f_i,\ k,\ n) = 1 - (1 - \frac{1}{2^k})^n \tag{3}$$

$$\forall i \in \{4,5,6,7\} \quad P_{FP}(f_i,\ k,\ n) = \sum_{x=0}^{k} \binom{k}{x} a_x (1-(1-a_x)^n) \quad \text{with} \quad a_x = (\tfrac{1}{4})^x (\tfrac{3}{4})^{k-x} \tag{4}$$

We have plotted in Figure 2a the theoretical FP rate for both types of hash functions, and we can see that balanced functions give much less false positives than unbalanced ones. This is due to the fact that balanced

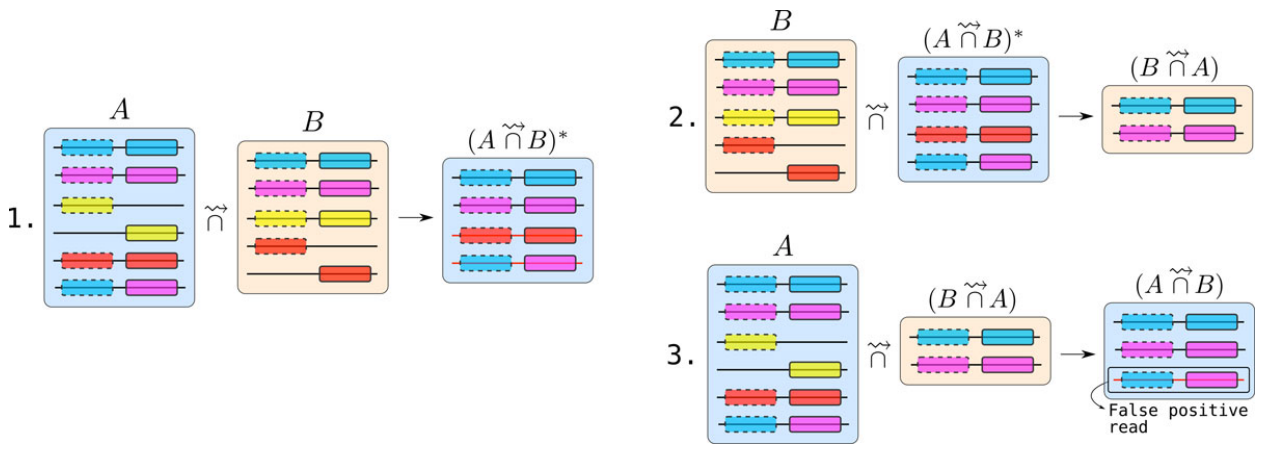


Figure 1 The Compareads pipeline. Representation of the three steps while comparing symmetrically read sets *A* and *B*. In each set, reads are represented by horizontal lines. On each read one or two shared *k*-mers are represented by rectangles.

functions distribute the hash codes uniformly over the 2^k bit-array, while this not the case for the unbalanced ones.

FP probablity for a combination of functions One important property of the balanced hash functions is that there do not exist two distinct *k*-mers that have the same couple of hash codes with any two of these functions. This implies that, in terms of false positives, two balanced functions have limited interferences with each other (see details in Additional file 1). The probability of FP can then be easily computed as follows:

$$P_{FP}(f_1 \cap f_2 \cap f_3,\ k,\ n) \underset{\sim}{<} (1 - (1 - \frac{1}{2^k})^n)^3 \qquad (5)$$

This "independence" property implies also that combining these 3 functions in our BDS is a very efficient strategy to reduce the FP rate, as can be seen in Figure 2a, especially for large values of *n*

Concerning the unbalanced functions, such property does not hold, since it is possible to find couples of distinct *k*-mers that share the same couple of hash codes for at least 2 of the unbalanced functions, or for one balanced function and at least one unbalanced. Therefore the theoretical FP rate of unbalanced functions is much more difficult to compute. We performed simulations to compute an empirical FP rate. We found that empirical results are very close to the formula obtained by multiplying the individual probabilities, *i.e.* assuming complete

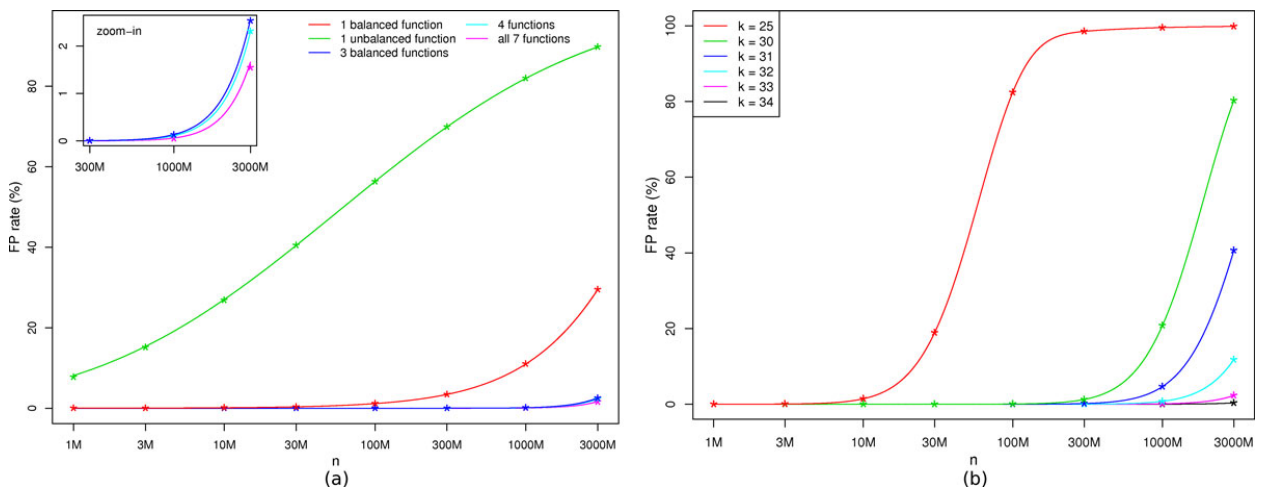


Figure 2 BDS false positive rate w.r.t. hash functions (a) and *k*-value (b). FP rate as a function of the number of indexed *k*-mers (in log scale). Plain lines correspond to theoretical predictions, whereas star points correspond to empirical values obtained with simulations. **(a)** This figure was obtained for $k = 33$ for balanced and unbalanced functions and some combinations of them. The combination entitled "4 functions" is composed of the 3 balanced functions plus one unbalanced. **(b)** For several values of *k*, this figure was obtained for a combination of 4 hash functions: all three balanced plus one unbalanced.

independence between all functions (Figure 2a). For details about how empirical results were obtained, see Additional file 2.

Choice of parameters The comparison of these FP curves led us to choose the combination of the three balanced functions plus an unbalanced one. This choice is motivated by the fact that unbalanced functions are not essential, as they have a limited effect on the FP rate (Figure 2a). Since hash functions described in Section "*Particular hash functions*" have a fixed range, the memory used by the BDS depends only on the value of k and the number of hash functions used. Recall that each hash function is associated to a dedicated bit array which occupies 2^k bits. Using 7 hash functions, the BDS has a total memory footprint of $7 \cdot 2^k$ bits and can be stored in 2^k bytes. When using only 4 hash functions, the BDS occupies $4 \cdot 2^k$ bits and can be stored in twice less space (2^{k-1} bytes).

For the chosen combination of functions, we plotted the FP rate as a function of n and for several values of k in Figure 2b. The larger is k, the less FP we get for a given number of indexed k-mers. Consequently, for large values of k, more k-mers can be indexed while maintaining a reasonable FP rate. However, the memory allocated to BDS grows with k and larger values of k increases the stringency of our similarity measure. We can see in Figure 2b, that using k-mers of size at least 30 enables to index at least 300 millions of k-mers with less than 2% of false positives.

For $k = 33$, when indexing up to one billion distinct k-mers, we obtain a theoretical upper bound of 0.13% of false positives (with 3 balanced functions, equation 5). The FP rate is even lower when adding one of the unbalanced function, we estimated it empirically to 0.114%. Thus, using 4 hash functions and $k = 33$ is a good set of parameters for indexing one billion distinct k-mers. With such parameters, the memory usage of Compareads is 4 GB.

Results

Practical performance of the BDS, comparison with other data structures

We propose here a comparative analysis of the BDS with other data structures. In the following, we show that classical non probabilistic data structures result in a worse time and memory performance, while in Section "*Comparison with other hash functions and with a classical Bloom filter*", we show that the BDS is the best suited for the problem of indexing huge amounts of k-mers.

Comparison with non probabilistic data structures: suffix array and hash table

Indexing n characters using the simplest version of a suffix array (not enhanced [19] and without LCP information) requires $5n$ bytes of memory [20]. Compared to our set of parameters where n = 1 billion, the memory footprint would be 5×10^9 bytes, *i.e.* 4.66 GB. While this is comparable to the BDS, the query time of the suffix array, $O(k \log n)$, is significantly worse.

An hash table can be used to store an exact set of k-mers. Such structure stores the k-mers explicitly, hence it requires at least $n \cdot \left\lceil \frac{2k}{8} \right\rceil \cdot 8$ bits (assuming no overhead), i.e. 16.5 GB for one billion of 33-mers. Thus, the BDS is four times more succinct.

Comparison with other hash functions and with a classical Bloom filter

Time comparison with other hash functions The hash functions defined for BDS were designed with speed in mind. In this paragraph, we compare them with a popular and fast hash function (Jenkins hash, specifically `hashlittle2` from http://burtleburtle.net/bob/c/lookup3.c). We simulated 1 millon of 100-bp reads, where each nucleotide is drawn uniformly and independently. To simulate the behavior of computing hashes for the BDS, 4 hash values were computed for each 33-mer. For the `hashlittle2` function, we simulated this behavior by computing 4 hashes with 4 different initial values. We recorded the time required to compute the hashes for all the 33-mers present in the reads, averaged over 3 executions. Computing the hash with the `hashlittle2` function took 13.1 seconds (5.2 MHashes/s), whereas for the BDS hash functions, the same computation took 1.4 seconds (49.8 MHashes/s). Hence, the BDS hash functions are one order of magnitude faster than a classical set of hash functions.

FP rate comparison with other hash functions We can see in Figure 3 that the FP rate of classical hash functions follows the FP rate of our balanced functions (it follows the equation 5 with the exponent 3 being replaced by the number of functions used). However it diverges with more than 3 functions, as we could not add other balanced functions and we added in place unbalanced ones which have higher FP rates. Even if, for more than three functions, classical hash functions produce less FP, the difference with our BDS structure is small: for 1 billion indexed k-mers, combinations of 4 classical functions give 0.01% FP on average, compared to 0.114%. We chose to have a slithly higher FP rate, but with a significant gain in computing time.

Comparison with a classical Bloom filter A classical Bloom filter requires a fixed amount of memory to index n k-mers. Evaluating the Bloom filter memory using formula from Section "*Bloom filter*" for one billion elements, with a false positive rate of 0.114%, yields 1.8 GB of memory. While this is twice smaller than the BDS, a classical Bloom filter would require classical hash functions. However, as shown above, classical hash functions are an order of magnitude slower to compute.

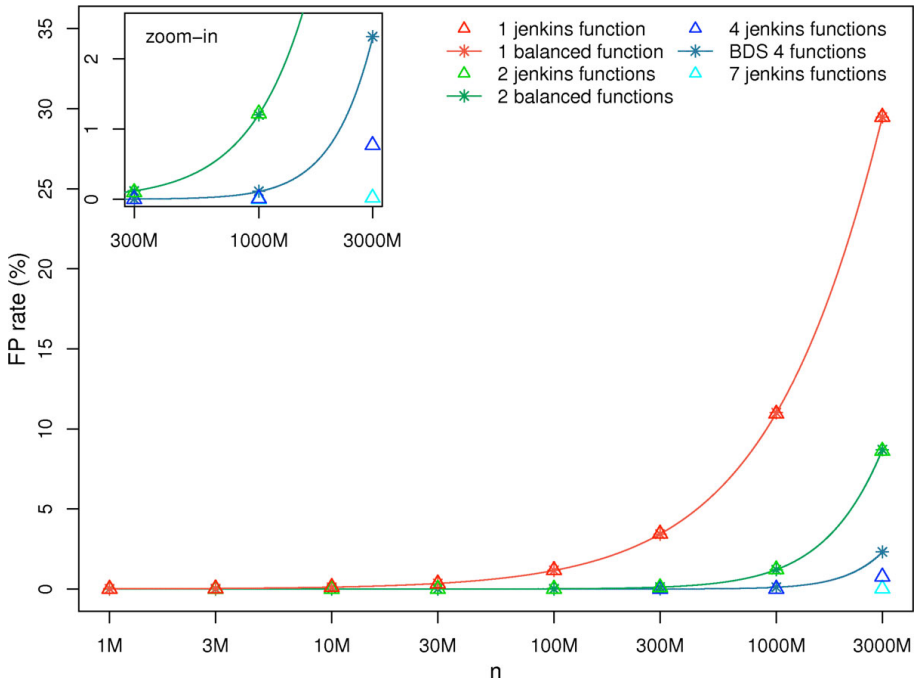


Figure 3 Jenkins versus BDS false positive rate. Comparison of FP rates between classical hash functions and the functions we used in the BDS. FP rate is plotted as a function of the number of indexed *k*-mers (in log scale), with $k = 33$. Plain lines correspond to theoretical predictions for the balanced functions (BDS), whereas star points and triangles correspond to empirical values obtained with simulations using respectively BDS functions and classical hash functions. The combination entitled "BDS 4 functions" is the one chosen for Compareads and is composed of the 3 balanced functions plus one unbalanced.

Comparison with a classical approach using BLAST

Our approach is an heuristic based on shared *k*-mers between reads. Here we compare Compareads with a well-established method, BLAST[21], that is based on sequence alignment. The dataset used is composed of 15 bacterial metagenomes obtained from fresh water with three different conditions of Carbon/Nitrogen ratio (unpublished data). On average, each sample is composed of 176409 reads with an average of 400 nucleotides per read (Roche 454 technology).

Both BLAST and Compareads were used to compute all of the 120 pairwise intersections between the 15 datasets. BLAST was configured to find similar sequences between two samples with a local alignment greater than 80 nucleotides and more than 90% of sequence identity. Compareads was used to find sequences sharing respectively $t = 1$, 4 and 10 *k*-mers of 33 nucleotides. As shown in Table 1, computing one intersection between two samples using Compareads is more than 30 times faster than using BLAST for a close total number of similar reads.

For each experiment, samples were hierarchically clustered based on their pairwise similarity scores and then drawn as a dendrogram. As shown in Figure 4, the dendrogram obtained with the BLAST approach **(a)** is slightly different but the three main branches are the same than with the Compareads approach **(b)**. Interestingly, these branches discriminate three groups of samples corresponding to the three different biological conditions indicated by 1, 10 and 40 in the samples names: 1 corresponds to addition of Carbon in the water, 10 stands for normal condition and 40 for introduction of Nitrogen. Notably, all dendrograms based on Compareads approach **(b, c, d)**

Table 1 Comparison between Compareads and BLAST.

	Total Time (min)	Mean Time for one intersection (s)	Reads Found
BLAST	7200	3600	33 400 091
Compareads 1 * 33	238	119	35 898 023
Compareads 4 * 33	230	115	31 997 243
Compareads 10 * 33	228	114	21 350 268

CPU time per intersection and global CPU time using a single core of an Intel® Xeon® CPU X5550 at 2.67GHz. Reads Found corresponds to the total number of similar reads in all the 120 intersections.

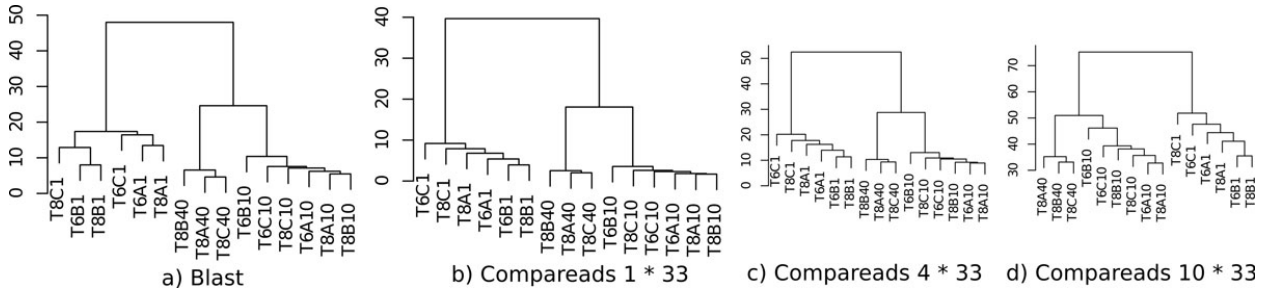


Figure 4 Clustering based on Compareads and BLAST results. Representation of hierarchical clustering based on pairwise intersections between all samples using BLAST (a) and Compareads (b, c, d).

show a similar organization. Increasing the number of shared *k*-mers leads to be more stringent and decreases the number of similar reads but do not affect the global organization of the dendrogram, demonstrating the robustness of our similarity measure.

Applying Compareads to Global Ocean metagenomic samples

We tested Compareads on a larger and famous public dataset from the Global Ocean Sampling (The Sorcerer II expedition) [22]. It is composed of 44 samples from the microbial world of seawater, collected across several thousand of kilometers from the Northwest Atlantic through the Eastern Tropical Pacific oceans and for which an analysis of similarity between samples has been done [22]. The whole dataset is composed of 44 samples containing each on average 174759 long reads (1249 nucleotides per read on average, Sanger technology). Compareads computed all of the 990 intersections in 72 hours and half: on average, one intersection was performed in 4 minutes and 23 seconds on a single core of an Intel® Xeon® CPU X5550 at 2.67GHz. Results presented in Figure 5 are highly similar to those presented in the original publication [22], p.418. Two main groups are well discriminated. The first one, represented in turquoise-blue, groups together almost all samples coming from temperate seawater of the North American East Coast except the 14 one, consistently with the original study. This group also contains two samples really different from all others: the first contains freshwater and the second hypersaline water. The dark-green part corresponds to samples coming from the north part while light-green one gathers samples from the south part. Orange samples correspond to estuary. All of those three groups are identical to the original study. The second main part, colored in yellow, groups together datasets of tropical and Sargasso seawater. The dark-blue part aggregates samples coming only from Galapagos Islands. Red square delimitates Sargasso Sea samples. On the original study, the sample 00a is not in this group. According to metadata, the gray part, like in the original publication, is composed of various samples. Finally, purple samples regroup both Caribbean Sea and some Open Ocean datasets, as the original study.

Those results show that Compareads can also be used on Sanger reads and deliver reliable biological conclusions. Indeed, despite of false positives and the simple definition of similarity, we were able to retrieve the classification of metagenomes according to their geographical origin.

Conclusion

Motivated by *de novo* comparative metagenomics, this paper proposes two main contributions. The first one is a data structure based on Bloom filters that can index, for instance up to one billion distinct words of length 33 (33-mers) using 4Gb of memory, with an error rate of 0.11%, and that is faster to build and request, to the best of our knowledge, that any other existing data structure. The second main contribution is a software, called Compareads which uses this data structure to efficiently perform *de novo* intensive comparisons of huge metagenomic datasets generated by High Throughput Sequencers. We have shown that this approach enables to retrieve and classify differences in species content between metagenomic samples. For this kind of comparison, our approach is much faster than alternative ones such as BLAST and thus enables to scale to huge datasets. We furthermore tested the scalability of Compareads on a large oceanic dataset (unpublished), from the Tara Ocean expedition [15]; it is composed of 31 metagenomes and contains overall 3.5 billions of Illumina short reads (108bp). Each intersection was performed in 10 hours and 55 minutes in average using 4Gb of memory. Such features enabled us to compute the $\frac{31*32}{2}$ = 496 metagenome datasets intersections in 6 days and 10 hours using 50 cores of Intel® Xeon® CPU X5550 at 2.67GHz. This would have been unfeasible with any other known existing tools (based on results Section "*Comparison with a classical approach using* BLAST", BLAST is about 30 times longer and would take more

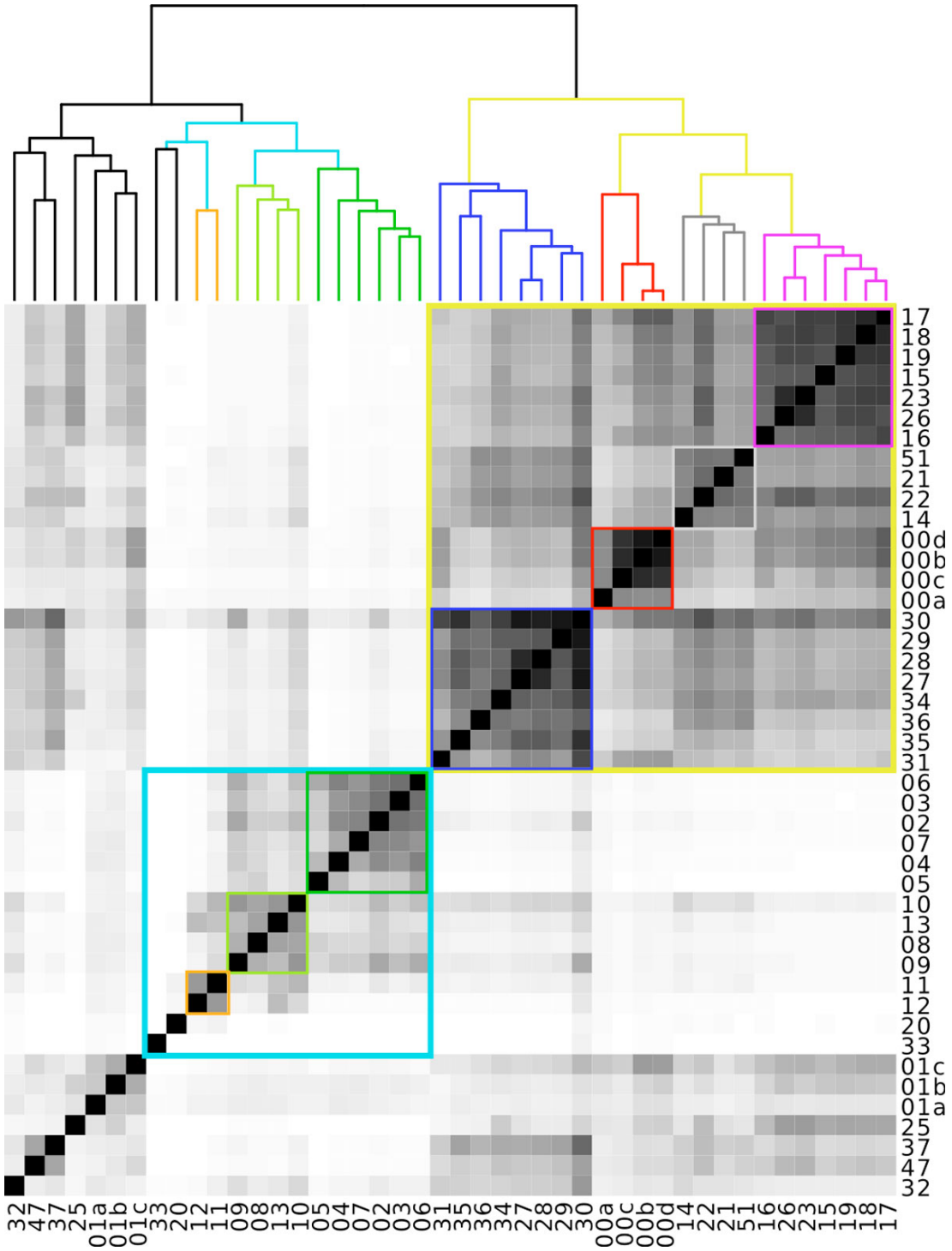


Figure 5 Heatmap of intersections in Global Ocean Sampling. Similarity matrix resulting from the comparison of 44 samples from The *Sorcerer II* Global Ocean Sampling Expedition using Compareads. Grey levels correspond to similarity levels, intersections with more than 50% of similarity are in black. The two main groups, in turquoise-blue and yellow, correspond respectively to north American east coast and tropical samples.

than 6 months to complete this task with the same resources).

Compareads has been conceived for being parallelizable both at fine and coarse grained levels. Future work will consist in implementing a parallel version exploiting multi-core and GPU chips. Compareads is released under the CeCILL license and can be freely downloaded from http://alcovna.genouest.org/compareads/.

Additional material

Additional file 1: Theoretical details for the false positive rate. Details about how theoretical false positive results were obtained. Theoretical details for the false positive rate. Details about how theoretical false positive results were obtained.

Additional file 2: Empirical estimation of false positive rate. Details about how empirical false positive results were obtained. Empirical estimation of false positive rate. Details about how empirical false positive results were obtained.

Acknowledgements
This work was supported by the french ANR-2010-COSI-004 MAPPI Project. Authors warmly thank O. Jaillon from the Genoscope and P. Vandenkoornhuyse from the Ecobio UMR for providing their biological expertise and metagenomic datasets. Additionally, we thank G. Rizk for its help and comments with the data structure and F. Gauthier for the "$\widetilde{\cap}$" symbol creation.
This article has been published as part of *BMC Bioinformatics* Volume 13 Supplement 19, 2012: Proceedings of the Tenth Annual Research in Computational Molecular Biology (RECOMB) Satellite Workshop on Comparative Genomics. The full contents of the supplement are available online at http://www.biomedcentral.com/bmcbioinformatics/supplements/13/S19.

Author details
[1]INRIA Rennes - Bretagne Atlantique/IRISA, EPI GenScale, Rennes, France. [2]ENS Cachan/IRISA, EPI GenScale, Rennes, France.

Authors' contributions
DL and PP initiated the work. RC and CL provided expertise about Bloom filters datastructures and their statistical aspects. NM and PP made the implementations. NM, CL, DL and PP performed the experiments. All authors participated to the redaction and approved the final manuscript.

Competing interests
The authors declare that they have no competing interests.

Published: 19 December 2012

References
1. Desai N, Antonopoulos D, Gilbert JA, Glass EM, Meyer F: **From genomics to metagenomics.** *Curr Opin Biotechnol* 2012, **23**:72-76.
2. Johnson DS, Mortazavi A, Myers RM, Wold B: **Genome-wide mapping of in vivo protein-DNA interactions.** *Science (New York, N.Y.)* 2007, **316(5830)**:1497-502.
3. Licatalosi DD, Mele A, Fak JJ, Ule J, Kayikci M, Chi SW, Clark TA, Schweitzer AC, Blume JE, Wang X, Darnell JC, Darnell RB: **HITS-CLIP yields genome-wide insights into brain alternative RNA processing.** *Nature* 2008, **456(7221)**:464-469.
4. Davey JW, Blaxter ML: **RADSeq: next-generation population genetics.** *Briefings in Functional Genomics* 2010, **9(5-6)**:416-423.
5. Wooley JC, Godzik A, Friedberg I: **A Primer on Metagenomics.** *PLoS Comput Biol* 2010, **6(2)**:e1000667.
6. Amann RI, Ludwig W, Schleifer KH: **Phylogenetic identification and in situ detection of individual microbial cells without cultivation.** *Microbiol Rev* 1995, **59**:143-169.
7. Mende DR, Waller AS, Sunagawa S, Jäelin AI, Chan MM, Arumugam M, Raes J, Bork P: **Assessment of Metagenomic Assembly Using Simulated Next Generation Sequencing Data.** *PLoS ONE* 2012, **7(2)**:e31386.
8. Wang Y, Leung HC, Yiu SM, Chin FY: **MetaCluster 4.0: a novel binning algorithm for NGS reads and huge number of species.** *J Comput Biol* 2012, **19(2)**:241-249.
9. Markowitz VM, Chen IM, Chu K, Szeto E, Palaniappan K, Grechkin Y, Ratner A, Jacob B, Pati A, Huntemann M, Liolios K, Pagani I, Anderson I, Mavromatis K, Ivanova NN, Kyrpides NC: **IMG/M: the integrated metagenome data management and comparative analysis system.** *Nucleic Acids Res* 2011 [http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3245048].
10. Huson DH, Auch AF, Qi J, Schuster SC: **MEGAN analysis of metagenomic data.** *Genome Res* 2007, **17(3)**:377-386.
11. Foerstner KU, von Mering C, Hooper SD, Bork P: **Environments shape the nucleotide composition of genomes.** *EMBO Rep* 2005, **6(12)**:1208-1213.
12. Raes J, Korbel JO, Lercher MJ, von Mering C, Bork P: **Prediction of effective genome size in metagenomic samples.** *Genome Biol* 2007, **8**:R10.
13. Jaenicke S, Ander C, Bekel T, Bisdorf R, Droge M, Gartemann KH, Junemann S, Kaiser O, Krause L, Tille F, Zakrzewski M, Puhler A, Schluter A, Goesmann A: **Comparative and joint analysis of two metagenomic datasets from a biogas fermenter obtained by 454-pyrosequencing.** *PLoS ONE* 2011, **6**:e14519.
14. Sommer MO, Dantas G, Church GM: **Functional characterization of the antibiotic resistance reservoir in the human microflora.** *Science* 2009, **325(5944)**:1128-1131.
15. Karsenti E: **Towards an 'Oceans Systems Biology'.** *Molecular Systems Biology* 2012, **8(575)**:1-2 [http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3321524].
16. Bloom BH: **Space/time trade-offs in hash coding with allowable errors.** *Commun ACM* 1970, **13(7)**:422-426.
17. Pell J, Hintze A, Brown T, Canino-koning R, Howe A, Tiedje JM: **Scaling metagenome sequence assembly with probabilistic de Bruijn graphs.** *PNAS* 2012, **109(33)**:13272-13277 [http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3421212].
18. Broder A, Mitzenmacher M: **Network applications of bloom filters: A survey.** *Internet Mathematics* 2004, **1(4)**:485-509.
19. Abouelhoda MI, Kurtz S, Ohlebusch E: **Replacing suffix trees with enhanced suffix arrays.** *Journal of Discrete Algorithms* 2004, **2**:53-86.
20. Vyverman M, De Baets B, Fack V, Dawyndt P: **Prospects and limitations of full-text index structures in genome analysis.** *Nucleic acids research* 2012, 1-23 [http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3424560].
21. Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ: **Basic local alignment search tool.** *J Mol Biol* 1990, **215**:403-410.
22. Rusch DB, Halpern AL, Sutton G, Heidelberg KB, Williamson S, Yooseph S, Wu D, Eisen JA, Hoffman JM, Remington K, Beeson K, Tran B, Smith H, Baden-Tillson H, Stewart C, Thorpe J, Freeman J, Andrews-Pfannkoch C, Venter JE, Li K, Kravitz S, Heidelberg JF, Utterback T, Rogers YH, Falcon LI, Souza V, Bonilla-Rosso G, Eguiarte LE, Karl DM, *et al*: **The Sorcerer II Global Ocean Sampling expedition: northwest Atlantic through eastern tropical Pacific.** *PLoS Biol* 2007, **5(3)**:e77.

doi:10.1186/1471-2105-13-S19-S10
Cite this article as: Maillet *et al.*: Compareads: comparing huge metagenomic experiments. *BMC Bioinformatics* 2012 **13**(Suppl 19):S10.

COMMET : comparing and combining multiple metagenomic datasets

COMMET: comparing and combining multiple metagenomic datasets

Nicolas Maillet*, Guillaume Collet†, Thomas Vannier‡, Dominique Lavenier* and Pierre Peterlongo*⁕
*INRIA / IRISA-UMR CNRS 6074, EPI GenScale, Rennes, France
†INRIA / IRISA-UMR CNRS 6074, EPI Dyliss, Rennes, France
‡CEA Genoscope / CNRS UMR 8030 / Université d'Évry, Evry, France
⁕ Corresponding author: Pierre Peterlongo pierre.peterlongo@inria.fr

***Abstract*—Metagenomics offers a way to analyze biotopes at the genomic level and to reach functional and taxonomical conclusions. The bio-analyzes of large metagenomic projects face critical limitations: complex metagenomes cannot be assembled and the taxonomical or functional annotations are much smaller than the real biological diversity. This motivated the development of *de novo* metagenomic read comparison approaches to extract information contained in metagenomic datasets.**

However, these new approaches do not scale up large metagenomic projects, or generate an important number of large intermediate and result files. We introduce COMMET ("COmpare Multiple METagenomes"), a method that provides similarity overview between all datasets of large metagenomic projects.

Directly from non-assembled reads, all against all comparisons are performed through an efficient indexing strategy. Then, results are stored as bit vectors, a compressed representation of read files, that can be used to further combine read subsets by common logical operations. Finally, COMMET computes a clusterization of metagenomic datasets, which is visualized by dendrogram and heatmaps.

Availability: http://github.com/pierrepeterlongo/commet

I. INTRODUCTION

NGS revolution enabled the emergence of the metagenomic field where an environment is sequenced instead of an individual or a species, opening the way to a comprehensive understanding of environmental microbial communities. Large metagenomic projects such as MetaSoil [1], MetaHit [2] or Tara Oceans [3] witness this evolution. Analyzing of metagenomic data is a major bottleneck. For instance, assembly tests over "simple" simulated metagenomes showed that N50 is only slightly larger than read sizes [4]. This situation becomes even worth on complex datasets, such as seawater, where millions of distinct species coexist. In this case, biodiversity can be estimated by using statistical approaches [5] or by mapping reads on reference banks [6], [7]. Nevertheless, statistical approaches are limited to a few dozens of species with limited differences in their relative abundance. In addition, the mapping approaches are limited to current knowledge contained in reference banks that suffer from their incompleteness and their inherent errors [8].

A key point of substantial metagenomic projects stands in the number of metagenomes they produce. Then, similarities and differences between metagenomes can be exploited as a source of information, measuring external effects like pollution sources, geographic locations, and patient microbial gut environment [2], [9]. A few methods were proposed to compare metagenomes using external information sources such as taxonomic diversity [10] or functional content [11]. However, these methods are biased because as they are based on partial knowledge.

Methods were proposed to compare metagenomes without using any *a priori* knowledge. These *de novo* methods use global features like GC content [12], genome size [13] or sequence signatures [14]. Theses methods face limitations as they are based on rough imprecise criteria and as they only compute a similarity distance: they do not extract similar elements between samples. We believe that it is possible to go further by comparing metagenomic samples at the read sequence level. This provides a higher precision distance and, importantly, it provides reads that are similar between datasets or that are specific to a unique dataset, enabling their latter analysis: assembly with better coverage or comparison with other metagenomic samples. Such comparisons may be performed using Blast [15] or Blat [16] like tools. Unfortunately, these methods do not scale up on large comparative metagenomic studies in which hundreds of millions of reads have to be compared to other hundreds of millions of reads. For instance, one can estimate that comparing a hundred of metagenomes each composed by a hundred of millions of reads of size 100 would require centuries of CPU computation. The crAss approach [17] constructs a reference metagenome by cross assembling reads of all samples. Then, it maps the initial reads on the so obtained contigs and several measures are derived, based on the repartition of mapped reads. This method provides results of high quality. However, due to its assembly and mapping approach, it does not scale up to large metagenomic datasets. Simpler methods such as TriageTools [18] or Compareads [19] measure the sequence similarity of a read with a databank by counting the number of k-mers (words of length k) shared with the databank. Due to memory consumption, TriageTools cannot use k values larger than 15 and is thus limited to small datasets (a few hundred of thousands reads of length 100). The Compareads tool scales up to large datasets with a small memory footprint and acceptable running time. However, applied on large metagenomic projects, this tools generates an important number of large intermediate result files. In practice, applying Compareads to N datasets generates N^2 resulting new datasets, each of the size of the original ones at worse. Additionally, Compareads leads to highly redundant computation raising up the execution time. These drawbacks are serious bottlenecks limiting the practical usage of Compareads.

In this paper, we introduce COMMET ("COmpare Multiple METagenomes"), a fast software that provides a global similarity overview between all datasets of a metagenomic project. COMMET is based on the Compareads philosophy that consists in determining similarity between two metagenomic datasets by extracting common reads using k-mer approach: two reads are considered similar if they share t non-overlapping k-mers (t and k are parameters). A metagenomic project involving N datasets will thus requires the computation of N^2 intersections which is both time- and storage-consuming. To keep computation time as low as possible, the computation of the N^2 intersections has been strongly improved compared to the Compareads approach through an efficient indexing strategy in which each file is fully indexed only once. In addition, to save storage space, intersections between metagenomic datasets are represented as bit vectors. This compact representation reduces the storage space by two orders of magnitude. Moreover, it provides an easy way to filter and sub-sample reads, or to combine various results by applying logical operations. Finally, COMMET computes a clusterization of metagenomic datasets, which is visualized by dendrogram and heatmaps.

II. METHOD

A. Comparing two sets of reads

The COMMET algorithm to compare two sets of read is based on the Compareads [19] methodology. It consists in finding reads from a set A that are similar to at least one read from a set B. The similarity between two reads is based on a minimal number t of non-overlapping identical k-mers. This core operation is directed : it provides reads from A similar to reads from B but it does not provides reads from B similar to reads from A. Note that, as explained below, this operation is based on a heuristic. Thus we denote this operation by $A \overrightarrow{\cap} B$.

Computing $A \overrightarrow{\cap} B$ consists in two steps. Firstly, k-mers from B are indexed in a Bloom filter like data-structure [20]. Secondly, non-overlapping k-mers of reads from A are searched in the Bloom filter. A read r from A sharing t non-overlapping k-mers with the Bloom filter is considered similar to at least one read from B. However, the algorithm does not check that these k-mers co-occur on a single read from B, which is a source of false positives. Readers are invited to refer to [19] for having more details on precision results.

We recall that the following strategy is applied in order to limit the second source of false positives. First $A \overrightarrow{\cap} B$ is computed. Then, instead of navely computing $B \overrightarrow{\cap} A$, $B \overrightarrow{\cap} (A \overrightarrow{\cap} B)$ is computed. This limits the indexed reads of A to those already detected as similar to at least one read from B. Finally, the symmetrical operation is performed: $A \overrightarrow{\cap} (B \overrightarrow{\cap} (A \overrightarrow{\cap} B))$.

The previously exposed strategy to fully compare sets A and B within three consecutive $\overrightarrow{\cap}$ operations has also the advantage to limit the indexation effort. Indeed, only the first $A \overrightarrow{\cap} B$ operation indexes the full set B. The two other operations only index subsets of A and B.

While comparing read samples A and B, the final results of interest are the reads of A similar to reads of B computed by $A \overrightarrow{\cap} (B \overrightarrow{\cap} (A \overrightarrow{\cap} B))$ and reads of B similar to reads of A computed by $B \overrightarrow{\cap} (A \overrightarrow{\cap} B)$. For sake of simplicity, we denotes these two sets as, respectively, $A \overrightarrow{\widehat{\cap}} B$ and $B \overrightarrow{\widehat{\cap}} A$.

In the following sections we present the COMMET novelties: represent read subsets with a limited disk space impact, new read filtering and read subsets manipulation features, compare multiple sets of reads, visualize dataset's similarities as heatmaps and dendrogram.

B. Read subsets representation

In COMMET we propose a simple yet compact data structure to represent a read subset: a vector of bits where each bit represents a read of the original read set. This is what we call the "bit vector representation". As shown below, this representation enables to filter and to subsample read files, to represent $\overrightarrow{\cap}$ (and thus $\overrightarrow{\widehat{\cap}}$) results and to easily perform logical operation between read subsets.

Note that with such a representation, a bit vector needs hundreds to thousand times less disk space than a classical uncompressed fastq file. Note also that this way of coding read subsets is not limited to the COMMET framework. It may be applied to any other programs that manipulate read subsets. Thus, the COMMET tool includes a C++ library of reusable components to manipulate read subsets.

In the COMMET framework, the bit vector representation is used as inputs and/or outputs of all tools. In particular they are used in the following operations:

1) Read subsampling and filtering: With huge datasets, it may appear necessary to subsample, for instance limiting each read file to a same number m of a few millions reads. This is immediate by creating a bit vector in which only the first m bits are set to 1, while others are set to 0.

Raw NGS reads also usually need to be filtered on several practical characteristics (read size, read complexity, ...). Thus, a bit vector is a direct representation of a filtered result: bit values associated to selected reads are set to 1, the others to 0. A combination of subsampling and filtering allows to select only the m first reads that fulfill the filtration criteria.

2) Representing the similar reads: Results of any $\overrightarrow{\cap}$ operation is represented by a bit vector. Bit values of reads from the query set detected as similar to at least one read from the reference set are set to 1 and the others are set to 0.

3) Compute logical operations on read subsets: The bit vector representation is ideally suited to perform fundamental logical operations. COMMET provides a module to perform the AND, OR and NOT operations between distinct subsets of a single initial set of reads.

As presented in the simple case study (Section II-F), these operations, although simple, are powerful while dealing with read subsets. They allow to combine comparison results and so to focus on read subsets intersections or exclusions.

These logical operations perform very efficiently, both in terms of execution time and memory footprint. Moreover, it is worth to notice that they do not generate large result files, as results of these logical operations are also represented as bit vectors. This allows to intensively manipulate read subsets with no technical limitations.

C. Dealing with more than two datasets

We recall that the computation of the $A \overrightarrow{\cap} B$ core operation involves indexation and search. Once the k-mers of the reads from B are indexed, then the k-mers of the reads from A are sequentially search in the index. If more than a threshold number t of such k-mers are find in the index, then the given read from A is considered as similar to a read from B, which means that the associated value in the bit vector is set to 1.

Consider $S = \{R_1, \ldots, R_N\}$ a set of $N \geq 2$ read sets. Applying COMMET on the whole S implies that $\forall (i,j) \in [1,N]^2, i < j$, three ordered operations are performed:

1) $R_i \overrightarrow{\cap} R_j$
2) $R_j \overrightarrow{\Cap} R_i = R_j \overrightarrow{\cap} (R_i \overrightarrow{\cap} R_j)$
3) $R_i \overrightarrow{\Cap} R_j = R_i \overrightarrow{\cap} (R_j \overrightarrow{\cap} (R_i \overrightarrow{\cap} R_j))$

Note that for each couple (i,j), the order (i,j) or (j,i) only slightly changes the overall results of the three operations. To avoid redundancies, we limit these operations to $i < j$.

1) Factorizing the indexation: In practice, applying COMMET on S implies to perform the $R_i \overrightarrow{\cap} R_j$ operations for all $i < j$. In particular, $R_1 \overrightarrow{\cap} R_N \ldots R_{N-1} \overrightarrow{\cap} R_N$ have to be computed. For these $N-1$ computations, the k-mer index of R_N is the same. To avoid redundancies, the R_N index is computed only once and the $N-1$ remaining sets are compared to R_N using this single index. In general, while R_{ref} ($ref \in [2,N]$) is indexed, the index is conserved in RAM memory during the computation of the $ref - 1$ comparisons $R_{query} \overrightarrow{\cap} R_{ref}$, with $query < ref$.

2) Results visualization: Comparisons of $N \geq 2$ read sets $\{R_1, \ldots, R_N\}$ provide useful metrics that give an overview of the genomic diversity of the studied samples. Those metrics are summarized in three matrices M_1, M_2, and M_3 with values calculated as follows:

- $M_1(i,j) = |R_i \overrightarrow{\Cap} R_j|$
- $M_2(i,j) = 100 \times \frac{|R_i \overrightarrow{\Cap} R_j|}{|R_i|}$
- $M_3(i,j) = 100 \times \frac{|R_i \overrightarrow{\Cap} R_j| + |R_j \overrightarrow{\Cap} R_i|}{|R_i| + |R_j|}$.

$M_1(i,j)$ with $(i,j) \in [1,N]^2$, is the raw number of reads from R_i that are similar to at least one read from R_j. As read sets may be of different sizes, $M_2(i,j)$ is the percentage of reads from R_i similar to at least one read from R_j. Those two first matrices are asymmetrical. M_3 is a symmetrical matrix. $M_3(i,j)$ is the percentage of similar reads between the two sets with respect to the total number of reads in R_i and R_j.

For each matrix, a heatmap is generated . Additionally, M_3 is used to construct a dendrogram representation by hierarchical clustering (see Fig 2 for an example of a heatmap and a dendrogram generated by COMMET).

D. The COMMET modules

COMMET integrates four independent modules written in C++, all manipulating, as inputs and outputs, the bit vector representation of read subsets. Additionally, COMMET provides a python script (Commet.py) that takes $N \geq 2$ read sets, filters them, compares them and generates explicit representations of comparative results, see Section II-E.

1) Filtering and subsampling reads: Thanks to the first module, $filter\_reads$, each read of each dataset (fasta or fastq format, gzipped or not) is filtered out according to user-defined criteria: minimal read length, number of undefined bases, and Shannon complexity [21], used to remove low complexity sequences. The result is a bit vector for each input read file. $Filter\_reads$ can also subsample each read set by limiting the number of selected reads to a user defined parameter m. The m first reads that passed the filters are selected.

2) Performing the $\overrightarrow{\cap}$ core operation: The second module, $index\_and\_search$, performs the $\overrightarrow{\cap}$ core operation, representing results using the bit vector representation. It inputs a set of read sets (the queries) to be searched in an indexed read set (the bank). A read set may be composed of several read files. Each file could be associated to a bit vector. In this latter case, $index\_and\_search$ only considers reads whose associated bit values are set to 1.

3) Manipulating read subsets: The third module, $bvop$ (bit vector operations) inputs one or two bit vectors. In this second case, the two bit vectors should represent subsets of the same initial set. This module performs the NOT operation on a single bit vector, and the AND, OR, and $AND\ NOT$ operations on two bit vectors.

4) From bit vectors to read files: Given an original read file and its bit vector, the last module, $extract\_reads$, generates an explicit representation of any read subset.

E. Automatization for $N \geq 2$ read sets

COMMET includes a python script (Commet.py) which inputs $N \geq 2$ read sets. This pipeline i) filters reads, given user-defined parameters, ii) compares all-against-all read sets, and iii) outputs a user-friendly visualization of results. The outputs consist in the three matrices in csv format, their heatmaps and a dendrogram as described in Section II-C2. The dendrogram is realized using the *hclust R* function, computing a hierarchical complete clustering.

F. Combining read subsets use case

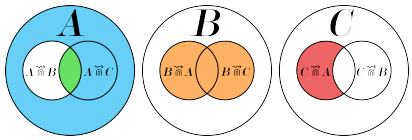


Fig. 1. Logical operations on intersections between A, B and C extract read subsets of interest. The blue subset corresponds to the $A\ AND\ NOT(A \overrightarrow{\Cap} B)$ operation. The green subset corresponds to the $(A \overrightarrow{\Cap} B)\ AND\ (A \overrightarrow{\Cap} C)$ operation. The orange subset corresponds to the $(B \overrightarrow{\Cap} A)\ OR\ (B \overrightarrow{\Cap} C)$ operation. The red subset corresponds to the $(C \overrightarrow{\Cap} A)\ AND\ NOT(C \overrightarrow{\Cap} B)$ operation.

By using the $bvop$ module, logical operations can be performed between inputs/outputs of the COMMET pipeline output. For instance, reads from A not similar to any read from set B (blue subset of Fig 1) are obtained by first applying $NOT(A \overrightarrow{\Cap} B)$ operation. Reads from A similar to at least one read from B and one read from C (green subset

TABLE I. 28 METAGENOMES FROM THE IMG/M DATABASE

Identifiers	Description
SWITGRA	Rhizosphere soil from *Panicum virgatum*
SUBGIN	Oral TM7 microbial community of Human
TERMITE2, TERMITE1	Gut microbiome of divers termites
SOILM, SOILD, SOILL	Soil microbiome from divers locations
OMIN, MESO, EUPHO	Divers marine planktonic communities
BEETLE	*Dendroctonus ponderosae*
ACOFUNT, ACOFUNB, CLOFUN, ACEFUN, TRAFUN, FUNCOMB	Fungus garden of divers ants
FUNTER	Fungus-growing termite worker
WALLABY	Forestomach microbiome of tammar wallaby
RICE	Endophytic microbiome from rice
SNAIL	*Achatina fulica*
SNOCT	*Sirex noctilio* microbiome
XALARV, XAAD	*Xyleborus affinis* microbiome (larvae, adult)
HGUT7, HGUT8	Human gut community
PANDA2, PANDA5	Wild panda gut microbiome

of Fig 1), are identified by computing the AND operation: $(A \overset{\rightarrow}{\widehat{\cap}} B)\ AND\ (A \overset{\rightarrow}{\widehat{\cap}} C)$. In the same spirit, reads from B similar to at least one read from A or one read from C (orange subset of Fig 1), are found by computing the OR operation: $(B \overset{\rightarrow}{\widehat{\cap}} A)\ OR\ (B \overset{\rightarrow}{\widehat{\cap}} C)$. Operations may be combined to obtain more complex results as, for instance, the red subset of Fig 1, representing reads from C similar to at least one read from A, but not similar to any read from B. This would be done by applying the $(C \overset{\rightarrow}{\widehat{\cap}} A)\ AND\ NOT (C \overset{\rightarrow}{\widehat{\cap}} B)$ operation.

III. RESULTS

A. COMMET *efficiently compares multiple metagenomes*

We tested COMMET on a set of 28 metagenomes from the IMG/M database [7] (see Table I). These 28 metagenomes were compared with options $k = 33$, $t = 2$ and $m = 10000$. Computations were done using COMMET (Commet.py) and Compareads (v1.3.1) on a 2.9 GHz Intel Core i7 processor with 8GB of RAM and a Solid-State Drive. COMMET calculated the 756 intersections in 35 minutes while Compareads took 81 minutes. In this experiment, COMMET is 2x faster than Compareads thanks to its indexing strategy (each file is fully indexed only once). The obtained dendrograms, shown in Figure 2, are biologically coherent. The different fungus samples are grouped together as well as soil samples, marine planktonic communities and insects. The two human gut microbiome samples are far from other species, as well as the two panda gut microbiome samples.

B. Metasoil study

The MetaSoil study focuses on untreated soils of Park Grass Experiment, Rothamsted Research, Hertfordshire, UK. One of the goals of this study is to assess the influence of depth, seasons and extraction procedure on the sequencing [22]. To achieve this, the 13 metagenomes from MetaSoil, two other soil metagenomes and a sea water metagenome, were compared at the functional level using MG-RAST [23]. This approach identified 835 functional subsystems present in at least one of those metagenomes. On Figure 3.a, samples were clustered using the relative number of reads associated with the 835 functions. This figure shows that the extraction procedure correlates with sample clusters: two metagenomic samples processed with the same extraction procedure share more similarities at the functional level than two samples processed with different extraction procedures [1].

This study was reproduced with COMMET on all available metagenomes. The generated bit vectors weigh 68MB while

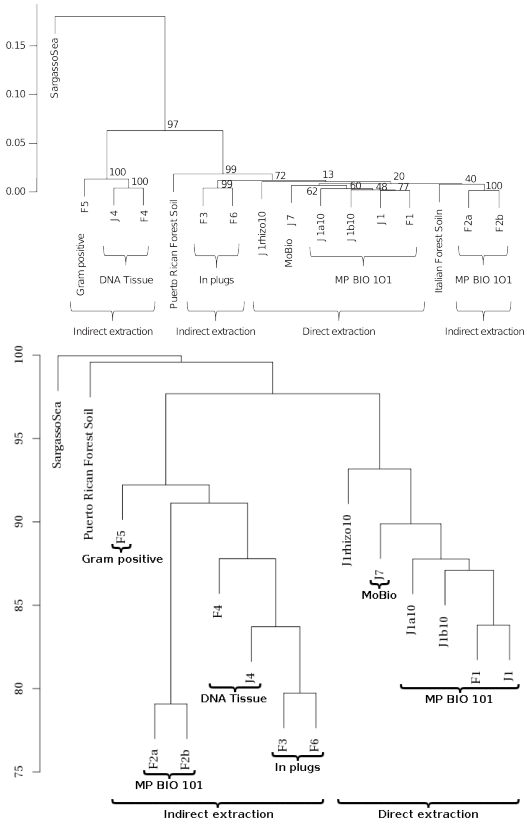


Fig. 3. Dendrograms from MetaSoil study (top, figure from [1]) and COMMET analysis (bottom), comparing the 13 MetaSoil samples, an other soil metagenome and a seawater metagenome (Sargasso Sea).

the explicit representation of the fasta results requires 6.4GB. The storage footprint is thus divided by a factor 100. This ratio is even higher if using fastq format or if dealing with larger read files. The COMMET computation time was 828 minutes (the same set treated by Compareads took 2981 minutes).

Although COMMET uses another metric, the produced dendrogram is highly similar to the MetaSoil one (see Fig 3). On both dendrograms, samples coming from direct extraction are clustered together and external metagenomes are far from the MetaSoil's. Moreover, on the COMMET dendrogram, all samples coming from indirect extraction are clustered together, which is not the case in the MetaSoil study. Even if the two comparing methods are different, they lead to the same conclusion: extraction procedures have a critical impact on sequencing.

IV. CONCLUSION

COMMET gives a global similarity overview of all datasets of a large metagenomic project. It performs all-against-all comparisons of N datasets by factorizing indexation phases. Disk I/Os and storage footprint are highly limited thanks to a new read subset representation which reduces the storage space by at least two orders of magnitude compare to explicit fasta or fastq format. Interestingly, this read subset representation is a powerful way to compute extremely fast boolean operations between read subsets without copying large read files. This enables to focus on reads that fulfill several distinct constraints of interest. The advantages of this representation and of

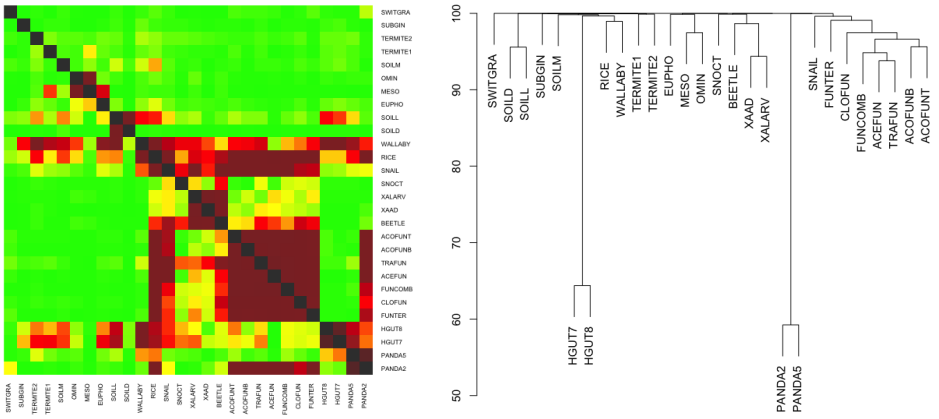


Fig. 2. Heatmap (left) and dendrogram (right) representation of the results of the comparison of 28 datasets from the IMG/M database. Results are given with $t = 2$, $m = 10000$ and $k = 33$. The heatmap is constructed from the matrix M_2 and is thus asymmetrical. The dendrogram is constructed from the matrix M_3 by the hierarchical clustering procedure available in *R* (method "complete").

the boolean manipulation are not limited to the COMMET framework. Thus, COMMET includes a C++ library of reusable components to manipulate read subsets.

COMMET produces graphical outputs that sum up all-against-all comparisons results and open the way for further statistical analysis, thanks to the provided similarity matrices.

A future work will consists in quickly identify significant clusters of read sets by applying rougher comparative metrics (such as the GC content) or a statistical framework based on Principal Component Analysis (PCA). Then, COMMET should be used to go further by precisely compute the shared reads between read sets inside clusters, or between clusters.

COMMET is available under the A-GPL license: http://github.com/pierrepeterlongo/commet.

ACKNOWLEDGMENT

Authors warmly thank Claire Lemaitre for her precious advices and her help designing the *R* functions. This work was supported by the french ANR-2010-COSI-004 *MAPPI* and by the ANR-12-BS02-0008 *Colibread* projects. Guillaume Collet's work is funded by the investment expenditure program IDEALG 1192 ANR-10-BTBR-02-04-11.

REFERENCES

[1] T. O. Delmont *et al.*, "Structure, fluctuation and magnitude of a natural grassland soil metagenome," pp. 1677–1687, 2012.

[2] J. Qin *et al.*, "A human gut microbial gene catalogue established by metagenomic sequencing," *Nature*, vol. 464, no. 7285, pp. 59–65, Mar. 2010. [Online]. Available: http://dx.doi.org/10.1038/nature08821

[3] E. Karsenti *et al.*, "A holistic approach to marine Eco-systems biology," *PLoS Biology*, vol. 9, 2011.

[4] M. Pignatelli and A. Moya, "Evaluating the fidelity of De Novo short read metagenomic assembly using simulated data," *PLoS ONE*, vol. 6, 2011.

[5] Y. Wang *et al.*, "Metacluster 5.0: A two-round binning approach for metagenomic data for low-abundance species in a noisy sample," *Bioinformatics*, vol. 28, 2012.

[6] D. H. Huson *et al.*, "MEGAN analysis of metagenomic data." *Genome research*, vol. 17, pp. 377–386, 2007.

[7] V. M. Markowitz *et al.*, "IMG/M: The integrated metagenome data management and comparative analysis system," *Nucleic Acids Research*, vol. 40, 2012.

[8] A. M. Schnoes *et al.*, "Annotation error in public databases: Misannotation of molecular function in enzyme superfamilies," *PLoS Comput Biol*, vol. 5, no. 12, p. e1000605, Dec 2009.

[9] T. O. Delmont, P. Simonet, and T. M. Vogel, "Describing microbial communities and performing global comparisons in the omic era," pp. 1625–1628, 2012.

[10] S. Jaenicke *et al.*, "Comparative and joint analysis of two metagenomic datasets from a biogas fermenter obtained by 454-pyrosequencing," *PLoS ONE*, vol. 6, 2011.

[11] M. O. A. Sommer, G. Dantas, and G. M. Church, "Functional characterization of the antibiotic resistance reservoir in the human microflora." *Science (New York, N.Y.)*, vol. 325, pp. 1128–1131, 2009.

[12] K. U. Foerstner *et al.*, "Environments shape the nucleotide composition of genomes." *EMBO reports*, vol. 6, pp. 1208–1213, 2005.

[13] J. Raes *et al.*, "Prediction of effective genome size in metagenomic samples." *Genome biology*, vol. 8, p. R10, 2007.

[14] B. Jiang *et al.*, "Comparison of metagenomic samples using sequence signatures." *BMC genomics*, vol. 13, p. 730, Jan. 2012.

[15] S. F. Altschul *et al.*, "Basic local alignment search tool," *Journal of Molecular Biology*, vol. 215, no. 3, pp. 403–410, 1990. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0022283605803602

[16] W. J. Kent, "BLAT—The BLAST-Like Alignment Tool," *Genome Research*, vol. 12, no. 4, pp. 656–664, Mar. 2002. [Online]. Available: http://www.genome.org/cgi/doi/10.1101/gr.229202

[17] B. E. Dutilh *et al.*, "Reference-independent comparative metagenomics using cross-assembly: crAss." *Bioinformatics (Oxford, England)*, vol. 28, no. 24, pp. 3225–31, Dec. 2012. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/23074261

[18] D. Fimereli, V. Detours, and T. Konopka, "TriageTools : tools for partitioning and prioritizing analysis of high-throughput sequencing data," pp. 1–8, 2013.

[19] N. Maillet *et al.*, "Compareads: comparing huge metagenomic experiments." *BMC bioinformatics*, vol. 13 Suppl 1, no. Suppl 19, p. S10, Jan. 2012.

[20] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," vol. 13, pp. 422–426, 1970.

[21] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, pp. 379–423, 1948. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/9230594

[22] T. O. Delmont *et al.*, "Accessing the soil metagenome for studies of microbial diversity," *Applied and Environmental Microbiology*, vol. 77, no. 4, pp. 1315–1324, Feb 2011.

[23] F. Meyer *et al.*, "The metagenomics RAST server – a public resource for the automatic phylogenetic and functional analysis of metagenomes," *BMC bioinformatics*, vol. 9, no. 1, p. 386, Jan 2008.

Simka : fast kmer-based method for estimating the similarity between numerous metagenomic datasets

Simka: fast kmer-based method for estimating the similarity between numerous metagenomic datasets

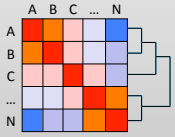
Gaëtan Benoit[1], Pierre Peterlongo[1], Dominique Lavenier[1], Claire Lemaitre[1]
[1] Inria/IRISA GenScale, Campus de Beaulieu, 35042 Rennes cedex, France.
gaetan.benoit@inria.fr, claire.lemaitre@inria.fr, pierre.peterlongo@inria.fr

Metagenomic

A liter of sea water:
- Hundred of millions of species
- > 90% unknown species

Comparative metagenomics

N metagenomic samples
N > 1000 in Tara Oceans project



Similarity heatmap N^2 with hierarchical clustering

fasta fasta ... fasta

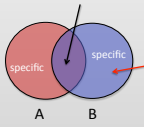
N read-sets
Usually **>100M** reads per dataset

Similarity between 2 read-sets

Idea: similarity is given by the **size of the intersection**

Intersection = number of **similar reads**

Similar: their alignment score > 90%



100M reads

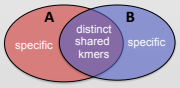
✖ Blast-like approach required **months** of computation for a single intersection

Commet [1] (state of the art)
- Heuristic: Two read are similar if they share some **kmers**
- Computes one intersection in **few hours**
- **Still does not scale on large metagenomic projects**
  - **N(N-1) intersections to compute**

Kmer-based similarity functions

- We have **very fast** methods for **indexing** and **querying** kmers
- A read-sets is now view as a **set of its kmers**
- We define new pairwize similarity measures based on **shared kmers**

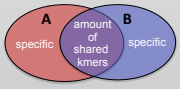
1) Presence / Absence of kmers



Similarity based on presence/absence of species

$$Jaccard(A,B) = \frac{DistinctKmers(A \cap B)}{DistinctKmers(A \cup B)}$$

2) Abundance of shared kmers



Similarity based on abundance of species

$$AbundanceJaccard(A,B) = \frac{\sum_{w \in A \cap B} N_A(w) + N_B(w)}{\sum_{w \in A \cup B} N_A(w) + N_B(w)}$$

w: kmer, *Ni(w)*: abundance of *w* in read-set *i*

Method

A B C ... N
fasta fasta fasta ... fasta

To scale on N large datasets, we count the kmers of N datasets simultaneously.

GATB[2] - DSK multi-dataset kmer counting

Kmer's abundances ***v***

	A	B	C	...	N
ACGTAT	0	4	52	...	0

For each kmer, we get a vector of its abundances in N datasets

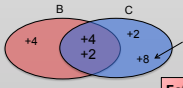
Is kmer solid ? (optional)

Yes

Update intersections

Kmer shared by B and C

	A	B	C
ACC	0	4	2



Kmer specific to C

	A	B	C
TGC	0	0	8

Execution time repartition:
- Counting kmers O(N): 75%
- Updating intersections **O(N^2)**: 25%

Estimating similarity is now a problem of counting kmers

Filtering sequencing errors

Kmer solid:
A kmer that contains no sequencing errors

Statistics:
80% of distinct kmers appear **only one time** and **only in a single dataset**

Noisy data:
It is hard to distinguish erroneous kmers from unique genomic kmers

We defined a solidity definition that can saved some unique kmers:
Given a solidity threshold ***t*** (***t***=2), a kmer ***w*** is solid if:

abundance(***w***) in at least one dataset >= ***t***

Examples on 4 datasets:

A	B	C	D
2	0	1	0

✓

A	B	C	D
52	0	1	1

✓

A	B	C	D
0	1	1	0

✗

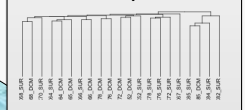
Statistics on solid 31-mers from 21 Tara Oceans samples:
- Distinct: 18G (15% of all distinct kmers)
- Abundance: 106G (49% of all kmers)

Datasets

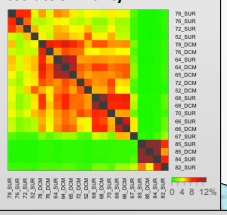
Simka was tested and compared to **Commet**[1] (state of the art) on **21 Tara Oceans samples**:
- 3G reads
- 400 GB of data
- 219G 31-mers
- 123G distinct 31-mers

Presence / Absence of kmers

Relative similarity:



Absolute Similarity:

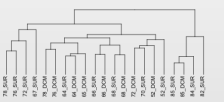


5 Hours

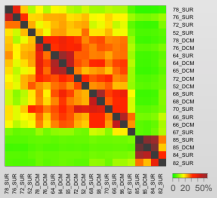
■ Can differs from Commet because not based on species's abundance
✓ Simka is the first method to provide quickly information about presence and absence of species

Abundance with error filter

Relative similarity:



Absolute Similarity:

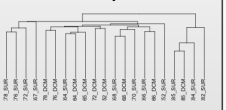


5 Hours

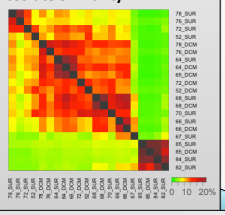
✓ Clustering less noisy and more readable
✓ Close to Commet while using only 18% of distinct kmers (half of the datasets)

Abundance of shared kmers

Relative similarity:



Absolute Similarity:

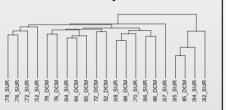


5 Hours

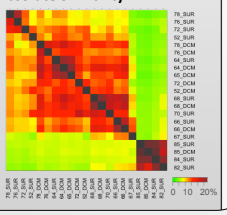
✓ Similarity and clustering close to read-based methods

Commet (state of the art)

Relative similarity:



Absolute Similarity:



Weeks

✓ Read-based
✗ Too slow on large metagenomic projects

Alignment based methods

Years

Perspectives

- Filtering input reads
- Provide similarity confidence intervals with bootstrap
- Selecting discriminative kmers

Simka

☆ New similarity functions based on **shared kmers**
  ☆ Based on **abundance** and **presence/absence** of species
  ☆ Results close to read-based methods
☆ Fast and low memory thanks to the **GATB library [2]**

References

[1] COMMET: comparing and combining multiple metagenomic datasets
N. Maillet, G. Collet, T. Vannier, D. Lavenier, P. Peterlongo
IEEE BIBM, 2014

[2] GATB: Genome Assembly & Analysis Tool Box
E. Drezen, G. Rizk, R. Chikhi, C. Deltel, C. Lemaitre, P. Peterlongo, D. Lavenier
10.1093/*Bioinformatics*/btu406, 2014
https://gatb.inria.fr/

Funding: ANR Hydrogen, ANR-14-CE23-0001

Publications associées au chapitre 4

BlastGraph : intensive approximate pattern matching in string graphs and de-Bruijn graphs

BlastGraph: Intensive Approximate Pattern Matching in Sequence Graphs and de-Bruijn Graphs

Guillaume Holley[1] and Pierre Peterlongo[1⋆]

Centre de recherche INRIA Rennes - Bretagne Atlantique, IRISA, Campus universitaire de Beaulieu, Rennes, France
guillaumeholley@gmail.com, pierre.peterlongo@inria.fr

Abstract. Many de novo assembly tools have been created these last few years to assemble short reads generated by high throughput sequencing platforms. The core of almost all these assemblers is a sequence graph data structure that links reads together. This motivates our work: BlastGraph, a new algorithm performing intensive approximate string matching between a set of query sequences and a sequence graph. Our approach is similar to blast-like algorithms and additionally presents specificity due to the matching on the graph data structure. Our results show that BlastGraph performances permit its usage on large graphs in reasonable time. We propose a Cytoscape plug-in for visualizing results as well as a command line program. These programs are available at http://alcovna.genouest.org/blastree/.

Keywords: sequence graph, de-Bruijn graph, string matching, high throughput sequencing, next generation sequencing, sequence assembly, Viterbi algorithm

1 Introduction

Compared to traditional Sanger technologies High Throughput Sequencing (HTS) technologies enable sequencing of biological material (DNA and RNA) at much higher throughput and a cost that is now affordable by most academic labs. They have revolutionized the field of genomics and medical research [5]. Sequencing became in a few years accessible to almost all biological labs while being able to produce sequences of full complex genomes in a few days.

HTS technologies do not output the entire sequence of a DNA or RNA molecule. Instead, they return small sequence fragments, called reads, whose length is usually ranging between 100 to 700 characters although some technologies produce longer reads. HTS produce overlapping reads, thus making possible to reconstruct the original sequence by assembling them. Over the last few years, many assemblers were created, such as Euler [2,3], Velvet [12] or Soapnovo [7] to cite a few among the most famous ones. They present different capacities and drawbacks, but all of them make use of a graph data structure storing sequences for organizing the reads. For assembly, the most used graph is the de-Bruijn graph, first proposed for assembly purposes by Pevzner, Tang and Waterman [9]. In a de-Bruijn graph a node represents a length-k substring (called a k-mer) and an edge connects nodes u and v if the two corresponding k-mers overlap over $k-1$ positions. Once the graph is created and usually after an error correction step, a traversal of the graph is performed for generating contiguous sequences called *contigs*.

⋆ Corresponding author

Guillaume Holley, Pierre Peterlongo: BlastGraph: Intensive Approximate Pattern Matching in Sequence Graphs and de-Bruijn Graphs, pp. 53–63.
Proceedings of PSC 2012, Jan Holub and Jan Žďárek (Eds.), ISBN 978-80-01-05095-8

In this paper, we present BlastGraph, a generic approach for aligning a (possibly large) set of query sequences on a graph storing sequences. The algorithm we propose applies on two kinds of graphs: 1/ any kind of graph storing sequences, called Sequence Graphs (SG); 2/ de-Bruijn graphs (DBG). Motivations for this work are multiple. For developers of assembly tools, it is of great interest to precisely detect query sequences in the graph, for instance while testing filter algorithms or correction algorithms. Biologically, checking the presence of approximate copies of a set of sequences in the graph, enables to detect homologies, to filter contaminants and to detect the presence of species. Avoiding the full assembly process presents two main advantages: first it avoids the time consuming contig generation phase, and second and more important, it avoids the usage of heuristics or statistical choices made while traversing the graph.

Note that the BlastGraph algorithm applies generically to any directed SG, and is also adapted to apply to a DBG. Given a directed sequence graph, a set of query sequences and a maximum edit distance, BlastGraph detects paths in the graph on which query sequences align at most at the given edit distance. Our approach is a blast-like algorithm [1]. The graph is indexed using seeds, this enables to decrease the request execution time. The main originality of our work stands in the fact that both seeds and mapped query sequences may be spread over several nodes of the graph.

This work presents similarities with the famous Viterbi algorithm [11]. In a few words, Viterbi is a dynamic programming algorithm for finding the most likely path in a rooted graph while reading a query sequence. The major fundamental differences with this work stand in the fact that:

- Viterbi nodes are composed by a unique symbol while in the BlastGraph framework, nodes store a full sequence, and their reverse complement in the DBG framework;
- In the Viterbi framework, the alignment is global: the full query sequence is aligned to the whole graph, starting from the root node, while in the BlastGraph algorithm, the alignment is semi global: the whole query sequence is aligned to any un-rooted sub-graph.

The next Section introduces preliminaries and definitions. In Section 3 we expose the BlastGraph algorithm when applied on a SG, while in Section 4 we show how BlastGraph is modified to apply on a DBG. We present some practical results in section 5.

2 Preliminaries

A *sequence* is composed by zero or more symbols from an alphabet Σ. A sequence s of length n on Σ is denoted also by $s[0]s[1]\cdots s[n-1]$, where $s[i] \in \Sigma$ for $0 \leq i < n$. The edit distance between two sequences is the minimal number of insertions, deletions and substitutions to transform one into the other. The length of s is denoted by $|s|$. We denote by $s[i,j]$ the *substring* $s[i]s[i+1]\cdots s[j]$ of s. In this case, we say that the substring $s[i,j]$ *occurs* at position i in s. We call *k-mer* a sequence of length k. If $s = u \cdot v$ for u and $v \in \Sigma^*$, we say that v is a *suffix* of s and that u is a *prefix* of s, the symbol "$\cdot$" designating the concatenation between two sequences. Let $s[i..]$ denote the suffix of s starting at position i (*i.e.* $s[i..] = s[i, |s|-1]$).

The symbol "$\overset{\circ}{}_k$" designates the concatenation of two sequences, removing the first k symbols of the second. Formally, $u \overset{\circ}{}_k v = u \cdot v[k..]$. In the DNA context, $\Sigma =$

$\{A, C, G, T\}$, and, given $s \in \Sigma^*$, $\overline{s}$ designates the reverse complement of s, that is s, read from right to left, switching characters A and T, and C and G.

2.1 Sequence Graphs (SG)

In a directed sequence graph $\mathcal{G}$, each node N stores a sequence s, denoted by $S(N)$. A node N_1 linked to a node N_2 denotes the fact that the sequence $S(N_1).S(N_2)$ is stored in $\mathcal{G}$. Example of a directed sequence graph is given Figure 1a.

2.2 De-Bruijn Graphs (DBG)

DBGs were first used in the context of genome assembly in 2001 by Pevzner *et al.* [9]. In 2007, Medvedev *et al.* [8] modified the definition to better model DNA as a double stranded molecule. In this context, given a fixed k value, a DBG is a bi-directed multigraph, each node N storing a k-mer s and its reverse complement $\overline{s}$. The sequence s, denoted by $F(N)$, is the forward sequence of N, while $\overline{s}$, denoted by $R(N)$, is the reverse complement sequence of N. An arc exists from node N_1 to node N_2 if the suffix of length $k-1$ of $F(N_1)$ or $R(N_1)$ overlaps perfectly with the prefix of $F(N_2)$ or $R(N_2)$. Each arc is labelled with a string in $\{FF, RR, FR, RF\}$. The first letter of the arc label indicates which of $F(N_1)$ or $R(N_1)$ overlaps $F(N_2)$ or $R(N_2)$, this latter choice being indicated by the second letter. Because of reverse complements, there is an even number of arcs in the DBG: if there is an arc from N_1 to N_2 then, necessarily, there is an arc from N_2 to N_1 (*e.g.* if the first arc has label FF then the second has label RR).

A DBG can be compressed without loss of information by merging *simple* nodes. A simple node denotes a node linked to at most two other nodes. Two adjacent simple nodes are merged into one by removing the redundant information. A valid path (see Definition 2) composed by $i > 1$ simple nodes is compressed into one node storing a sequence of length $k + (i-1)$ as each node adds one new character to the first node. Figure 1b represents a DBG (upper) and the corresponding compressed DBG (lower). In the remainder of the paper, we denote by cDBG a compressed DBG.

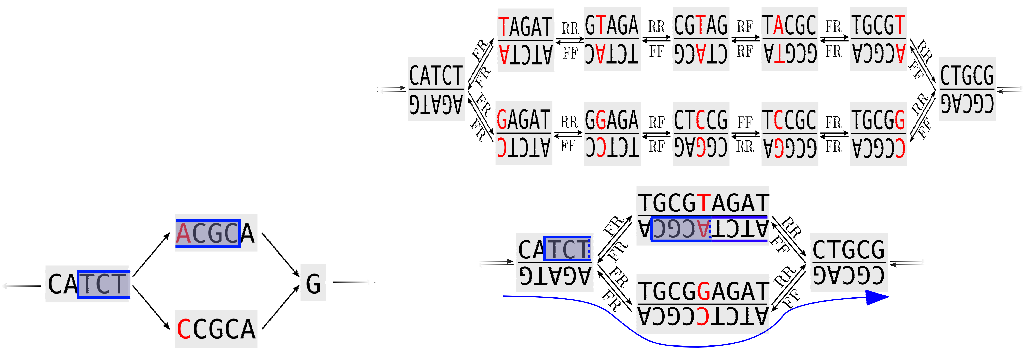
Definition 1 (Active strand of a node in a DBG). *The active strand of a node N in a DBG denotes which strand of the node, forward or reverse, is considered while traversing N.*

Definition 2 (Valid path). *The traversal of a node N is said to be valid if the rightmost label (F or R) of the arc used for entering the node is equal to the leftmost label of the arc used for leaving the node.*

A path in the graph is valid if for each node involved in the path, its traversal is valid, that is, each pair of adjacent arcs in the path are labelled, respectively, XY and YZ with $X, Y, Z \in \{R, F\}$.

Definition 3 (Sequence stored in a cDBG). *A valid path in a cDBG composed by ordered nodes $N_0, N_1, \ldots, N_l$, stores two sequences as following:*

1. *$s = F/R(N_0)_{\mathring{k}}F/R(N_1)_{\mathring{k}} \cdots_{\mathring{k}} F/R(N_l)$, the choice between R or F for node N_0 is equal the first label of the edge going from N_0 to N_1, while for $i \in [1, l]$, the choice between R or F for node N_i is equal the second label of the edge going from N_{i-1} to N_i.*
2. *$\overline{s}$.*



(a) SG (b) DBG

Figure 1: **(a)** Directed sequence graph. **(b)** Uncompressed (upper) and compressed (lower) de-Bruijn Graphs with $k = 5$. For each node, lower sequence is the reverse complement of the upper sequence, it should be read from right to left. **Boxes** both on (a) and (b): example of a seed of length 7 (TCTACGC) spread over 2 nodes. In the de-Bruijn Graph, the $k-1$ first characters of the second node are pruned due to overlap, and the reverse part of the second node is considered as the edge between the first (left) and the second (central) node is F**R**

For instance, the arrowed path on the cDBG presented Figure 1b, stores the sequences

$$\begin{aligned} s &= CATCT_{\hat{k}}ATCTCCGCA_{\hat{k}}CGCAG \\ &= CATCT.CCGCA.G \\ &= CATCTCCGCAG \end{aligned}$$

and

$$\overline{s} = CTGCGGAGATG$$

2.3 Approximate pattern matching in a graph (SG or cDBG)

Definition 4 (Approximate pattern matching in a graph). *Given a query Q, a graph $\mathcal{G}$ (SG or cDBG), and a parameter d, approximate pattern matching consists in finding all occurrences of Q in sequences stored in $\mathcal{G}$ within an edit distance of at most d.*

3 The BlastGraph algorithm

Blast like seed-based heuristics rest on the idea that if two sequences share some similarities, then there exists (at least) a common word (a seed) between these two sequences. Such algorithms consist in, first, anchoring the detection of similarities by exact matching of short sub-sequences, the seeds, and then, performing the similarity distance computation once sequences are anchored. The algorithm we propose applies these ideas between a graph (the bank) and a string (the query). It is divided into four main stages:

1. Index all seeds present in the graph $\mathcal{G}$.

2. Anchor query sequences to nodes of $\mathcal{G}$ using seeds. In the case of genomic data, reverse complement of query sequences may also be used as queries.
3. Align anchored query sequences on the left and right of the matched seeds.
4. Merge left and right alignments.

In the four following sections, we provide some more details for each of these four stages simply considering the graph as a SG. Then, in Section 4, we describe the modifications needed for applying the algorithm on a cDBG.

3.1 Stage 1: Indexing the seeds

Let n denote the length of the seeds. Each word of length n of the sequence of each node of $\mathcal{G}$ as well as those spread over several linked nodes are indexed using a hash table. The index contains for each seed a set of its occurrence positions.

Occurrence position in a graph: An occurrence position in the graph is defined as a couple (node identifier N, position on $S(N)$) indicating the starting position of the occurrence.

Seeds spread over more than one node: Any seed starting at less than n positions to the end of the sequence of a node is spread over more than one node. For instance, the seed $TCTACGC$ starting at position 2 on the leftmost node of Figure 1a, is spread over two distinct nodes. Seeds spread over more than one node are detected thanks to a depth first algorithm recursive approach.

In order to make a light index, the BlastGraph algorithm only stores the starting position of a seed (node identifier N, position on $S(N)$) and not all possible nodes over which the seed is spread.

3.2 Stage 2: Anchoring query sequences to sequences of the graph

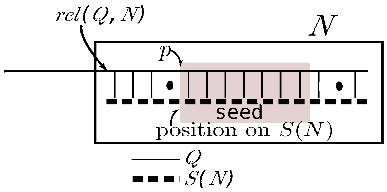


Figure 2: Value $rel(Q, N)$ while anchoring a query sequence Q on a node N with a seed

For each query sequence Q, all overlapping words of length n (seeds) are read. Let s be such a seed occurring at position p on Q, also having at least one occurrence in the graph. Then the index provides a set of couples (node N, position on $S(N)$). For each such couple, the query Q is anchored on the sequence $S(N)$, giving a relative position $rel(Q, N)$ of Q on $S(N)$. More precisely, $rel(Q, N) = p-$position on $S(N)$ (see Figure 2) is the position where Q aligns to $S(N)$. Note that $rel(Q, N)$ could be < 0 if a prefix of $S(N)$ is not aligned to Q. This is the case of $S(Q, L_1)$ in the example presented Figure 3.

Computing an alignment only once: If a node N and a sequence Q share more than one seed for the same alignment, each of them generate the same value $\{rel(Q, N)\}$. As this is a very usual case, in order to avoid computing several times the same alignments, while aligning sequence Q, the value $\{rel(Q, N)\}$ is stored in memory. Thus, the same alignment anchored at position $\{rel(Q, N)\}$ is computed only once.

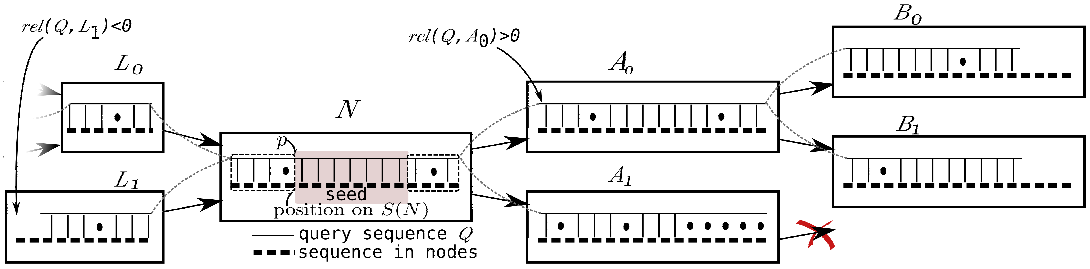


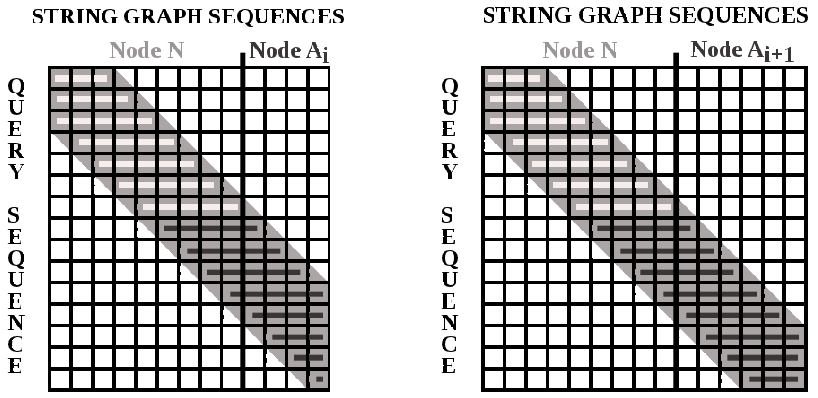
Figure 3: Overview of the alignment process. **Anchoring**: Using a seed, a query sequence Q is anchored to the node N. **Right alignment**: edit distance is computed between $Q[rel(Q, N) + i + k..]$ and $S(N)[i + k..]$ (right dotted square in node N), then between $Q[rel(Q, A_0)..]$ and $S(A_0)$, between $Q[rel(Q, B_0)..]$ and $S(B_0)$, between $Q[rel(Q, B_1)..]$ and $S(B_1)$, and so on. In this example, path using node A_1 presents an edit distance higher than the threshold; its children are not explored. **Left Alignment**: the same procedure is applied on the sequence on the left of the seed (left dotted square in node N), then on parents L_0, L_1 of node N, and so on...

3.3 Stage 3: Alignment between query sequence and sequence graph nodes

Given a query sequence Q anchored at position $\{rel(Q, N)\}$ in a node N of the graph, this stage computes all possible alignments (based on edit distance) between Q and all paths readable from node N (see Figure 3 for an example).

Computing the edit distance between two strings is a dynamic programming procedure that involves the usage of a matrix of size the product of the string lengths. However, in the particular case of this work, the user restricts the maximum edit distance for having a match. Consequently, the matrix computation is limited to a diagonal (see Figure 4 for an example) of width $\left\lfloor \frac{maximum\ edit\ distance}{cost\ indel} \right\rfloor \times 2$. Outside the diagonal, number of insertions or deletions becomes bigger than maximum number of insertions or deletions accepted equal to $\left\lfloor \frac{maximum\ edit\ distance}{cost\ indel} \right\rfloor$. Thus during this stage, the time and memory complexity for aligning query Q to one path of the graph is in $O(|Q|)$ considering *maximum edit distance* and *cost indel* as fixed parameters.

Right alignments The alignment is done between Q and $S(N)$ on the right of the matched seed. Additionally, as shown Figure 3, right extremity of the query sequence may finish after $S(N)$. In such a case the alignment has to be done on children $A_0, A_1, \ldots, A_n$ of node N. On each child A_i, the right extremity of the query sequence may finish after the $S(A_i)$, in this case, alignment continues on its children $B_0, B_1, \ldots, B_{n'}$, and so on. Thus, right part of sequence Q (starting after the anchored seed), may be aligned to $S(N \cdot A_i \cdot B_j \ldots)$. This is done via a recursive depth-first traversal of the graph, starting from N as long as the full right part of S is not aligned. An alignment between the sequence of a node and Q is never computed twice. For instance (Figure 3), if the alignment between Q and $S(N \cdot A_0 \cdot B_0)$ was computed,



(a) (b)

Figure 4: Dynamic programming matrix. Only the shadowed diagonal is computed. **(a)** distance computed between the query sequence S and $S(N.A_i)$. **(b)** distance computed between S and $S(N.A_{i+1})$. Lighter lines are not recomputed for computing matrix (b) if matrix (a) was already computed

the computation between Q and $S(N \cdot A_0 \cdot B_1)$ starts from the last full line of the alignment of Q with $S(N \cdot A_0)$. Thus the alignment between Q and $S(N)$ and $S(A_0)$ is never recomputed.

Left alignments Aligning the part of the sequence Q on the left of the seed to the graph is done using almost the same approach as the one previously described for right alignments. However, there are two main differences: 1) Sequences both from Q and from the nodes are reversed (read from right to left); 2) when the reversed query sequence is longer than the reversed sequence of a node N, the parents $L_0, L_1, ...$ of N are explored in depth first search approach (see Figure 3 for an example).

3.4 Joining left and right alignments

For a given aligned query sequence, each left alignment is compared to each right alignment. For each such couple whose sum of the cost of the alignments is below or equal the user defined maximum edit distance, the full alignment is reported.

4 BLASTGRAPH on compressed de-Bruijn graphs

The three main differences between the SG and the cDBG are:

1. In the cDBG, the sequences of two connected nodes overlaps over $k-1$ characters. Thus, whatever the stage, the concatenation of the sequences of two nodes of the cDBG, has to be done removing the $k-1$ overlapping characters using the "$\overset{\circ}{}_k$" concatenation instead of the classical "$\cdot$" one.
2. In the cDBG, each node N stores a sequence ($F(N)$) and its reverse complement ($R(N)$).
3. Label of edges have to be considered while traversing the graph. Thus, in the cDBG, the general rule is the following: a node N is always traversed either as

forward ($F(N)$) or as reverse complement ($R(N)$), with F or R being its "active strand" (see Definition 1).
In the first case (resp. second case), accessing the children of the node is done following edges starting with the letter F (resp. R).
While following an edge, the active strand of the targeted node F (resp. R) is the second letter of the label of the edge.

Seeding in a DBG: The seeding approach is the same than the one applied on the SG. By convention, all seeds start on forward sequence of each node. This is done without loss of information as each query is considered both in its forward and its reverse complement directions.

Right extension in a DBG: The right extension in a DBG is the same as the one described for a SG. However, the algorithm takes into account some DBG specificities:

- query sequence is mapped on $F(N)$ (seeds are indexed only on forward strands);
- children of a node N are reached using only outgoing edges whose label first character corresponds to the active strand of N, and, once a child is reached, its active strand is the one corresponding to the second character of this label;
- concatenation of sequences of two linked nodes is done pruning the overlapping $k-1$ characters.

Left extension in a DBG: Left extensions in the DBG are done by right extending the reverse complement $\overline{Q}$ of the sequence Q to the DBG, starting from the reverse strand of the node N: $R(N)$.

5 Results

Two prototype versions of this algorithm are implemented. Under the CeCILL License, they can be downloaded here: `http://alcovna.genouest.org/blastree`.
A Java version is implemented in a Cytoscape plug-in. Cytoscape [10] is an open-source platform for visualization and interaction with complex graph, especially in bioinformatics. The second version is implemented in C and can be run under Unix platforms. In the two prototypes, while working on nucleotides, characters are coded in two bits.

The next section proposes a use case of the Java version, while section 5.2 proposes some results over the C prototype.

5.1 Use case

We present in this section a use case, on a toy example. We created a sequence graph containing five nodes (Figure 5). We searched for the sequence

$$ggcgTtcagac/cTatacgcatacgcagcagact/agCctacg,$$

spread over 3 nodes of this graph and containing two mismatches and one insertion. To help the reader, we indicated here substitutions and indel with an upper case letter and we indicated separations between nodes with a '/' character. Of course the practical query sequence is a raw un-annotated sequence. We fixed the cost of a mismatch to 1, the cost of an indel to 2 and the maximum edit distance to 4. BlastGraph (Cytoscape plug-in version)) found the correct path, as presented in Figure 5 where selected nodes are those in which alignment is found between the query and the graph.

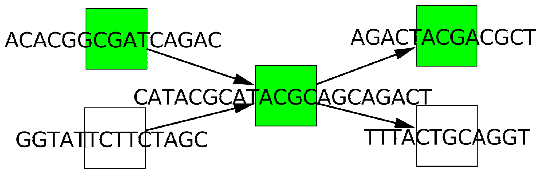


Figure 5: Cytoscape view of the selected nodes (green) in the sequence graph after the research of query sequence

5.2 Performances on DBG

We present results obtained in a typical use case while applying BlastGraph on a DBG graph. Results were obtained on a 64 bit 2×2.5 GHz dual-core computer with 3 MB cache and 4 GB RAM memory. From the NCBI Sequence Read Archive (SRA, `http://www.ncbi.nlm.nih.gov/Traces/sra`), we downloaded the DRR000096 Illumina run containing approximately 4 million reads and approximately 150 million nucleotides.

Increasing graph sizes Subsets of different sizes were generated by randomly sampling DRR000096 reads. For each subset, we constructed the de-Bruijn graph using $k = 31$. Table 1 reports the total number of nodes and nucleotides stored in some of these graphs.

No Reads	No Nodes	No nucleotides
10K	59K	1833K
100K	573K	17774K
150K	849K	26306K

Table 1: Total number of nodes and nucleotides stored in the graph with respect to the number of reads

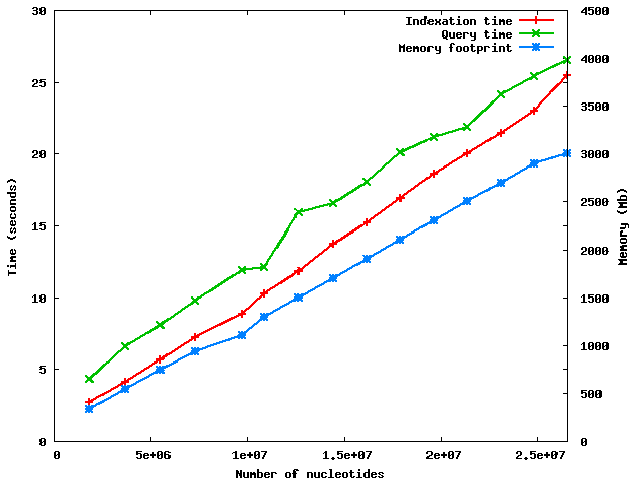


Figure 6: Time and memory consumption with respect to the number of nucleotides stored in the graph

On each graph, we applied the C version of BlastGraph, aligning a set of 10000 query sequences derived from the initial read set. We used seeds of length 19, a mismatch cost equal to 1 and an indel cost equal to 2 and a maximum edit distance equal to 5. We report in Figure 6 time and memory needed both for constructing the index and for performing the 10000 queries.

We can observe that memory footprint and both indexation and query execution times increase linearly with the quantity of information contained in the graph. While memory usage is the main bottleneck of this approach, the indexation and query time are acceptable. Even on the biggest tested graph (containing more than 26 million characters stored in approximately 849000 distinct nodes), indexation is done in 26 seconds and the 10000 queries are performed in less than 52 seconds.

Increasing number of queries In order to measure the impact of the number of queries on the execution time, we used the graph composed of 100000 reads from the DRR000096 data set using $k = 31$. We ran BlastGraph using queries dataset composed of $500, 1000, \ldots, 10000$ reads taken from the 100000 reads used for creating the graph. We report the query time (not including the indexation time) Figure 7. We note that, as expected, the query time increases slowly and linearly with the number of queries.

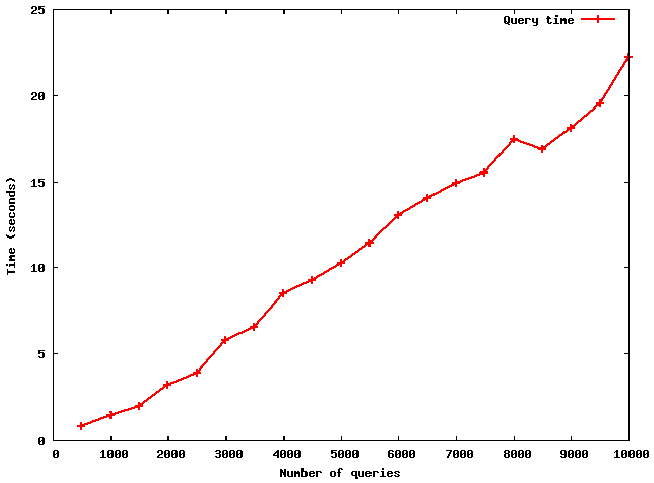


Figure 7: Query time with respect to the number of queries. Note that reported time do not include the indexation time equal to 16 seconds independently of the number of queries

6 Conclusion

We presented BlastGraph, a new algorithm for performing intensive approximate string matching between a set of query sequences and a directed sequence graph including the application to de-Bruijn graphs. This blast-like algorithm presents novelties with respect to "classical" blast-like approaches as seeds and alignments may be spread over several nodes and as the algorithm takes into account double stranded de-Bruijn graph features. Results showed that BlastGraph performances permit its usage on quite large graphs in reasonable time.

The main bottleneck of the approach comes from the memory footprint. Storing in memory graphs containing hundreds of millions of nucleotides together with seed index is challenging. Future work will include either an adaptation of BlastGraph to extremely light DBG representation [4] or a non indexed version of the algorithm, for instance based on KMP [6] algorithm. This will increase the query time, while decreasing the memory usage. Possible applications will exceed the frontiers of the current work as this problem is central in many algorithms associated to high throughput sequencing problems.

7 Acknowledgements

Authors warmly thank Vincent Lacroix and François Coste for their participation to discussions. This work was born from and supported by the Inria "action de recherche collaborative" ARC Alcovna `http://alcovna.genouest.org/`.

References

1. S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman: *Basic local alignment search tool.* Journal of Molecular Biology, 215(3) 1990, pp. 403–410.
2. M. J. Chaisson, D. Brinza, and P. A. Pevzner: *De novo fragment assembly with short mate-paired reads: Does the read length matter?* Genome Research, 19(2) 2009, pp. 336–346.
3. M. J. Chaisson and P. A. Pevzner: *Short read fragment assembly of bacterial genomes.* Genome Research, 18(2) 2008, pp. 324–330.
4. R. Chikhi and G. Rizk: *Space-efficient and exact de Bruijn graph representation based on a Bloom filter*, in WABI, 2012, p. to appear.
5. S. Feature: *Next-generation sequencing transforms today's biology.* Most, 5(1) 2008, pp. 16–18.
6. D. E. Knuth, J. James H. Morris, and V. R. Pratt: *Fast pattern matching in strings.* SIAM Journal on Computing, 6(2) 1977, pp. 323–350.
7. R. Li, H. Zhu, J. Ruan, W. Qian, X. Fang, Z. Shi, Y. Li, S. Li, G. Shan, K. Kristiansen, S. Li, H. Yang, J. Wang, and J. Wang: *De novo assembly of human genomes with massively parallel short read sequencing.* Genome Research, 20(2) 2010, pp. 265–272.
8. P. Medvedev, K. Georgiou, G. Myers, and M. Brudno: *Computability of models for sequence assembly*, in WABI, 2007, pp. 289–301.
9. P. A. Pevzner, H. Tang, and M. S. Waterman: *An Eulerian path approach to DNA fragment assembly.* Proceedings of the National Academy of Sciences of the United States of America, 98(17) Aug. 2001, pp. 9748–53.
10. M. E. Smoot, K. Ono, J. Ruscheinski, P.-L. Wang, and T. Ideker: *Cytoscape 2.8: new features for data integration and network visualization.* Bioinformatics, 27(3) 2011, pp. 431–432.
11. A. Viterbi: *Error bounds for convolutional codes and an asymptotically optimum decoding algorithm.* Information Theory, IEEE Transactions on, 13(2) april 1967, pp. 260 –269.
12. D. R. Zerbino and E. Birney: *Velvet: Algorithms for de novo short read assembly using de Bruijn graphs.* Genome Research, 18(5) 2008, pp. 821–829.

Read Mapping on de Bruijn graph

RESEARCH

Read mapping on De Bruijn graphs

Full list of author information is available at the end of the article

Abstract

Mapping reads on references is a central task in numerous genomic studies. Since references are mainly extracted from assembly graphs, it is of high interest to map efficiently on such structures. In this work we formally define this problem and provide a first theoretical and practical study toward this direction when the graph is a de Bruijn Graph. We show that the problem is NP-Complete and we propose simple and efficient heuristics with a prototype implementation called BGREAT that handle real world instances of the problem with moderate resources. It achieves to map millions reads per CPU hour on a complex human de Bruijn graph. Results show that mapping on graphs enables to map up to 22% more reads than mapping on assembled sequences. BGREAT availability: github.com/Malfoy/BGREAT

Keywords: Read mapping; De bruijn graphs; NGS; NP-Completeness

1 Introduction

Next Generation Sequencing technologies (NGS) have drastically accelerated the generation of sequenced genomes. However these technologies are still not able to provide a unique sequence representing the whole genome. Instead they produce reads, that are short and redundant genomic sequences, substrings from the whole genome. Using the redundancy, it is possible to assemble reads together in order to reconstruct the original reference sequence.

Assembling reads remains nowadays a complex task for which no single piece of software performs consistently well [1]. The assembly problem itself has been shown NP-Hard [2]. Practical limitations arise both from the genome structures (repeats longer than reads cannot be correctly resolved) and from the sequencer technical biases (non-uniform coverage and sequencing errors). Applied solutions organize reads using the de Bruijn graph (DBG) [3] or using the string graph introduced in the Celera Assembler [4]. Then, using heuristics, paths from such graphs are selected and output. Due to heuristics, the produced sequences (the contigs) are biased and fragmented because of various patterns in the graph, generated by genomic repeats, variants, or sequencing errors.

Conversely to assembled contigs, the DBG contains all the data information without taking any uncertain choice. This motivates our choice to consider this kind of graph as a reference of a sequenced genome.

In parallel, a key aspect of bio-analysis is the sequence comparison. Comparing sequences is a fundamental need while searching for similarities or differences between or inside genomes of individuals or species. This is for instance witnessed by the success of the BLAST tool [5], being one of the most cited paper of the scientific community. In the NGS framework, comparisons mainly consist in read mapping,

that is aligning reads on reference genome(s). This is a semi-global alignment, usually extremely stringent (authorizing only a small edit or Hamming distance, limited to $\approx 1\%$ of the read length). A lot of work has been done for providing fast and accurate mapping tools. Notable mapper examples are BWA [6] and Bowtie [7].

In this context, we explore the problem of mapping reads on a graph, and in particular on a DBG, as we aim at considering this data structure as a unbiased representation of the assembly of a genome.

Several tools mention the problem of mapping sequences on graphs. Wang et al [8] proposed to estimate the taxonomic composition of closely-related species by mapping reads on their genomes, all represented by a unique DBG, using the fact that common regions are merged. However this approach still relies on standard aligner and maps reads on a concatenation of DBG nodes. This method takes advantages of the DBG content but does not align on the edges of the graph. Recent works have also been interested in mapping reads on multiple close references in a non-repetitive way [9, 10] using the DBG. Such approaches are particularly efficient but only concern the case of a set of very close sequences that results in very specific flat graphs. Some tools have been proposed [11, 12] for the generic purpose of sequence alignment on graphs of sequence but rapidly become unusable on real world graphs (see Results section).

Interestingly, numerous assembly tools mention the alignment of reads on graph during their assembly process. However, to the best of our knowledge, none of them propose formalism nor practical solution for this problem. To cite a few, Cerulean [13] mentions this step but only use an regular alignment on assembled sequences, Allpaths-LG [14] also proposes such stage, but in a non generic way since it aims at locally resolving repeats by finding paths in accordance with long noisy reads from the third generation sequencing. Those tools often apply the read-threading problem (or eulerian superpath problem [15]) that consists in finding a read coherent path in the DBG (a path that could be represented as a sequence of reads as defined by [16]). The read-threading problem has also been shown NP-Hard [2]. It should not be confused with the generic problem we propose to solve in this work since we are interested in the mapping of arbitrary sequences on an arbitrary DBG. The practical solution we propose could help assemblers in tackling the read-threading problem by mapping reads on their assembly graphs in a generic way. Nowadays, as this is for instance the case in Spades [17], read-threading is performed by keeping in the graph the tracks of the reads used for its construction, which is highly memory consuming.

In this work, we start by showing that the problem of mapping short reads on a DBG is NP-Complete. For efficiency purpose we introduce a heuristic approach for which we provide a prototype implementation that scales actual NGS data set sizes. We propose results showing the advantages of mapping reads on graphs with respect to mapping on linear sequences obtained from assemblies. These results also show the advantages and the limitations of the proposed heuristic approach compared to the brute force algorithm both in term of computational resources and of mapping efficiency.

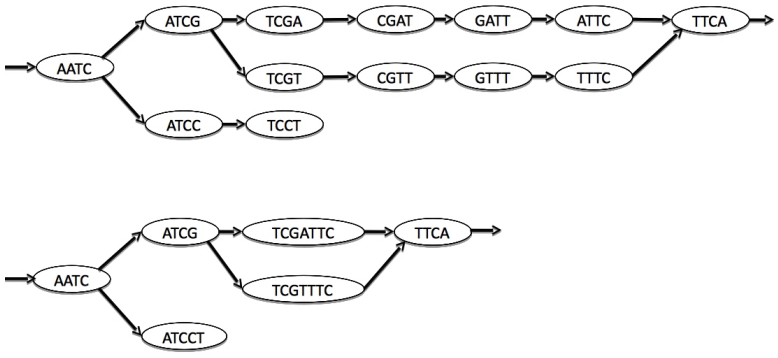


Figure 1 A k-DBG toy example with $k = 4$ (top) and its compacted version (bottom)

2 Read mapping hardness

In this section, we present the formal problem we aim to solve and prove its intractability. We first start by introducing preliminary definitions, before formalizing the problem of mapping reads on graphs and proving its NP-Completeness.

Definition 1 (de Bruijn graph (k-DBG)) *Given a set of strings $S = \{R_1, R_2, ..., R_n\}$ on an alphabet Σ and a value $k \geq 2$, the k order de Bruijn graph (k-DBG) for S is a directed graph (V, E) where:*

$V = \{d \in \Sigma^k \mid \exists\ i \text{ such that } d \text{ is a substring of } R_i \in S\}$

$E = \{(d_i, d_j) : \text{if the suffix of length } k-1 \text{ of } d_i \text{ is the prefix of } d_j\}$

Definition 2 (Path in a k-DBG) *A path in a k-DBG is a walk in this graph with no repeated vertices. A path composed of n nodes produces a sequence of length $k + n - 1$: the k-mer of the first node concatenated with the last character of all remaining ordered nodes of the path.*

Definition 3 (Simple paths) *Given a k-DBG $G = (V, E)$, a simple path is a path of G $v_1, v_2, ..., v_n$ such that $\forall i \in [1, n-1]$*

$(v_i, v_i + 1) \in E$ *and*

$\forall j$ *such that* $(v_j, v_i + 1) \in E$ *implies that* $j = i$ *and*

$\forall j$ *such that* $(v_i, v_j) \in E$ *implies that* $j = i + 1$

An other version of the De Bruijn graph is used, called the compacted De Bruijn graph, where each node contains the sequence of a simple path of maximal length. The sequence of each node is called a "unitig". In this representation, the sequence of a node is of arbitrary length (superior or equal to k) and the edges still represent $k-1$ overlaps (see for instance Fig. 1). Note that, in a compacted DBG, a path composed of n nodes produces a sequence of length $\geq k + n - 1$ as the first node contains k characters or more and as each other node adds one or more character(s). Sequences generated by walking the DBG or the compacted DBG are the same.

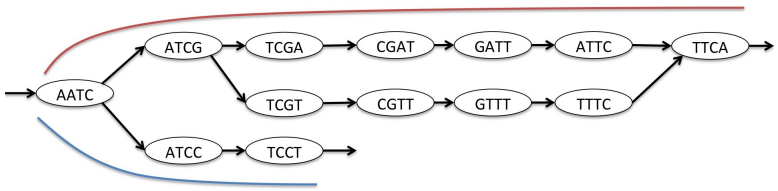


Figure 2 k-DBG toy example with $k = 4$, showing how reads could be mapped. For instance, read $AATACT$ could be mapped with one mismatch following the blue path, while read $AATCGATTCA$ could be mapped with no mismatch following the red path.

Sequence to sequence mapping under Hamming distance is a classical problem. Given a query sequence Q and a reference sequence R, a substitution function $F(\Sigma, \Sigma) \rightarrow \mathbb{N}$, the mapping consists in finding the sub-sequence M of R, such that $|M| = |Q|$, and such that it minimizes the cost $C(M, Q) = \sum_{i=1}^{|Q|} F(M_i, Q_i)$. We recall that, for any $W \in \Sigma^*$, $|W|$ represents the length of W and that W_i designs the i^{th} character of W. The substitution function can be trivial (0 for matches, 1 for mismatches) or can be more complex for biological meanings.

Here we extend the mapping problem with the paths of a DBG instead of a reference sequence. In this case, we search for the path of the graph (represented as a sequence) containing a substring M, with $|M| = |Q|$, minimizing the cost $C(M, Q)$. The decision version of the problem is to decide whether a substring M exists such that $C(M, Q)$ is lower than a specified threshold.

In practice, the query may be a NGS read and the DBG is constructed over NGS read set(s). An example of reads mapped on a DBG is presented Fig. 2. We call this problem DBGRMP for "De Bruijn Graph Read Mapping Problem" and we describe it in a formal way Definition 4.

Definition 4 (De Bruijn Graph Read Mapping Problem) *We define the de Bruijn Graph Read Mapping Problem DBGRMP(G, Q, F, T) as follows :*

Given :

- *G, a k-DBG on Σ*
- *Q a word in Σ^* with $|Q| \geq k$*
- *a (substitution) cost function $F(\Sigma, \Sigma) \rightarrow \mathbb{N}$*
- *a threshold $T \in \mathbb{N}$*

An instance of DBGRMP consists in deciding whether there exists a path of G composed of $|Q| - k + 1$ nodes (producing a word $M \in \Sigma^{|Q|}$) whose cost $C = \sum_{i=1}^{|W|} F(M_i, Q_i)$ is lower than T.

DBGRMP is in NP since a polynomial non-deterministic algorithm exists to solve this problem (pick in a non-deterministic way a path and check if the score is correct). We show that DBGRMP is NP-Hard (and then NP-Complete) by reducing the Hamiltonian path problem to it using three straightforward reductions.

Definition 5 *We define the Hamiltonian Path Problem HPP(G) as follows : Given a directed graph G, an instance of HPP(G) consists in deciding whether there exists a path that passes through all the vertices of G exactly once. Such a path is called a Hamiltonian path.*

Theorem 1 *HPP is NP-Complete [18].*

Definition 6 *We define the fixed length Travelling Salesman Problem L-TSP(G, T) as follows :*

Given :

- *a directed graph G whose edges are labelled with a non-negative integer cost.*
- *a threshold* $T \in \mathbb{N}$
- *an integer L*

An instance of L-TSP(G, T) consists in deciding whether there exists a path (a walk without repeated vertices) of G composed of L nodes whose cost C is inferior to T.

Proposition 1 *L-TSP is NP-Hard.*

Proof We can reduce HPP to L-TSP. Given an instance of HPP (G), we solve L-TSP (G', T) with T equal to the number of nodes of G minus 1 and with G' the same graph as G but with a cost of 1 labelled on each edges. □

Definition 7 (Graph Read Mapping Problem) *We define the Read Graph Mapping Problem GRMP(G, Q, F, T) as follows :*

Given :

- *a directed graph G whose edges are labelled over* Σ
- Q *a word in* Σ^*
- *a (substitution) cost function* $F(\Sigma, \Sigma) \to \mathbb{N}$
- *a threshold* $T \in \mathbb{N}$

An instance of GRMP(G, Q, F, T) consists in deciding whether there exists a path of G composed of $|Q|$ *nodes, thus generating a word* $M \in \Sigma^{|Q|}$*, such that* $C = \sum_{i=1}^{|W|} F(M_i, Q_i)$ *is lower than T.*

Proposition 2 *GRMP is NP-Hard.*

Proof We can reduce L-TSP to GRMP. Given an instance of L-TSP (G, T), we solve GRMP(G', α^L, F, T), where G' is the same graph as G with each letter M_i associated with each edge of cost i for all i, and F such that $F(\alpha, M_i) = i$. Here $\Sigma = \{\alpha\} \cup \{M_i \mid i \text{ cost of an edge of } G\}$ with α any symbol different from all symbols M_i. □

Proposition 3 *DBGRMP is NP-Hard.*

Proof We can reduce GRMP to DBGRMP. Given an instance of GRMP (G, Q, F, T), we solve DBGRMP (G', Q, F', T) where G' is a 2-DBG created from G as follows. Each node of G is labeled with a unique integer. Each quadruplet i, α, j, β, with i and j two nodes from G having an edge from i to j labeled with α and such that β is the label of an output edge from j, generates a node in G' containing the 2-mer $\alpha_i\beta_j$.

Note that G' is defined on an alphabet Σ', where each $\gamma \in \Sigma$ is transformed to γ_i in Σ', with $i \in \mathbb{N}$. We can define a function $p(\Sigma') \to \Sigma$ as $p(\gamma_i) = \gamma$ and a function

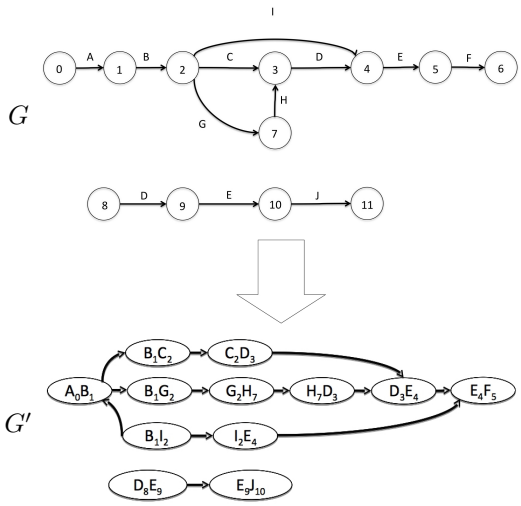


Figure 3 Encoding of a directed graph to a 2DBG. The directed graph G (top) admits the same words than the 2-DBG G' (bottom), if we ignore the numbers.

$P((\Sigma')^*) \to \Sigma^*$ as $P(Q') = Q$, with Q the ordered concatenation of $t(\sigma)$ for all σ in Q'. We then define F' such that $\forall M_i \in \Sigma', y \in \Sigma, F'(y, M_i) = F(y, p(M_i))$ (the size of the representation of F' can be linearly bounded by the size of the representation of F).

The graph G' is a 2-DBG that produces the same words as G (except the 1-letter words) if we apply the linear function P to words read from G'. An example of this reduction is shown Fig. 3, the word $ABCDEF$ of the directed graph is equivalent to the word $A_0B_1C_2D_3E_4F_5$ of the DBG for the scoring function. The graph G' contains at most $card(G)^3$ nodes in the worst case where G is a complete directed graph (a graph in which any two vertices are connected in both directions). In this case, each node in G has $card(G)$ father(s) and $card(G)$ son(s) thus creates $card(G)^2$ node(s) to the DBG. The reduction is therefore polynomial. □

3 Methods

We propose a practical solution for solving the DBGRMP. This section motivates and presents the algorithmic choices we made. Our proposed solution had been designed for mapping on a compacted de Bruijn graph (CDBG) any source of short reads, either those used to build the graph or reads from another individual or species. We focused our work on a model of short (hundred of base pairs) reads with a low error rate (1% of substitution), which is a good approximation of currently widely used NGS reads. Since errors are mostly substitutions, mapping is computed using the Hamming distance.

In a CDBG, nodes are no longer k-mers (words of length k) but *unitigs* that are the sequences obtained after the compaction of simple paths of the DBG, some of the unitigs being longer than reads. Thus, while mapping on a CDBG, one may distinguish between two mapping schemes: **i/** reads mapping completely on a unitig of the graph, and **ii/** reads whose mapping spans two or more unitigs. Given the extensive work carried out these last few years for mapping reads on flat strings,

Figure 4 Representation of the mapping of a read (highest represented sequence) on a CDBG whose nodes are represented lines 2, 3, and 4. **(step 1)** overlaps of the graph also present in the read are found (here $TACAC$, $GCTGC$, and $AGCTA$, represented line 1). **(step 2)** unitigs that map the begin and the end of the read are found (those represented line 2). **(step 3)** cover the rest of the read, guided by the overlaps (here with unitigs represented line 3 and 4).

we concentrate here only on the second scheme. We say that reads mapping on two or more unitigs map on a *branching* part of the graph.

As presented in the previous section, the DBGRMP is NP-Complete. An exhaustive search of all possible mapping positions of a read on a DBG would require to compare each read to a potentially intractable number paths in the graph. Thus we propose a heuristic approach for solving DBGRMP using reasonable resources both in terms of time and memory. Our approach follows the usual "seed and extend" paradigm. An exact match is used as a starting point and it is then extended by following some paths of the graph. More precisely, the proposed implementation applies heuristic schemes, both regarding the indexing and the alignment phases.

3.1 Indexing heuristic

We recall that our algorithm maps reads that span at least two distinct unitigs. Such mapped reads inevitably traverse one or more DBG edge(s). In a CDBG, edges are represented by the prefix and suffix of size $k-1$ of each unitig. We call such sequences the *overlaps*. In order to limit the index size and the computation time, our algorithm indexes only overlaps, that are used as seeds. Those overlaps are good anchors for several reasons: they are long enough ($k-1$) to be selective, they cannot be shared by more than eight unitigs (four starting and four ending with the overlap), and a CDBG usually has a reasonable number of unitigs and then of overlaps (our human experiment CDBG has 70 millions unitigs and 87 millions overlaps for 3 billions kmers, see Table 1). In our proposed implementation, the index is a hash table indicating for each overlap the unitig(s) starting or ending with this $k-1$ sequence.

3.2 Read alignment

As presented Fig. 4, given a read, each of its $k-1$-mers are read in order to detect those that are an overlap of the CDBG. Once such overlaps are detected together with their corresponding unitigs, the alignment of the read is performed from left to right, as presented in the Algorithm Fig. 5. Given an overlap position i on the read, the unitigs starting with this overlap are aligned to the sequence of the read starting at position i. The best alignment is conserved. In addition, for improving speed, if one of the at most four unitigs ending overlap is the next overlap detected on the read, then this unitig is tested first, and if the alignment contains less mismatch than the authorized threshold (user defined), the other (at most three) unitigs are not tested. Note that this optimization does not apply for the first and last overlaps of the read.

```
Algorithm: MapGreedy

Data: Read r, Integer n
for the n first overlaps of r do
    Find a path begin that map the begin of r
    if begin found then
        for the n last overlaps of r do
            Find a path end that map the end of r
            if end found then
                Find (in a greedy way) a path cover that map the read from
                begin to end
                if cover found then
                    write paths
                    return
```

Figure 5 Greedy algorithm for mapping a read on two or more unitigs once potential overlaps present in the read are detected.

The previously described full read mapping process is tested only if the two extremities of the read are mapped by two unitigs. The extreme overlaps of the read enable to quickly filter out unmappable reads. For doing this, the first (resp. last) overlap of the read is used to align the read to the first (resp. last) unitig. Note that, as polymorphism may exists between the read and the graph, some of the overlaps present on the read may be spurious. In this case the alignment fails and the algorithm continues with the next (resp. previous) overlap. At most n alignment failures are authorized left and right. If a read cannot be anchored both left and right using this process, then it is not aligned to the graph.

Note that the whole approach is greedy: given two or more possible choices, the best one is chosen with no possible backtracking. This enables a linear mapping process since each position in the read can lead to a maximum of four comparison and the algorithm continues as long as the accumulated number of mismatches is below a user defined parameter.

Because of heuristics, a read may be unmapped or badly mapped for any of the following reasons.

- All overlaps on which the read should map contain errors, in this case the read is not anchored or only badly anchored and thus not mapped.
- The n first or n last overlaps of the read are spurious, in this case the the *begin* or *end* is not found and the read is not mapped. By default and in all experiments $n = 2$.
- The greedy choices made during the path selection are wrong.

4 Results

This section is divided into two main contributions. The first one presents the prototype we developed in order to test the proposed approach, while the second one presents results obtained applying our methods on real data sets.

We implemented the previously described algorithm in a prototype tool called *BGREAT*. It was written in C++ and can be downloaded from github.com/Malfoy/BGREAT. *BGREAT* maps reads on branching parts of the DBG.

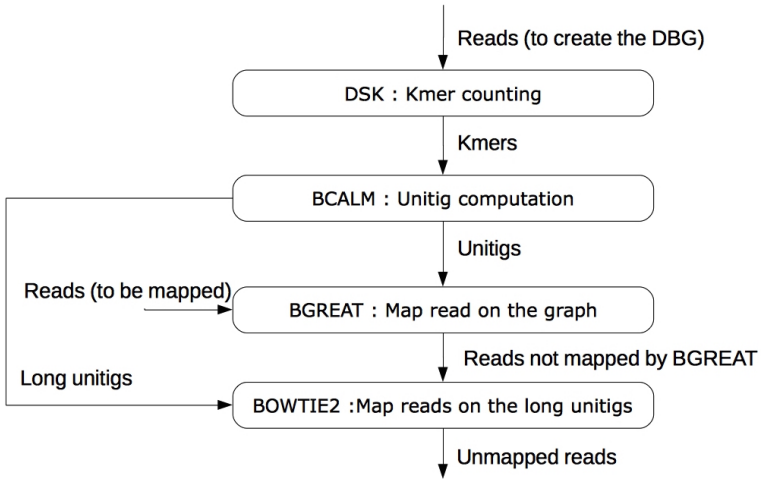


Figure 6 Proposed pipeline. Reads to be mapped can be different from reads used for building the graph. Long unitigs are unitigs longer than reads

dBG Id	Reads Id	k	c	Number of unitigs	Mean length of unitigs	Number of overlaps
E.coli	SRR959239	31	3	42,843	134	62,615
*C.elegans*_cpx	SRR065390	21	2	8,273,338	34	10,856,948
*C.elegans*_norm	SRR065390	31	3	1,627,335	93	2,212,883
Human	SRR345593 & SRR345594	31	10	69,932,343	70	86,948,988

Table 1 CDBG used in this study. *C.elegans*_cpx and *C.elegans*_norm are two distinct graphs, constructed using the same read set from *C.elegans* genome. The suffix *cpx* and *norm* respectively stand for "complex" (using a low threshold $c = 2$ and small value $k = 21$, and for "normal" (using $c = 3$ and $k = 31$).

BGREAT takes as inputs a query read set and a reference DBG. For removing sequencing errors from DBG, in the presented experiments, the reference graphs were constructed after removing k-mers having a coverage below a threshold c. This error removal step is a classical data treatment, performed in k-mer based assemblers. For this purpose, k-mers are counted using *DSK* [19]. The unitigs of the CDBG are computed using *BCALM* [20] on the k-mers having a coverage $\geq c$. *BGREAT* uses such a set of unitigs as DBG.

In addition, in the proposed experiments, a third-party mapper (Bowtie2 [7]) is used in order to map reads that are fully contained in DBG unitigs.

The full pipeline applied in the following experiments is represented Fig. 6.

Reference CDBG were constructed using three real read data sets of growing size and complexity. They are composed of Illumina reads of length ≈ 100 from whole genome sequencing data sets. The three read sets are respectively a sequencing of *E.coli* (SRR959239) at 110x coverage, a sequencing of *C.elegans* (SRR065390) at 70x coverage and a sequencing of human individual (SRR345593 and SRR345594) at 112x coverage.

We tested *BGREAT* on each of the DBG generated from these data sets. From the *C.elegans* read set, we tested *BGREAT* on two DBG of distinct size and complexity.

For this purpose, we varied the size of the k-mers and the threshold c. As depicted Table 1, with small k and c values, the graph contains more and smaller unitigs.

All *BGREAT* alignments were performed authorizing up to two mismatches. The queries were either composed of the same reads as those used to build the reference graph or composed of read sets from other individual of the same species.

Graph	Mapped set (nb reads)	Mapping Strategy	Nb reads mapped (on branching parts of DBG)	Nb reads fully mapped on unitigs	Overall nb mapped reads
E.coli	SRR959239 (5,128,790)	Greedy	687,074 (13.40%)	4,296,710 (83.78%)	4,983,784 (97.17%)
〃	〃	Complete	688,929 (13.43%)	4,295,814 (83.76%)	4,984,743 (97.19%)
*C.elegans*_cpx	SRR065390 (67,617,092)	Greedy	44,686,355 (66.09%)	15,592,918 (23.06%)	60,279,273 (89.15%)
*C.elegans*_norm	〃	Greedy	13,994,715 (20.70%)	48,442,146 (71.64%)	62,436,861 (92.34%)
*C.elegans*_norm	SRR1522085 (22,509,110)	Greedy	3,523,416 (15.65%)	16,682,194 (74.11%)	20,205,610 (89.77)%
Human	SRR345593 and SRR345594 (2,967,536,821)	Greedy	1,004,182,363 (33.84%)	1,533,456,046 (51.67%)	2,537,638,409 (85.51%)

Table 2 Results of *BGREAT* on real read sets. The '〃' symbol indicates same value(s) between two consecutive lines. We recall that the column "*Nb reads fully mapped on unitigs*" corresponds to the number of reads mapped by Bowtie2 [7] with defaults parameters.

Graph	Mapped set (nb reads)	Mapping Strategy	Time	Reads by second	Memory
E.coli	SRR959239 (5,128,790)	Greedy	2m2	42,039	15 MB
〃	〃	Complete	1h24	1,017	〃
*C.elegans*_cpx	SRR065390 (67,617,092)	Greedy	3h08	5,965	1.16GB
*C.elegans*_norm	〃	Greedy	1h55	9,438	380MB
*C.elegans*_norm	SRR1522085 22,509,110	Greedy	12m25	30,213	380M
Human	SRR345593 and SRR345594 (2,967,536,821)	Greedy	11h48	70,526	18 GB

Table 3 Time and memory footprints of *BGREAT*. Results do not present Bowtie2 time and memory footprints. Indicated times are wallclock time using one thread, except for the human samples for which 12 cores were used.

Table 2 presents results while applying *BGREAT* in association with Bowtie2. This table is completed by Table 3 in which *BGREAT* time and memory footprints are presented for the same experiments. All these results enable to draw the following main conclusions:

- The idea of mapping reads on branching parts of the graph cannot be substituted by simply mapping reads on unitigs. Indeed, at least 13.40% of reads

(mapping reads SRR959239 on the *E.coli* DBG) and up to 66.09% of reads (mapping reads SRR065390 on *C.elegans*_cpx DBG) map the graph through mapping over branching parts. These reads would no be mapped if one would use only the set of unitigs as a reference.
As expected, the more complex the graph is, the higher the usefulness of the *BGREAT* approach. On the complex *C.elegans*_cpx graph, only 23.06% of reads can be fully mapped on unitigs while 89.15% of them are mapped by additionally using *BGREAT*. On a simpler graph as *C.elegans*_norm the gap is smaller, but remains significant (from 71.64% to 92.34%).

- Results on the *E.coli* data set show that the heuristics we propose for mapping reads on the DBG are useful. In order to measure the impact of the read alignment heuristics, we can force the prototype to explores exhaustively all potential alignment paths once a read is anchored to the graph (this strategy is referred as *Complete* in Table 2)
  Results show that the greedy approach is much faster than the exhaustive one (38x faster on this experiment) while the mapping quality is poorly impacted by heuristic approach: the overall number of mapped reads is increased by only 0.03% with the *Complete* approach.
- The approach can be applied also for mapping a distinct read set than the one used for constructing the DBG. We mapped a read set (SRR1522085) on the *C.elegans*_norm CDBG, this read set (from *C.elegans*) being different from the one used for constructing the graph. Results in this situation are similar to those observed while mapping to a DBG the read set used to construct this graph (an overal of 89.77% of mapped reads while 15.65% of them are mapped on branching parts of the graph).
- The proposed prototype scales real-world instances of the problem. It achieves to map millions reads per CPU hour on a human DBG. Note that the memory footprint could be highly reduced by using a lightweight data structure such as a FM-index [7] or a sparse hash table, instead of a classical hash table.

We are aware of only one another published tool, called *BlastGraph* [12] designed for the mapping reads on DBG. However, on the simplest tested data set from *E.coli* genome (see Table 1), *BlastGraph* crashed after $\approx$ 124h of computation. This tool is thus not further investigated here. In addition, one may mention the SSW library [21], initially designed for mapping on strings. It has been extended for mapping on graphs (library GSSW [11]). To the best of our knowledge, no implemented tool using this library is available and its documentation indicates that reference and query size do not scale-up current NGS instances.

We may compare *BGREAT* to the popular approach consisting in mapping the reads to the reference contigs generated by an assembly process. For testing this approach, for each of the three sets used, we first assembled them and then we mapped back these reads on the so created assemblies. We used two different assemblers, Velvet [22] a widely used tool and Minia [23] a resource efficient assembler based on bloom filters. Finally, we used Bowtie2 for aligning the reads on the obtained contigs.

				Mapping on assembly	Mapping on CDBG
Reads set	Assembler	k	c	Mapping rate	Mapping rate
E.coli	Velvet	31	3	95.57%	97.17%
〃	Minia	〃	〃	96.19%	〃
C.elegans	Velvet	21	2	56.33%	89.15%
〃	Minia	〃	〃	51.43%	〃
〃	Velvet	31	3	80.60%	92.34%
〃	Minia	〃	〃	80.43%	〃
Human	Minia	31	10	63.16%	85.51%

Table 4 Assembly and mapping approach. Columns k and c are the assembly main parameters for Velvet and Minia. The last column recalls the overall rate of reads mapped using $BGREAT$ + Bowtie2 on the CDBG obtained with the same read sets and k and c parameters. Because of limited computational resources, Velvet was not tested on the Human data-set.

Results of this approach are presented Table 4. They show that Velvet and Minia obtained similar results in term of quantity of mapped read on their assembly. However, the number of mapped reads on these assemblies is lower than the one obtained by our proposed approach as presented Fig. 6.

We notice that the more complex the graph is, the higher the advantage of mapping on CDBG because of the inherent difficulty to assemble huge and highly branching graphs. We also highlight that our approach is resource efficient compared to most assembly processes. For example, Velvet used more than 80GB memory to produce the contigs for the *C.elegans* data set with k=31. On this data set, our workflow used at most 4GB memory (during k-mer counting). In term of throughput, using *BGREAT* and then Bowtie 2 on long unitigs is comparable to using Bowtie 2 on contigs. For instance, on the *C.elegans* data set, both methods used slightly less than 3 CPU hours.

Finally, an important message here is that mapping on assembled sequence could be highly improved by mapping on graph structures, on which more reads may be mapped (up to ≈22% here for the human data set and ≈ 38% for the artificially complex *C.elegans* data set).

5 Discussion

We proposed a formal definition the of de Bruijn graph Read Mapping Problem (DBGRMP) and we proved its NP-Completeness. We proposed a heuristic algorithm offering a practical solution. As a proof of concept, we developed a prototype called *BGREAT* implementing this algorithm using a compacted de Bruijn graph (CDBG) as a reference. Experiments show that many reads (between ≈13% and ≈66% of them depending on the experiment) can be mapped only on branching parts parts of the graph. The idea of mapping only on nodes is thus insufficient. This is true either while mapping reads used for constructing the graph or while mapping reads from other experiment. Moreover, results show that a potential high number of reads (up to ≈ 38%) that are mapped on CDBG could not be mapped on classical assemblies.

A weak point of our algorithm lies in its anchoring technique. Reads mapped with *BGREAT* must contain at least a $k-1$-mer being an edge of the CDBG, ie an overlap between two connected nodes. This may be a serious limitation in case the data used for constructing the graph highly diverge from the reads to be mapped. Improving the mapping technique may be done by using more than only unitig

overlaps as anchors at the cost of higher computational resources. Another solution may consist in using a smarter anchoring approach for instance using seeds allowing errors as proposed in [24].

A natural future work will consist in adapting *BGREAT* for mapping reads produced by Third Generation Sequencers (TGS) providing much longer reads (a few kb in average) with a larger error rate (up to 15%, being mainly insertions and deletions for Pacific Biosciences reads) on CDBG graphs obtained from NGS short reads. This approach would open the doors to TGS read assembly and/or correction [25]. Such adaptation is not straightforward because of our "seed strategy" that requires long exact matches. The anchoring process must be highly sensible and very specific while the mapping itself must implements Blast-like classical edit distance heuristics or even alignment-free methods. However, mapping such long read on a DBG could be of high interest for repeat resolutions if long read map on cycle of the DBG. Our NP-Completeness proof only consider mapping on (acyclic) paths. Proving the hardness of the problem of mapping reads on a DBG with cycles is an open problem.

By the way, using the same read set for constructing the CDBG and for mapping, opens the way to straightforward major applications. Indeed, the graph and the exact location of each read on it may be used for **i/** read correction, by detecting differences between read and the mapped area of the graph in which low support k-mers likely due to sequencing errors are absent as in [26], or **ii/** read compression by conserving additionally mapping errors, or **iii/** both correction and compression conserving nothing else.

Having for each read (used for constructing the graph or not) its location on the CDBG provides the opportunity to design algorithms for enriching the graph, for instance enabling a fine quantification sensible to local variations. This would be valuable for instance for applications such as variant calling and analysis of RNA-seq [27] or metagenomic data [28].

Additionally, *BGREAT* results provide pieces of information for distant k-mers in the CDBG, about their co-occurrences in the mapped read data sets. This offers a way for the resolution, in the de Bruijn graph, of repeats larger than k. It could also allow to phase the polymorphisms and to reconstruct haplotypes.

Competing interests
The authors declare that they have no competing interests.

Author's contributions
Anonymous

Acknowledgements
Anonymous

References

1. Bradnam, K.R., Fass, J.N., *et al.*: Assemblathon 2: evaluating de novo methods of genome assembly in three vertebrate species. GigaScience **2**, 10 (2013). doi:10.1186/2047-217X-2-10. 1301.5406
2. Nagarajan, N., Pop, M.: Parametric Complexity of Sequence Assembly: Theory and Applications to Next Generation Sequencing. Journal of Computational Biology **16**(7), 897–908 (2009). doi:10.1089/cmb.2009.0005
3. Chaisson, M.J., Pevzner, P.A.: Short read fragment assembly of bacterial genomes. Genome Research **18**(2), 324–330 (2008). doi:10.1101/gr.7088808
4. Myers, E.W., Sutton, G.G., *et al.*: A whole-genome assembly of Drosophila. Science (New York, N.Y.) **287**(5461), 2196–2204 (2000). doi:10.1126/science.287.5461.2196
5. Altschul, S.F., Gish, W., *et al.*: Basic local alignment search tool. Journal of Molecular Biology **215**(3), 403–410 (1990). doi:10.1016/S0022-2836(05)80360-2

6. Li, H., Durbin, R.: Fast and accurate short read alignment with Burrows-Wheeler transform. Bioinformatics **25**(14), 1754–1760 (2009). doi:10.1093/bioinformatics/btp324
7. Langmead, B., Salzberg, S.L.: Fast gapped-read alignment with Bowtie 2. Nature Methods **9**(4), 357–359 (2012). doi:10.1038/nmeth.1923. #14603
8. Wang, M., Ye, Y., Tang, H.: A de Bruijn Graph Approach to the Quantification of Closely-Related Genomes in a Microbial Community. Journal of Computational Biology **19**(6), 814–825 (2012). doi:10.1089/cmb.2012.0058
9. Huang, L., Popic, V., Batzoglou, S.: Short read alignment with populations of genomes. Bioinformatics **29**(13), 361–370 (2013). doi:10.1093/bioinformatics/btt215
10. Dilthey, A., Cox, C., Iqbal, Z., Nelson, M.R., McVean, G.: Improved genome inference in the MHC using a population reference graph. Nature Genetics **47**(6), 682–688 (2015). doi:10.1038/ng.3257
11. GSSW web site. http://github.com/ekg/gssw. Accessed: 2015-04-27
12. Holley, G., Peterlongo, P.: BlastGraph : intensive approximate pattern matching in string graphs and de-Bruijn graphs. In: PSC 2012, vol. 2012, pp. 1–13 (2012)
13. Deshpande, V., Fung, E.D.K., Pham, S., Bafna, V.: Cerulean: A Hybrid Assembly Using High Throughput Short and Long Reads. In: Lecture Notes in Computer Science vol. 8126 LNBI, pp. 349–363 (2013). doi:10.1007/978-3-642-40453-5_27
14. Ribeiro, F.J., Przybylski, D., Yin, S., Sharpe, T., Gnerre, S., Abouelleil, A., Berlin, A.M., Montmayeur, A., Shea, T.P., Walker, B.J., Young, S.K., Russ, C., Nusbaum, C., MacCallum, I., Jaffe, D.B.: Finished bacterial genomes from shotgun sequence data. Genome Research **22**(11), 2270–2277 (2012). doi:10.1101/gr.141515.112
15. Pevzner, P.A., Tang, H., Waterman, M.S.: An Eulerian path approach to DNA fragment assembly. Proceedings of the National Academy of Sciences **98**(17), 9748–9753 (2001). doi:10.1073/pnas.171285098
16. Myers, E.W.: The fragment assembly string graph. Bioinformatics **21**(Suppl 2), 79–85 (2005). doi:10.1093/bioinformatics/bti1114
17. Bankevich, A., Nurk, S., Antipov, D., Gurevich, A.A., Dvorkin, M., Kulikov, A.S., Lesin, V.M., Nikolenko, S.I., Pham, S., Prjibelski, A.D., Pyshkin, A.V., Sirotkin, A.V., Vyahhi, N., Tesler, G., Alekseyev, M.A., Pevzner, P.A.: SPAdes: A New Genome Assembly Algorithm and Its Applications to Single-Cell Sequencing. Journal of Computational Biology **19**(5), 455–477 (2012). doi:10.1089/cmb.2012.0021
18. Karp, R.M.: Reducibility Among Combinatorial Problems. In: 50 Years of Integer Programming 1958-2008, pp. 219–241. Springer, Berlin, Heidelberg (2010). doi:10.1007/978-3-540-68279-0_8. http://link.springer.com/10.1007/978-3-540-68279-0_8
19. Rizk, G., Lavenier, D., Chikhi, R.: DSK: K-mer counting with very low memory usage. Bioinformatics **29**(5), 652–653 (2013). doi:10.1093/bioinformatics/btt020
20. Chikhi, R., Limasset, A., *et al.*: On the representation of de bruijn graphs. In: Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 8394 LNBI, pp. 35–55 (2014). doi:10.1007/978-3-319-05269-4-4
21. Zhao, M., Lee, W.-P., Garrison, E.P., Marth, G.T.: SSW Library: An SIMD Smith-Waterman C/C++ Library for Use in Genomic Applications. PLoS ONE **8**(12), 82138 (2013). doi:10.1371/journal.pone.0082138
22. Zerbino, D.R., Birney, E.: Velvet: Algorithms for de novo short read assembly using de Bruijn graphs. Genome Research **18**(5), 821–829 (2008). doi:10.1101/gr.074492.107
23. Chikhi, R., Rizk, G.: Space-efficient and exact de Bruijn graph representation based on a Bloom filter. Algorithms for Molecular Biology **8**(1), 22 (2013). doi:10.1186/1748-7188-8-22
24. Vroland, C., Salson, M., Touzet, H.: Lossless Seeds for Searching Short Patterns with High Error Rates. In: Combinatorial Algorithms - 25th International Workshop, IWOCA 2014, Duluth, MN, USA, October 15-17, 2014, Revised Selected Papers, pp. 364–375 (2014)
25. Salmela, L., Rivals, E.: LoRDEC: accurate and efficient long read error correction. Bioinformatics **30**(24), 3506–3514 (2014). doi:10.1093/bioinformatics/btu538
26. Benoit, G., Lavenier, D., Lemaitre, C., Rizk, G.: Bloocoo, a memory efficient read corrector. In: European Conference on Computational Biology (ECCB) (2014)
27. Sacomoto, G.A., Kielbassa, J., Chikhi, R., Uricaru, R., Antoniou, P., Sagot, M.-F., Peterlongo, P., Lacroix, V.: Kissplice: de-novo calling alternative splicing events from rna-seq data. BMC bioinformatics **13**(Suppl 6), 5 (2012)
28. Ye, Y., Tang, H.: Utilizing de bruijn graph of metagenome assembly for metatranscriptome analysis. arXiv preprint arXiv:1504.01304 (2015)