Mapping reads on a genomic sequence: an algorithmic overview and a practical comparative analysis

Sophie S. Schbath, Veronique V. Martin, Matthias Zytnicki, Julien Fayolle,
Valentin Loux, Jean-François Gibrat

HAL Id: hal-02645697

https://hal.inrae.fr/hal-02645697

Submitted on 29 May 2020

Mapping Reads on a Genomic Sequence: An Algorithmic Overview and a Practical Comparative Analysis

*SOPHIE SCHBATH,[1] *VÉRONIQUE MARTIN,[1] MATTHIAS ZYTNICKI,[2] JULIEN FAYOLLE,[1]
VALENTIN LOUX,[1] and JEAN-FRANÇOIS GIBRAT [1]

ABSTRACT

Mapping short reads against a reference genome is classically the first step of many next-generation sequencing data analyses, and it should be as accurate as possible. Because of the large number of reads to handle, numerous sophisticated algorithms have been developed in the last 3 years to tackle this problem. In this article, we first review the underlying algorithms used in most of the existing mapping tools, and then we compare the performance of nine of these tools on a well controled benchmark built for this purpose. We built a set of reads that exist in single or multiple copies in a reference genome and for which there is no mismatch, and a set of reads with three mismatches. We considered as reference genome both the human genome and a concatenation of all complete bacterial genomes. On each dataset, we quantified the capacity of the different tools to retrieve all the occurrences of the reads in the reference genome. Special attention was paid to reads uniquely reported and to reads with multiple hits.

Key words: NGS, benchmarking, short read alignment, Burrows-Wheeler Transform, suffix tree, suffix array, hashing, spaced seeds.

1. INTRODUCTION

NEXT-GENERATION SEQUENCING DATA ARE NOW THE STANDARD to produce genomic and transcriptomic knowledge about an organism, and they are massively produced due to an affordable cost. Mapping short reads against a reference genome is typically the first step to analyze such next-generation sequencing data, and it should be as accurate as possible. Because of the high number of reads to handle, numerous sophisticated algorithms have been developed in the last 3 years to tackle this problem and many mapping tools exist now. These tools usually have their own specificities, and the question of which one to use for a given application is a very vexing question. A very recent article (Ruffalo et al., 2011) presents a comparative analysis of six mapping tools run on the human genome. Their criteria to compare the performances of the mapping tools are based on the quality score, computed by the different tools, of the retrieved mappings. For a given set of reads, they define the accuracy of a mapping tool as the proportion of "good" mapping at the

---

[1]INRA, UR1077 Unité Mathématique Informatique et Génome, Jouy-en-Josas, France.
[2]INRA, Unité de Recherche Génomique Info, Versailles, France.
*These two are First authors.

correct location. They globally analyze the accurracy with respect to various parameters such as the error rate, the size and the frequency of the indels in the reads. Our aim in this article is also to compare some mapping tools (four tools in common with the previous reference and five additional ones), but we choose to build a more controlled benchmark in order to study quantitatively some detailed aspects of the mapping task. By more controlled benchmark, we mean that we know how the reads really map to the reference genome and that all reads have the same amount of errors. We thus address the following questions: Are the tools capable to systematically map a read occurring exactly (with no mismatch) in the reference genome? Can they always do it for a read having as many errors as the maximum number of mismatches allowed in the alignments? For reads occurring at several positions, do they retrieve all the occurrences or only a subset? Do the reads reported as unique really occur only once along the genome? As we will see, the answer will not always be positive, so it is important to know the limitation of each tools.

We have evaluated the performance of the nine following mapping tools: BWA (Li and Durbin 2009), Novoalign (Novocraft, 2010), Bowtie (Langmead et al., 2009), SOAP2 (Li et al., 2009), BFAST (Homer et al., 2009), SSAHA2 (Ning et al., 2001), MPscan (Rivals et al., 2009), GASSST (Rizk and Lavenier, 2010), and PerM (Chen et al., 2009).

They can be divided into two main categories according to the type of algorithm they are based on: hash table based algorithms (indexing either the reads or the reference genome) and Burrows-Wheeler Transform based algorithms (see Table 2 below). MPscan uses a different approach based on suffix trees. In Section 2, a description of these algorithms will be presented. Section 3 will then describe the benchmarks we have built to compare these nine mapping tools. Results are given in Section 4 and are discussed in Section 5.

2. ALGORITHMIC OVERVIEW

In this section, our aim is to describe the algorithms on which mapping tools are based. Although these algorithms are complex, we tried to make them as clear as possible. The beginning of each section broadly describe the methods and the structures, then a more in-depth description follows.
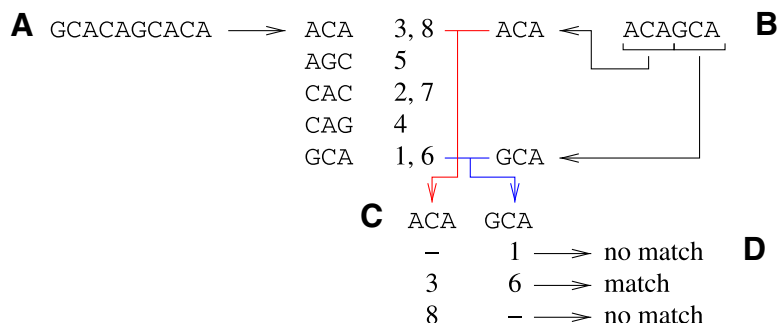
2.1. Hashing

2.1.1. Basic algorithms. Let us suppose, for the sake of convenience, that all the reads have the same size, say 36 nucleotides.

The most straightforward way of finding all the occurrences of a read, if no gap is allowed, consists in "sliding" the read along the genome sequence and noting the positions where there exists a perfect match. Unfortunately, although conceptually simple, this algorithm has a complexity $O(L_G L_r N_r)$ where L_G is the size of the genome sequence, L_r the size of the read and N_r is the number of reads. When gaps are allowed, one has to resort to the popular dynamic programming algorithm, such as the Needleman and Wunsch algorithm, whose complexity is also $O(L_G L_r N_r)$. Algorithms with this complexity are far too slow for aligning several hundreds millions reads on the human genome as the case may happen nowadays.

Therefore, to be efficient, all the methods must rely on some sort of pre-computing. For instance, it is theoretically conceivable to compile a list of all the words of length 36 (36-mers) that are found in the genome and determine once for all their positions. Then, we can use a *hashing* algorithm to transform a string into a key that allows a fast search. Consequently, finding the positions of a read would be straightforward using this list. However, this method does not work in practice since it requires too much computer memory or disk space. Indeed, in the worst case, assuming that all the 36-mers generated from the genome are different, there are about L_G such words. In addition, this method does not allow us to consider mismatches easily.

A possible workaround to this problem of space, is to store all k-mers in a list, using a value of k significantly less than the read size, say $k = 9$ (Fig. 1A). Some tools chose to select a k-mer for each read (a good choice is the leftmost part, because the quality is better) and map it to the genome using the hashing procedure. In terms of space, the problem is now more tractable since there is, at most, $4^9 = 262144$ different 9-mers in the genome. Then, for each possible hit, the procedure would then try to map the rest of the read to the genome (possibly allowing errors, in a Needleman–Wunsch-like algorithm). This two-steps strategy, called *seed and extend*, is implemented in SSAHA and Stampy (Lunter and Goodson, 2010).

FIG. 1. The hashing algorithm.
(A) The genome is cut into over-
lapping 3-mers, and their respective
positions in the genome are stored.
(B) The read is cut into 3-mers. The
3-mers from the reads are compared
to 3-mers from the genome using a
hashing procedure. (C) Positions
for each seed are sorted and com-
pared to the other seeds. (D) Com-
patible positions are kept.

A GCACAGCACA ⟶ ACA   3, 8 ┬─ ACA ⟵ ACAGCA B
                     AGC   5
                     CAC   2, 7
                     CAG   4
                     GCA   1, 6 ┴─ GCA ⟵

C   ACA   GCA
    –   1 ⟶ no match   D
    3   6 ⟶ match
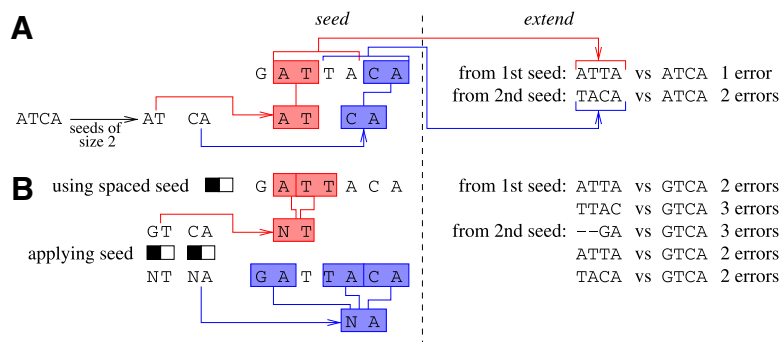   8   – ⟶ no match

A drawback of the previous method is that seeds may be highly repeated in the genome. As a conse-
quence, many hits should be verified in the ''extend'' phase, which is time consuming. Another approach is
to divide each read into four 9-nucleotide-long substrings. Then, as previously, every substring of a read
can be matched using the 9-mer list (Fig. 1B). If the four substrings of a read are found in the list, in the
correct order and adjacent to each other, the read exists in the genome (Fig. 1C, D). However, this
algorithm does not permit to consider mismatches.

Chopping reads into smaller pieces can also be used to find reads with errors (mismatches and indels).
Suppose that one allows for two errors when mapping reads, then one can be sure that at least two out of the
four 9-mers will map exactly (in the worst case, the two errors are located in two different 9-mers, thus
leaving two 9-mers without error) (Fig. 2). This is known as the *pigeon hole* principle. The two 9-mers that
exactly match the genome constitute an anchor. A more finely-tuned search, in the immediate vicinity of
this anchor, is able to recover the read containing errors. This two-steps strategy is implemented for
instance in MAQ (Li et al., 2008), SOAP, RMAP (Ning et al., 2001), and SeqMap (Jiang and Wong, 2008).

A similar method, called *q-gram filtering*, is an extension of the pigeon hole principle. The main
difference is that the reads are not chopped into non-overlapping k-mers, but every k-mer is extracted and
mapped to the genome (in the read ACGT, the 2-mers would be AC, CG, and GT). If a sufficient number of
k-mers map in a small region, the hit is then chosen for a more careful alignment. This method has been
used for instance in SHRiMP2 (David et al., 2011) and RazerS (Weese et al., 2009).

Still, for real instances the list is usually too large to be kept in memory. Characters are usually stored
using a byte (8 bits). DNA alphabet only consists of the four letters A, C, G, and T. To gain memory space,
some methods choose to encode each nucleotide with 2 bits (e.g., A is 00, C is 01, G is 10, and T is 11).
Notice that such an encoding does not allow the handling of ambiguous nucleotides (N), which sometimes
exist in the reads or the reference genome. To get around this problem, different methods adopt different
strategies. For instance, BWA randomly transforms all ambiguous nucleotides into a regular nucleotide.
SOAP2, arbitrarily, transforms them into Gs in the reads.

FIG. 2. The seed and extend al-
gorithm. (A) A read ATCA is sought
for in GATTACA, using seeds of
size 2, with one error. Each seed
maps once (left part). After exten-
sion of each seed (right part), it turns
out that only one mapping contains
only one error. (B) A read GTCA is
sought for in GATTACA, using
spaced seeds of size 2, with two er-
rors. Notice that using simple seeds
would not retrieve any hit. The
spaced seed mask is used to generate
the seeds: xC and xA. The hashing
algorithm retrieves many hits. After
extension, three hits are kept.

A   seed | extend
                             |
        G A T T A C A | from 1st seed: ATTA vs ATCA 1 error
                             | from 2nd seed: TACA vs ATCA 2 errors
ATCA ──seeds of──→ AT   CA A T   C A |
       size 2
                             |
B using spaced seed ■□ G A T T A C A | from 1st seed: ATTA vs GTCA 2 errors
                             | TTAC vs GTCA 3 errors
        GT   CA   N T | from 2nd seed: --GA vs GTCA 3 errors
applying seed ■□ ■□ | ATTA vs GTCA 2 errors
        NT   NA   G A T T A C A | TACA vs GTCA 2 errors
                      N A |

Let us note that the reads too can be hashed, in which case one searches to match regions of the genome, having the size of the reads, to the reads. Some of the first developed tools used this strategy (e.g., MAQ, RMAP, and SeqMap). However, since nowadays sequencers are capable of producing huge amounts of reads in a single run, up to one billion reads, hashing the reads is now too memory demanding. As a consequence, current mapping tools now hash the genome.

2.1.2. Improvements. A major drawback of the seed and extend algorithm is that the reads need to be split into smaller and smaller substrings when the number of allowed errors increases. However, small substrings (say, of four nucleotides) are not effective during the seed phase, since they can match many regions of the genome. For this reason, seeds of less than 10 nucleotides are seldom used. This implies that hashing-based algorithms cannot map reads with many errors.

To circumvent this problem, it is possible to use spaced seeds (as it is implemented in ZOOM [Lin et al., 2008], BFAST, GASSST, SHRiMP2, or PerM, for instance), i.e., seeds with so-called "don't care" positions. A "don't care" position, "x", is a position in the read where the algorithm does not check the type of nucleotide present. For instance, the simple spaced seed ACGxACG is able to match ACGAACG, ACGCACG, etc. It is clear that using a set of spaced seeds instead of a single regular seed increases the sensitivity of the method, although it has also the side-effect of increasing the computation time. For instance, in Figure 2B, we can observe that each spaced seed yields more hits than normal seeds.

In fact, most of the time spent by a seed and extend algorithm is generally spent in the "extend" part (this part solves the problem: "Given a seed, do we have a match around it?"). This part usually uses some form of the dynamic programming algorithm (e.g., Needleman & Wunsch if insertions and deletions are considered) to do so. Most tools use widely known optimizations but some, like SHRiMP, uses also special CPU commands to parallelize the job.

Other methods, like GASSST, add an intermediate step in the pipeline, and use a *seed, filter and extend* algorithm. Each time GASSST finds a seed, it quickly compares the neighboring region with the rest of the read, using an algorithm which is much faster than the dynamic programming algorithm. The filter consists in computing the so-called Euler distance. This distance just computes the number of letters of each type in the compared regions. For instance, if one tries to map a read containing 3 G with a region that only contains 1 G, there will be at least 2 errors in the alignment. Regions that fail this filter are thus quickly discarded.

Authors of PerM observed that, in current mapping tools, several seeds are usually used to achieve good specificity. As a consequence, several hashing structures are used, which increases the required space. PerM simply uses one atomic small gapped seed which is repeated to cover the reads. For instance, for a 36-nucleotide long read, four seeds 1111xx1xx of size 9 are used (where x is a "don't care" sign). Every atomic seed is used to map parts of the reads and promising hits are then investigated for full mapping. Since PerM uses only one hashing structure, it can use seeds with higher weights (i.e., with less don't care) for the same specificity, thus saving time.

2.2. Burrows–Wheeler transform based algorithms

2.2.1. Suffix trees. Hashing based algorithms give poor results when the reads fall into repeated regions, since many seeds should be checked. Some clever data structures have been recently used to solve this problem: the suffix trees and the suffix arrays. Remember that a suffix of a string is a substring that starts at any position and whose end coincides with that of the string. For instance, TTACA is a suffix of GATTACA, whereas TTAC is not. A suffix tree is a tree in which there is a one-to-one correspondance between the paths from the root to the leaves and the suffixes existing in a string, in other words all the suffixes of the string exist as a path joining the root to a leave in the tree (Fig. 3). Notice that some space is saved since all the suffixes are not explicitly written. Actually, current algorithms build suffix trees whose size is proportional to the size of the genome, in a time proportional to the size of the genome, too.

Now, with a suffix tree, looking for a read in a genome is very simple, provided no error is allowed. Suppose we look for GAT in the genome. We start from the root and take the branch which is labelled G. We arrive to a new node, and follow the branch A, then at the next node the branch labelled T (Fig. 3). If we can successfully spell the read, then it is present in the genome. Notice, for instance, that we cannot spell the word GAG. The search, if successful, is $O(L_r)$. We will show below how to solve the problem with errors.

FIG. 3. The suffix tree of the genome GAT-TACA. Dotted arrows indicate that the tree continues there. Double circle indicate that a suffix ends there.
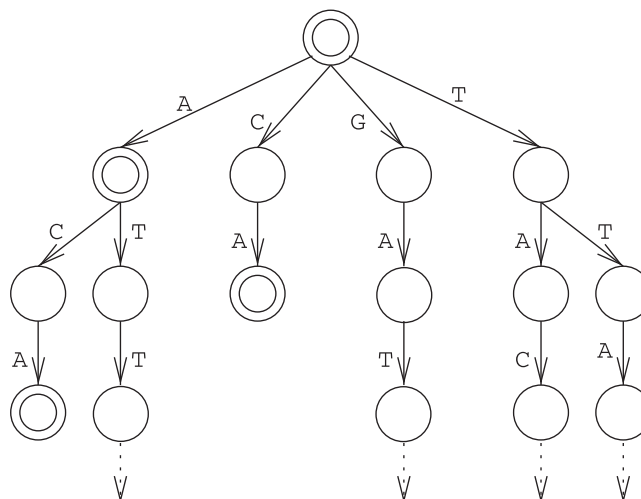
Notice that if the word we are looking for is in a repeated region, it will not change the time spent to retrieve it. This structure somehow ''collapses'' the genome and squeezes the repeated regions into only one path.

The MPscan program uses a slightly modified version of a suffix tree to store the reads (instead of the genome). Using a dedicated algorithm, it scans the whole genome and checks if one of the reads stored in the suffix tree matches some part of the genome. While the method is very fast and uses no heuristics that may give rise to false positives, it only supports exact alignment.

2.2.2. Suffix arrays.   Despite all the memory optimizations, it seems that suffix trees are still too big to fit in RAM for large genomes such as the human genome. Suffix arrays have thus been proposed to overcome this difficulty. Even though using a suffix array is a bit more tricky than using a suffix tree, its definition is much simpler: a suffix array is the set of suffixes of the genome sorted lexicographically (Fig. 4). Of course, if we had to store all the suffixes, it would be untractable. So, a first trick is to store the position of the beginning of the suffixes. The rationale is that the whole suffix can be recovered at once using its start position and the genome. Finally, a suffix array is a mere list of numbers: the start positions of the lexicographically sorted suffixes.

Although this may not be immediately apparent, there is a strong connection between suffix trees and suffix arrays. For instance, let us consider the first column of the suffix array. It consists of the set of letters labeling the edges that connect the root of the suffix tree to the first level of nodes. Similarly, the first two-letter words of the suffix array correspond to the paths from the root to the second level nodes, and so on. This suggests that it is possible to traverse a suffix array in a way similar to what is done with a suffix tree.

There are several ways to do so. The most popular solution is to use the Burrows–Wheeler transform. This transform has been described for text compression but it has many interesting properties in bioinformatics.

In this transform, it is useful to consider the suffixes as being written on cylinders, meaning that the first letter of the genome comes after the last letter of the suffix, and so on (Fig. 4). In other words, we write down all the circular permutations of the genome string. To mark the end of the genome, we append a sigil

FIG. 4. Suffix array of the genome GATTACA. Last column is the Burrows-Wheeler transform.

genome	suffixes	sorted suffixes	positions	cylinder suffix array	B–W
GATTACA$	GATTACA$	ACA$	4	ACA$GATT	T
	ATTACA$	ATTACA$	2	ATTACA$G	G
	TTACA$	A$	6	A$GATTAC	C
	TACA$	CA$	5	CA$GATTA	A
	ACA$	GATTACA$	1	GATTACA$	$
	CA$	TACA$	4	TACA$GAT	T
	A$	TTACA$	3	TTACA$GA	A
	$	$	7	$GATTACA	A

B–W	sorted	concat. 1	sort	concat. 2	sort
T	A	TA	AC	TAC	ACA
G	A	GA	AT	GAT	ATT
C	A	CA	A$	CA$	A$A
A	C	AC	CA	ACA	CA$
$	G	$G	GA	$GA	GAT
T	T	TT	TA	TTA	TAC
A	T	AT	TT	ATT	TTA
A	$	A$	$A	A$A	$GA

FIG. 5. Reconstructing the suffix array.

$ to it. By convention, the sigil is the last character in the lexicographic order. The Burrows-Wheeler transform of a suffix array is simply the last column of this cylinder suffix array. In fact, a Burrows-Wheeler transform merely is a list of characters (shown in the last column of Fig. 4).

Notice that suffix arrays, together with their Burrows-Wheeler transforms, are very compact. It is just a set of two vectors: a vector of numbers (the suffix array) and a vector of characters (the Burrows-Wheeler transform). The size of the arrays are equal to the size of the genome.

It is possible to retrieve the whole suffix array from the Burrows-Wheeler transform only (Fig. 5). For this, we begin by sorting lexicographically the list of characters (see column 2). We can observe that this corresponds exactly to the first column of the cylinder suffix array (since suffixes are sorted lexicographically). We now have the first and the last columns of the cylinder suffix array (the last column being the Burrows-Wheeler transform). Since we work on a cylinder, we know that the last column is also the one which is right before the first column. So, when we concatenate the two columns, we have the set of all two-letter words of the genome (column 3). Again, if we sort them lexicographically (column 4), they form the first two columns of the cylinder suffix array. If we prepend the Burrows–Wheeler transform to this set, and we sort it, we have the three first columns of the cylinder suffix array (column 5). Proceeding likewise, the whole suffix array is retrieved. The original genome corresponds to the line ending with the sigil.

2.2.3. Finding a word with the Burrows-Wheeler algorithm. The algorithm previously described is not used to find the mappings of a read, it builds iteratively all the two-letter words of the genome, then three-letter words, etc. This is useless and time consuming for our problem. What we actually need is a way to traverse the array, just as we did for the suffix tree: letter by letter. Restricting the search to a specific read has been proposed in the frame of the Ferragina–Manzini algorithm (Ferragina and Manzini, 2000), and it is actually much more complicated than the algorithm previously described. First, it proceeds from right to left, meaning that if we look for the read GAT in the genome GATTACA, we will first look for the T, then the A, and finally the G. Second, it mainly uses two data structures: the transform itself and the sorted transform (which is the first column of the array, and can be computed from the transform by sorting it).

We first look for all the occurrences of T in the sorted transform (this search can be done in constant time using an additional data structure). There are two Ts, at rows 6 and 7 (Fig. 6a). The corresponding characters in the transform are the letters T and A respectively (Fig. 6b). This means that the only letters which are before a T in the genome are a A and a T. Here, we are interested in the A (Fig. 6c). We now would like to know what are the letters which are before this AT. Here comes a magic trick, which will be explained later. The A of the row 7, that we have selected, is the second A when we read the Burrows–Wheeler transform from top to bottom. So we jump to the second A of the sorted column (again, this can be done in constant time with yet some additional data structures). It corresponds to the A at row 2 (Fig. 6d). The search recursively resumes here. The letter before the AT is found in row 2 of the transform. It is a G, which shows that the read GAT exists in the genome (Fig. 6e).

Suffix arrays behave equally well as suffix trees when handling repetitions. Since the array is sorted, if a word is repeated twice, the occurrences will appear in two consecutive rows. As a consequence, the algorithm actually works with intervals of rows. As in Figure 6a, b, the search algorithm considers as putative match all the solutions which are between rows 6 and 7.

B–W	sorted		B–W	sorted		B–W	sorted		B–W	sorted
T	A		T	A		T	A		T	A
G	A		G	A		G	A		G	A
C	A		C	A		C	A		C	A
A	C		A	C		A	C		A	C
$	G		$	G		$	G		$	G
T	T		T	T		T	T		T	T
A	T		A	T		A	T		A	T
A	$		A	$		A	$		A	$
(a) Finding Ts.			(b) Finding the let-ters before Ts.			(c) Selecting AT.			(d) Finding the cor-responding A.	

B–W	sorted
T	A
G	A
C	A
A	C
$	G
T	T
A	T
A	$

(e) The read GAT is present.

FIG. 6. Looking for the read GAT.

An explanation of the magic trick used above is the following. The whole problem is to look for the letters which are before a given word in the genome. If we want to know which letters are before a T, we can look for the Ts in the sorted transform (rows 6 and 7) and get the corresponding characters in the transform (T and A). By construction, these letters are right before the Ts. To proceed, we need to know which letters are before AT. Suppose we are able to generate the sorted list of 2-letter words of the genome (see column 4 of Fig. 5). We then can use the Burrows–Wheeler transform to find which nucleotides are before AT (here, it is row 2). But since we do not have the second column, but only the first one, all that we know is that we should focus on the rows such that the sorted transform is an A, namely rows 1, 2 or 3 (Fig. 7a). The problem is to find to which row AT corresponds to.

In the transform, these As are located at rows 4, 7, and 8 (Fig. 7b). Using the sorted transform, we can see that they correspond to the two letter words AC, AT, and A$ (Fig. 7c). Notice that these words are sorted according to the second letter, when we read from top to bottom. This is a direct consequence of the fact that the sorted transform is the first column of the sorted suffix array, and that the Burrows–Wheeler is the set of nucleotides that are before this column. This is why AT is after AC and A$ is after AT.

Now, we know that AT is the second 2-letter word starting with a A. It is easy to find it in the sorted transform, since it is also sorted: it is simply the second A, read from top to bottom. This is row 2 (Fig. 7d). We can finally retrieve the G which is before this A, and the word GAT is found.

B–W	sorted		B–W	sorted		B–W	sorted		B–W	sorted
T	A		T	A		T	A		T	AC
G	A		G	A		G	A		G	AT
C	A		C	A		C	A		C	A$
A	C		A	C		AC	C		A	C
$	G		$	G		$	G		$	G
T	T		T	T		T	T		T	T
A	T		A	T		AT	T		A	T
A	$		A	$		A$	$		A	$
(a) The As in the sorted Burrows–Wheeler transform.			(b) The As in the Burrows–Wheeler transform.			(c) The correspond-ing 2-mers in the Burrows–Wheeler transform.			(d) The correspond-ing 2-mers in the sorted Burrows–Wheeler transform.	

FIG. 7. The magic trick explained.

2.2.4. Search with errors. While the Ferragina–Manzini algorithm permits to retrieve efficiently a read in a genome, the algorithm described above does not allow for errors. Adapting the Burrows–Wheeler algorithm to a search with errors is currently an active field of research. SHRiMP, for instance, when no exact match is found, restarts the search and accepts errors, randomly. Suppose that the read GCT is to be found in GATTACA. The letter T is matched first (remember that the reads are searched for T from right to left), then the algorithm finds that the letters before T in the genome are either T or A, but not C. SHRiMP randomly picks up one of these two letters and records one error. This procedure is carried out several times, and either a match is found, or the number of allowed errors is exceeded and the search is brought to a close. Of course, this algorithm may not provide the best solution. In recent developments, BWA proposed an interesting adaptation to suffix arrays of the well known Smith–Waterman algorithm. However, it seems that the algorithm is too slow when too many errors are allowed and BWA first aligns the leftmost and the rightmost parts of the read with a fixed small number of errors, hoping that at least one of these regions contains few errors (which is grounded by the fact that the $5'$ ends of the reads usually have a better quality).

2.3. Remarks

Two categories of methods exist to map reads on genomes: those based on hashing or the ones using Burrows–Wheeler transform. Although it is difficult to foresee what will be the trend in the future, it seems that most recent tools are based on the Burrows–Wheeler transform. Each method has strong and weak points. Burrows–Wheeler-based methods are fast even when a read has many possible locations in the genome because the latter is somehow "collapsed" and repeated regions are scanned only once. However, there is still no established algorithm to handle errors, and for the existing heuristics, the computation time does not scale well when the number of errors increases. On the other hand, hashing methods can handle errors, as long as they are not spread uniformly along the read (otherwise no seed may be found). However, seeds of regions which are highly repeated are not specific and mapping may be slow for repeated sequences. Maybe a good option is, as mentionned by the authors of Stampy, to use both methods one after the other to avoid the problems of each single method.

Another interesting trend involves the quality of the mapping. What is the probability that a read comes from a predicted region? Given the number of mappings for a read, the number of mismatches and the sequencing quality, some tools like MAQ, BWA, and Stampy try to infer a probability of correctness of the mapping. As a consequence, the user may want to control the probability of the reads to be misaligned instead of the maximum number of errors, which may make more sense.

Last but not least, the way the algorithm are implemented has a great impact on the speed of the mapping tools. For instance, reading or writing to a hard disk is extremely slow and should be avoided for the frequent operations. As a consequence, most tools use several strategies to accelerate the search. They include: parallelization of the process (like bit–wise operations to compare two strings), caching results (such as Smith–Waterman results, should the same comparison be computed once again afterwards), and scaling data structure so that they can fit on RAM (by removing data which can be computed on the fly). Undoubtedly, the successful modern mapping tool designers consider equally important both the algorithm and the implementation details to deliver a time effective tool.

3. BENCHMARK DESCRIPTION

3.1. Datasets

The mapping tools are evaluated by two similar experiments. The first experiment (named $\mathcal{H}$ in the following) is run on the human genome (25 chromosomes for 2.7 Gbp). The second experiment (named $\mathcal{B}$) is run on bacterial genomes (904 genomic sequences for 1.7 Gbp).

In the experiment run on the human genome ($\mathcal{H}$), the reference genome $\mathcal{H}_{ref}$ is taken from the assembly 37.1 made by the NCBI. We have built two sets of reads, all of length 40. The first set of reads ($\mathcal{H}_0$) is composed of 10 millions reads drawn uniformly from the reference genome $\mathcal{H}_{ref}$. The drawing is done with wgsim.[1] Human chromosomes sometime contain a large proportion, as high as 30%, of the letter N.

---

[1]Available from https://github.com/lh3/wgsim or SAMtools and developed by Heng Li.
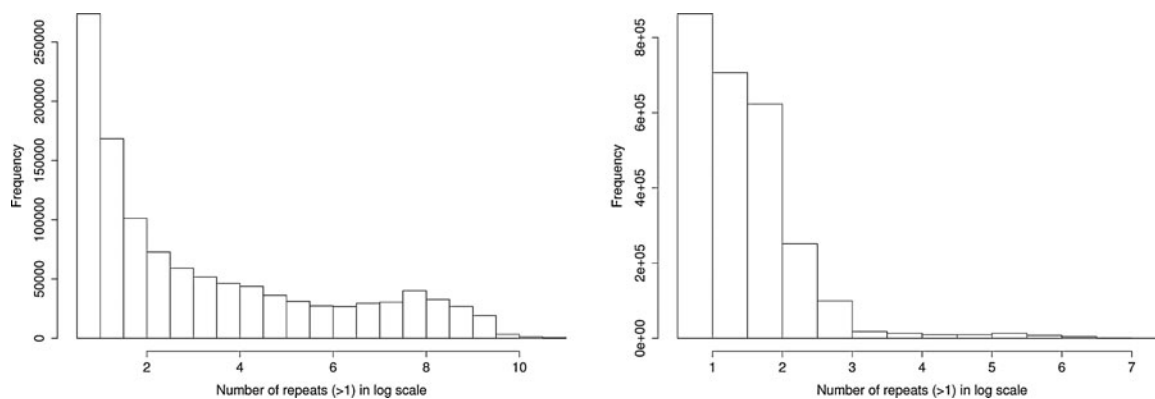
FIG. 8. Histogram of the logarithm of the number of occurrences of the 1,122,893 (respectively 2,620,394) reads from $\mathcal{H}_0$ (resp. $\mathcal{B}_0$) occurring more than once in the reference genome **(left)** (resp. **(right)**).

TABLE 1. NUMBER OF READS WITH A GIVEN NUMBER OF NS, FROM EACH OF THE FOUR DATA SET $\mathcal{H}_0$, $\mathcal{H}_3$, $\mathcal{B}_0$, $\mathcal{B}_3$

	1	2	3	4	5	6	8	9	10	37	40	Total
$\mathcal{H}_0$	5	2	3	2	5	1	0	3	27	0	1	49
$\mathcal{H}_3$	5	2	6	1	4	0	4	15	11	1	0	49
$\mathcal{B}_0$	193	25	6	2	1	1	1	1	1	0	0	231
$\mathcal{B}_3$	187	21	4	3	0	2	0	1	1	0	0	219

Mapping reads with long runs of N is of little information to assess the efficiency of mapping tools because these reads should map in numerous locations. We thus decided, beforehand, to remove runs longer than 10 Ns from the reference genome.[2] The majority of the reads (8,877,107) from $\mathcal{H}_0$ occurs only once,[3] but some reads can be repeated many times along the reference genome as shown on Figure 8. For reads occurring more than once, the mean number of occurrences is 722.81 with a standard deviation of 2424.86. Moreover, the most frequent read occurs 53,162 times. The second set of reads ($\mathcal{H}_3$) is built from $\mathcal{H}_0$ by adding exactly three mismatches to each read. Therefore $\mathcal{H}_3$ contains also 10 millions reads. We are aware that modern sequencers are very unlikely to produce reads with such an error rate, but the rationale of this dataset is that many projects now produce resequencing and metagenomics data, which may diverge greatly from already sequenced genomes. The positions for the three mismatches are drawn uniformly within the 40 positions.[4] A nucleotide A, C, G, or T is mutated to any of the three other nucleotide with equal probability 1/3, whereas an N is mutated into A, C, G, or T with probability 1/4. Among the 10 millions reads from $\mathcal{H}_0$ and $\mathcal{H}_3$, only 49 reads contain some Ns; the number of Ns per read is given in Table 1.

In the second experiment ($\mathcal{B}$), the reference genome $\mathcal{B}_{ref}$ consists of 904 bacterial genomes found in Genome Reviews release 111.0 (Kersey et al., 2005). We also built two sets of 40-bps reads. The first set of reads ($\mathcal{B}_0$) is composed of 10 millions reads drawn uniformly from $\mathcal{B}_{ref}$.[5] There are 7,379,606 reads with a unique occurrence and 2,620,394 reads occurring more than once (mean 8.82 and SD 39.03). The most frequent read from $\mathcal{B}_0$ occurs 1685 times. The second set of reads ($\mathcal{B}_3$) is built from $\mathcal{B}_0$ by adding uniformly three mismatches in each read as described for $\mathcal{H}_3$. 231 reads from $\mathcal{B}_0$ and 219 reads from $\mathcal{B}_3$ contain some Ns (Table 1).

---

[2]One run of 40 Ns is nevertheless kept in the experiment and this read is included in the $\mathcal{H}_0$ set.

[3]The true number of occurrences of each read from $\mathcal{H}_0$ have been computed thanks to a dedicated naive algorithm.

[4]We want each read to contain *exactly* three mimatches, hence a mismatch position cannot be drawn twice. Uniform drawing means that the first position is drawn uniformly within the 40 positions, the second position is drawn uniformly within the 39 remaining positions, and the third position is drawn uniformly within the 38 remaining positions.

[5]Unlike human chromosomes there are few Ns in bacterial genomes, so we did not need sequence curation.

TABLE 2. GLOBAL CHARACTERISTICS OF THE MAPPING TOOLS

Tool	Format	Algorithm	Threads	Gaps	Mismatches
BWA	SAM	BWT	yes	yes	yes
Novoalign	SAM	hash the ref.	yes	yes	yes
Bowtie	SAM	BWT	yes	no	yes
SOAP2	perso	BWT	yes	no	at most 2
BFAST	SAM	hash the ref.	yes	yes	yes
SSAHA2	SAM	hash the ref.	no	no	yes
MPscan	perso	suffix tree	no	no	no
GASSST	SAM	hash the ref.	yes	yes	yes
PerM	SAM	hash the ref.	no	no	yes

SAM, Sequence Alignments Map.

3.2. Mapping tools

We evaluated the performance of the following nine mapping tools: BWA_v0.5.8, Novoalign_v2.06.09, Bowtie_v0.12.7, SOAP_v2.20, BFAST_v0.6.5a, SSAHA2_v2.5.2, MPscan, GASSST_v1.28, and PerM_v0.3.9.

Table 2 gathers global characteristics of the tools, namely the type of algorithms they are based on, their output format, their ability to allow mismatches and/or indels in the alignments, and if they can use multiple threads.

Additional information is now given separately for each tools, in particular on the way to perform the comparisons. For each tool, our aim is indeed to retrieve all the alignments (so-called hits hereafter), either with no mismatch or with at most 3 mismatches, of our read datasets (see Section 3.1).

3.2.1. BWA. Running BWA consists in using successively three commands: the first one (bwa index) indexes the reference genome, the second one (bwa aln) finds the coordinates of the hits of each individual read in the suffix array, and the last one (bwa samse) converts the suffix array coordinates to reference genome coordinates and generate the alignments in the SAM format. By default, a non exhaustive search is done in the second step to reduce the computation time; we then used the $-N$ option to disable this behavior and to search for all possible hits. Using this option or not has a dramatic effect on the results when mismatches are allowed, as we will see in the next section. It is possible to set the maximum number of mismatches per hit (option $-n$ in the second step) and also per seed (option $-k$); we used the same value for both parameters. Moreover, one can specify the maximum number of hits to output (option $-n$ in the third step). If a read has more hits in the reference, outputted hits are randomly chosen. The only way to get all the hits per read is to set the maximum number of hits to output to a value larger than the maximum number of occurrences of the reads in each read set. We then took the bounds 54,000 for $\mathcal{H}_0$ and $\mathcal{H}_3$,[6] and 2,000 for $\mathcal{B}_0$ and $\mathcal{B}_3$.[7] BWA randomly changes Ns in the reference genome to regular nucleotides.

3.2.2. Novoalign. Running Novoalign consists in running two successive comands: the first one (novoindex) indexes the reference genome and the second one (novoalign) aligns the reads against the indexed reference. Novoalign (at least in its academic version) does not allow the user to set the maximum (or exact) number of mismatches between the read and the reference genome. We then post-processed the results to retrieved exact matches ($\mathcal{H}_0$ and $\mathcal{B}_0$) or matches with at most three mismatches ($\mathcal{H}_3$ and $\mathcal{B}_3$). For reads with multiple hits, it is possible to report all hits (option $-r$ A) or at most a fixed number of randomly chosen hits.

3.2.3. Bowtie. Running Bowtie consists in using successively two commands, bowtie-build which indexes the reference genome and bowtie which takes an index and a set of reads as input and outputs a list of alignments. Bowtie allows the user to set the maximum number of mismatches per hit (option $-v$). By default, Bowtie returns only one hit per reads; if one wants to retrieve more, or all, hits per read, one needs

---

[6]The highest number of hits retrieved by BWA for $\mathcal{H}_3$ is 30,265, so far less than the 54,000 bound.
[7]The highest number of hits retrieved by BWA for $\mathcal{B}_3$ is 1,749, so far less than the 2,000 bound.

to specify a maximum number of hits to report (option $-k$). As for BWA, this maximum number should be set to the maximum number of occurrences of the read set[8] to retrieve all the hits. Alignments involving one or more ambiguous characters, such as Ns, in the reference are considered invalid by Bowtie, whereas they account for mismatches if they belong to the reads.

3.2.4. SOAP2. Running SOAP2 consists in using successively two commands: the first one (2bwt-builder) creates the Burrows-Wheeler index of the reference genome, and the second one (soap) performs the alignments. SOAP2 allows the user to set the maximum number of mismatches per hit (option $-v$) but this maximum number is limited to 2. SOAP2 outputs systematically all the hits (no limitation is allowed). Unmapped reads can be obtained in a FASTA file. SOAP2 seems to replace all the Ns in the reads by a G.

3.2.5. BFAST. Running BFAST requires five steps: (1) the reference genome is first rewritten in a special format (bfast fasta2brg), (2) bfast index indexes the reference genome by using the spaced seeds set by the user (this step should be done with several seeds, leading then to several indexes; we used the 10 seeds proposed in Homer et al., [2009]), (3) then the bfast match command takes a set of reads and searches a set of indexes to find candidate alignment locations (or CALs) for each read, (4) the bfast localalign command takes the CALs for each read and performs a local alignment to the reference, and (5) finally an output file is created (bfast postprocess). As for Novoalign, the user cannot set the maximum (or exact) number of mismatches, so we post-processed the outputted hits. BFAST can output all the hits (option $-a$).

3.2.6. SSAHA2. Running SSAHA2 consists in two steps: indexing the reference genome (command ssaha2Build) and mapping the reads (ssaha2). It is possible to specify the number of mismatchs allowed, or equivalently the percentage of identity (option $-identity$). The number of reported hits per read is limited to 500 and cannot be changed. We asked for the ''best'' (Smith-Waterman score) mapping for each read ($-best\ 1$), which seems appropriate for exact mapping, but probably not for $\mathcal{H}_3$ and $\mathcal{B}_3$ (actually we also used $-best\ 0$ in the mismatches case).

3.2.7. MPscan. To run MPscan, there is only one command (mpscan) but it must be used twice, one for mapping on the direct strand and the second one for the reverse strand (options $-rev\ -ac$). No mismatch is allowed in the alignments and all alignments are reported in the output file (not in SAM format).

3.2.8. GASSST. Indexing and mapping steps are performed by issuing the Gassst command. It is possible to specify the number of mismatchs allowed, or equivalently the percentage of identity (option $-p$). To retrieve exhaustively all the hits for each read, we disabled the filtering process used by default to reduce the computation time (option $-l\ 0$) and we set the sensitivity to its maximal value (option $-s\ 5$). Alignments involving ambiguous characters necessarily account for mismatches. GASSST reports the alignments in a specific format; the output file can be converted in SAM format by using the command gassst_to_sam which appeared to be quite time consuming.

3.2.9. PerM. Indexing and mapping steps are done by running the perm command. It is possible to set the maximum number of mismatches per hit (option $-s$) and to specify a maximum number of hits to find (option $-k$). To report all the hits, we set the previous option to the highest number of occurrences found in our read sets (as for Bowtie and BWA) and we also activated the ''all'' option $-A$. Finally, since some of the reads contain some Ns (Table 1), we used option $--includeReadsWN$ followed by 40, 37, or 10 depending on the read set.

4. RESULTS

We present here the results obtained when mapping the human datasets $\mathcal{H}_0$ and $\mathcal{H}_3$ to the human reference genome $\mathcal{H}_{ref}$. Results for the bacterial datasets are presented in appendix because the tools'

---

[8]We took the same maximal values than for BWA, except for $\mathcal{H}_3$ for which we took $-k\ 400,000$ because the maximum number of hits retrieved by Bowtie appeared to be 305,424.

TABLE 3. GLOBAL CHARACTERISTICS OF THE RUN OF EACH SOFTWARE ON THE EXACT HUMAN DATASET ($\mathcal{H}_0$)

Software	Memory usage (Gb)	Indexing time	Mapping time	Non-mapped reads	Mapped reads	Original position	
						Retrieved	Not retrieved.
BWA	2.18	1h 36mn	1h 13mn	49	9,999,951	9,999,951	0
Novoalign	8.12	8mn	13h 24mn	632	9,999,368	9,999,368	0
Bowtie	7.36	3h 25mn	2h 42mn	49	9,999,951	9,999,951	0
SOAP2	51.87	*1h 56mn[‡]*	*56mn[‡]*	49	9,999,951	9,996,385	3,566
BFAST	9.68	18h 01mn*	15h 02mn	726,332	9,273,668	9,253,642	20,026
SSAHA2	9.60	24mn	1d 1h	35,875	9,964,125	9,770,914	193,211
MPscan	2.67	1h 20mn		26	9,999,974	9,999,974	0
GASSST	57.93	*8h 45[†‡]*		49	9,999,951	9,999,897	54
PerM	13.77	13h 05mn		115,871	9,884,129	9,884,125	4

Columns refer to memory usage, computation times, number of reads which have not been mapped among the 10 millions reads, number of mapped reads and number of mapped reads whose original position has been retrieved in the complete list of hits.

*Average indexing time per spaced seed computed on 10 seeds.

[†]This time does not include the running time of the gassst_to_sam command.

[‡]This time is slightly over-estimated.

results are globally similar to those obtained on the human datasets and genome even though their performances are slightly worse.

4.1. Exact mapping from $\mathcal{H}_0$ reads

We firstly performed an exact mapping (i.e., no mismatch is allowed) of the read set $\mathcal{H}_0$ onto the human genome with each of the nine mapping tools presented in Section 3.2. Note that, by construction of the read set $\mathcal{H}_0$, all reads from $\mathcal{H}_0$ do exactly occur, at least once, in the reference genome. All the tools should then map all the reads with no mismatch, or at least the reads with no Ns because alignments involving some Ns account for mismatches for many tools.

4.1.1. Computation time and memory usage.
Computation times have been obtained by running the mapping tools in single-thread mode, on the same computer[9] and without any competition. However, as we can see in Table 3, SOAP2 and GASSST require more than 50 Gb of memory, so they have been run on a more powerful computer[10] but without avoiding competition with other jobs (computation times could then be slightly over-estimated for both tools). For tools which are run through several command lines, we have reported the maximum of the memory required by each step. The most memory consuming step for Bowtie and BFAST is the indexing step, whereas it is the mapping step for SOAP2. Indexing times and mapping times have been reported separately whenever these steps corresponded to distinct command lines (Table 3). The general trend is as follows: BWA, Bowtie, SOAP2, MPscan, and GASSST take several hours to map the 10 millions reads (MPscan is the fastest tool), Novoalign and PerM use half a day, and one day or more is required for SSAHA2 and BFAST.

4.1.2. Unmapped reads.
None of the nine mapping tools we have analyzed maps all the reads (Table 3) but for some tools, it is just a question of Ns. Indeed, BWA, Bowtie, SOAP2, and GASSST correctly map all the reads except the 49 reads with some Ns. MPscan only fails to map 26 reads having some Ns and occurring only on the reverse strand. The other tools do not map some of the regular reads. Novoalign could not map 632 reads but succeeded in mapping 22 reads with Ns at their original[11] position. More dramatically, SSAHA2, PerM and BFAST do not map many reads (35,875, 115,871, and 726,332, respectively). BFAST did not map any of the 49 reads with Ns, whereas SSAHA2 mapped 36 reads with Ns but, most of the time, not at their original location. Despite the option $-k$ 54000, PerM seems to discard reads with more than 2000 occurrences: the highest number of hits reported by PerM is 1999 and there are exactly 115,822 reads in $\mathcal{H}_0$ which occur 2000 or more times in $\mathcal{H}_{\text{ref}}$.

[9]Intel Quad Core 2.33 GHz 16 Gb RAM.

[10]Four Intel Six Core 2.40 GHz 132 Gb RAM

[11]The original position of a read is defined as the position randomly drawn to generate the read.

4.1.3. Is the original position retrieved?. For all the mapped reads, we have checked if their original position, (i.e., the one randomly drawn to generate the read) belongs to the complete list of returned hits. Indeed, we did not impose constraints on the number of hits per reads. BWA, Novoalign, Bowtie, and MPscan retrieve the original position of all their mapped reads (Table 3). PerM misses the original position for only four reads and GASSST for 54 reads. However, SOAP2, BFAST and SSAHA2 fail for many reads; we will point out some possible explanations in the next paragraph.

4.1.4. Unique and multiple reads. We can now get in more details and distinguish between *unique* and *non unique* reads. By *unique read* we mean that the read has a unique occurrence into the reference genome wheareas a *non unique* read will have more than one occurrence (repeats). This distinction is important when one knows that many post-analyses only consider reads mapping at a unique location. We know that H_0 is composed of 8,877,107 unique reads (including 46 reads with Ns) and 1,122,893 multiple reads; the later have on average 722.81 occurrences. These reference values are indicated at the bottom of Table 4 ("Reference" row).

We can see in Table 4 that BWA, Bowtie, SOAP2, and GASSST report 8,877,061 reads as unique and at their original position which is correct if one discards the 46 unique reads with Ns. MPscan also correctly reports the unique reads. Novoalign reports the correct number of unique reads, but this is a coincidence: indeed, Novoalign only maps 22 reads with Ns, meaning that some uniquely reported reads are in fact multiple (some hits have been missed). PerM seems to be quite reliable regarding the uniquely reported reads even if it misses some occurrences for three of them. SSAHA2 reports more unique reads than in reality, meaning that it misses some hits for some of these reads; This explains also why 9,847 of these reads are not mapped at their original location.

Regarding the non unique reads, BWA, Bowtie and MPscan seem to provide the correct numbers of hits (722.81 hits on average with a standard deviation of 2424). GASSST, Novoalign and SOAP2 are also quite good. Note that SSAHA2 returns much less hits per read (around 80 on average), leading to original positions not retrieved, but this could be explained by the fact that SSAHA2 limits the number of hits per read to 500. This phenomenon is more drastic for BFAST which only returns three hits on average per read. It looks like these tools do not explore all the possible occurrences leading to many unmapped reads, to many reads not mapped at their original location and to many reads wrongly reported as unique. The case is different for perM: its small mean number of hits (126) per mapped read can be explained by the fact that PerM does not map very frequent reads (more than 2000 occurrences).

4.2. Mapping reads from H_3 with three mismatches

We then performed a mapping, allowing up to three mismatches, of the read set H_3 onto the human genome. Since SOAP2 and MPscan do not allow three mismatches (SOAP2 is limited to two mismatches,

TABLE 4. MAPPING RESULTS FOR EACH SOFTWARE RUN ON THE EXACT HUMAN DATASET (H_0)

Software	Non-mapped reads	Reads uniquely retrieved		Reads with multiple hits		
		Nb	Orig. pos. not retr.	Nb	Mean nb hits [sd]	Orig. pos. not retr.
BWA	49	8,877,061	0	1,122,890	722.81 [2424.86]	0
Novoalign	632	8,877,107	0	1,122,261	698.63 [2171.49]	0
Bowtie	49	8,877,061	0	1,122,890	722.81 [2424.86]	0
SOAP2	49	8,877,061	0	1,122,890	653.26 [1804.95]	3,566
BFAST	726,332	8,840,305	9,193	433,363	2.96 [1.47]	10,833
SSAHA2	35,875	8,886,204	9,847	1,077,921	79.52 [151.74]	183,364
MPscan	26	8,877,081	0	1,122,893	722.81 [2424.86]	0
GASSST	49	8,877,061	0	1,122,890	722.47 [2422.11]	54
PerM	115,871	8,877,068	3	1,007,061	126.42 [333.29]	1
Reference		8,877,107		1,122,893	722.81 [2424.86]	

The number of unmapped reads, the number of reads mapped to a unique location and the number of reads mapped to several locations are displayed. Moreover, for the two later categories, the number of reads whose original position has not been retrieved is reported. The mean number of hits per reads mapped more than once is also presented.

TABLE 5.   GLOBAL CHARACTERISTICS OF THE RUN OF EACH SOFTWARE ON THE 3 MISMATCHES HUMAN DATASET ($\mathcal{H}_3$)

Software	Memory usage (Gb)	Indexing time	Mapping time	Non-mapped reads	Mapped reads	Original position	
						Retrieved	Not Retrieved
BWA	10.01	1h 38mn	17h 04mn	49	9,999,951	9,999,951	0
Novoalign	8.07	8mn	2d 6h	47	9,999,953	9,334,497	665,456
Bowtie	7.36	3h 25mn	9h 57	49	9,999,951	9,999,951	0
BFAST	9.68	18h 01mn(*)	10h 45mn	199,451	9,800,549	8,997,831	802,718
SSAHA2	9.60	24 mn	3d 11h	213	9,999,787	5,537,652	4,462,135
GASSST	27.14	1d 12h(†‡)		326,598	9,673,402	9,631,509	41,893
PerM	13.75	12h 25mn		186,752	9,813,248	9,813,241	7

Columns refer to memory usage, computation times, number of reads which have not been mapped among the 10 millions reads, number of mapped reads and number of mapped reads whose original position has been retrieved in the complete list of hits.

* Average indexing time per spaced seed computed on 10 seeds.

† This time does not include the running time of the gassst_to_sam command.

‡ This time has been obtained on a computer with more memory.

whereas MPscan only performs an exact mapping), we only used the seven other tools. By construction of the read set $\mathcal{H}_3$, all the reads do occur at least once with exactly three mismatches, meaning that (i) all reads should be mapped, (ii) each read is expected to have more hits than in the previous experiment without mismatch (Section 4.1), which leads to (iii) less reads uniquely retrieved.

4.2.1. Computation time and memory usage.   The amount of memory required by each tool is similar to the one required in the previous experiment for Novoalign, Bowtie, BFAST, SSAHA2, and PerM, it increases for BWA (due to the mapping step) while it decreases for GASSST. GASSST still needs more than 16 Gb of memory, so it has been run on a different computer (see Section 4.1.1).

Since the indexes are built on the reference genome, indexing times are identical to the previous experiment, only mapping times may change due to allowed mismatches. The running times of BFAST and Perm decrease, respectively, by 30% and 5%. The mapping times of other methods increase globally by a factor 4, except BWA which becomes 10 times slower (Table 5). To our surprise, BFAST is significantly faster when run on the $\mathcal{H}_3$ instance. The reason for this behavior is not clear to us.

4.2.2. Unmapped reads and reads mapped at their original position.   Again, none of the tools maps all the reads even when three mismatches are allowed (Table 6). The read set $\mathcal{H}_3$ still contains 49 reads with Ns (Table 1). Bowtie and BWA[12] map all the reads with no Ns and their original position is always retrieved. Novoalign maps more reads (47 unmapped reads) but globally the returned list of hits is incomplete for many reads (666,456 reads). SSAHA2 maps a reasonable number of reads but half of them are not mapped at their original position indicating a very partial list of reported hits; this is not really due to the −best option set to 1 because we still obtain 4,325,039 reads not mapped at their orginal position when −best is set at 0. BFAST, PerM and GASSST do not map many reads but the diagnostic is different: PerM does not map frequent reads (2000 occurrences or more) but maps the other reads at their original location, whereas GASSST and BFAST seem to produce incomplete lists of hits. PerM seems then to be the most reliable tool after Bowtie and BWA.

4.2.3. Unique and multiple reads.   The mean number of hits per read, for non unique reads, should be greater than 722.81 (the value for the exact mapping case) because one knows that all exact hits of $\mathcal{H}_0$ are hits with three mismatches of $\mathcal{H}_3$. Unlike the results obtained for the exact mapping case, Novoalign retrieves much less hits than expected (15 on average) indicating that many hits are missed. BFAST and SSAHA2[13] report also very few hits per read leading to a high number of reads not mapped at their original

---

[12]If we omit the −N option, BWA has catastrophic results because it would only map half of the reads, and 500,000 of them would not be mapped at their original positions.

[13]With −best 0, the mean number of hits increases to 198.16 but is still low. Moreover, surprisingly, 9 689 113 reads are reported with multiple hits.

TABLE 6. MAPPING RESULTS FOR EACH SOFTWARE RUN ON THE 3 MISMATCHES HUMAN DATASET ($\mathcal{H}_3$)

Software	Non-mapped reads	Reads uniquely retrieved		Reads with multiple hits		
		Nb	Orig. pos. not retr.	Nb	Nb hits mean [sd]	Orig. pos. not retr.
BWA	49	8,496,649	0	1,503,302	1161.98 [4634.98]	0
Novoalign	47	8,699,117	202,440	1,300,836	15.12 [36.42]	463,016
Bowtie	49	8,496,649	0	1,503,302	1161.98 [4634.98]	0
BFAST	199,451	8,476,476	84,019	1,324,073	6.17 [4.92]	718,699
SSAHA2	213	8,286,416	3,085,913	1,713,371	6.81 [14.88]	1,376,222
GASSST	326,598	8,193,650	5,703	1,479,752	1139.25 [4554.11]	36 190
PerM	186,752	8,496,655	2	1,316,593	147.25 [342.7]	5

The number of unmapped reads, the number of reads mapped to a unique location and the number of reads mapped to several locations are displayed. Moreover, for the two later categories, the number of reads whose original position has not been retrieved is reported. The mean number of hits per reads mapped more than once is also presented.

location. Bowtie, BWA, and GASSST provides a mean number of hits greater than 1100 which is more satisfactory. As already mentionned, PerM discards reads occurring 2000 or more times, which leads to less reads with multiple hits and to a smaller mean number of hits per mapped reads.

Regarding the reads uniquely retrieved by the different tools, they are less frequent than in the exact mapping case, as expected. However, Novoalign (which performed well in the exact case) returns too many unique hits, many of them at a position different from the orginal one. Unfortunately, this behavior could have a very negative impact on further analyses that only consider uniquely mapped reads. Concerning this point, BFAST is not good, SSAHA2[14] is catastrophic, Bowtie, BWA, and PerM are excellent, and GASSST provides intermediate results.

5. CONCLUSION

We performed a quantitative comparison of nine mapping tools on well-controlled benchmarks built for this purpose. We designed a first setting in which all reads do occur exactly in the reference, in one or several copies, and a second setting in which all reads occur with three or fewer mismatches. The reads were uniformly drawn from the reference genome and the original position of these reads was expected to belong to the list of hits returned by each mapping tool. A special attention has been given to reads uniquely reported because many further analyses only consider reads with a single hit.

Regarding the mapping of reads with no mismatch, we can discern three groups. BWA, Bowtie, MPscan, and GASSST belong to the first group. Programs in this group map all the reads without Ns and the original position is returned in the list (with few exceptions for GASSST). MPscan is faster than BWA, Bowtie, and GASSST, but the latter programs can be threaded. The second group contains PerM, Novoalign, and SOAP2. The first two programs do not align many reads but they retrieve the original position of the mapped reads. SOAP2 maps all the reads without Ns, but the original position of the reads, in case of multiple copies in the genome, is not always returned. The last group consists of BFAST and SSAHA2. They fail to align a large number of reads and they do not return the original position for a number of reads (the list of returned reads is incomplete). As a consequence, reads reported uniquely by these tools are not necessarily unique and could map elsewhere.

Regarding the mapping of reads with three mismatches, we can cluster the programs in four groups. The first group consists of Bowtie and BWA. These two programs map all the reads without Ns and, for the human dataset, return their original position, even for unique reads. However, we note that both tools miss the original position of a few reads for the bacterial dataset, with a slight advantage for Bowtie. For comparable results, BWA is about 40% slower than Bowtie. The next group contains only perM that, although it does not map a large number of reads, usually finds the original position of the reads. The third

---

[14]With $-$best 0, SSAHA2's results are worst because it reports only 310,674 reads as unique and half of them are not mapped at their original position

TABLE 7. MAPPING RESULTS FOR EACH PROGRAM RUN ON THE EXACT BACTERIAL DATASET ($\mathcal{B}_0$)

Software	Non-mapped reads	Reads uniquely retrieved		Reads with multiple hits		
		Nb	Orig. pos. not retr.	Nb	Nb hits mean [sd]	Orig. pos. not retr.
BWA	231	7,379,407	36	2,620,362	8.82 [39.03]	0
Novoalign	67	7,379,544	0	2,620,389	8.82 [39.03]	0
Bowtie	231	7,379,377	0	2,620,392	8.82 [39.03]	0
SOAP2	99	7,379,464	2	2,620,437	8.82 [39.03]	15
BFAST	300,194	7,379,439	60	2,320,367	3.90 [2.03]	604
SSAHA2	193	7,379,644	154	2,620,163	8.50 [31.82]	1,578
MPscan	104	7,379,461	1	2,620,435	8.82 [39.03]	19
GASSST	231	7,379,377	0	2,620,392	8.82 [39.03]	1
PerM	198	7,379,385	7	2,620,417	8.82 [39.03]	29
Reference		7,379,606		2,620,394	8.82 [39.03]	

The number of unmapped reads, the number of reads mapped to a unique location and the number of reads mapped to several locations are displayed. Moreover, for the two latter categories, the number of reads whose original position has not been retrieved is reported. The mean number of hits per reads mapped more than once is also presented.

group consists of GASSST and BFAST that do not map a large number of reads and that also do not return the original position of many reads (BFAST being much worst than GASSST in this respect). The last group consists of Novoalign and SSAHA2 that align most reads without Ns but that miss the original position of many reads (SSAHA2 being much worst than Novoalign).

It should be emphasized that the good results of BWA and, to a lesser extent, GASSST in our study rely on the fact that we did not use the default parameters, generally set to favor a rapid but incomplete search. We indeed forced both tools to perform an exhaustive search (options $-N -k\,3$ for BWA and options $-10 -s\,5$ for GASSST) despite a higher computation time. This trade-off between sensitivity and speed remains one of the difficulties for tuning the numerous parameters of such tools. In this respect, Bowtie appears to be the tool that is the most finely tuned to keep an excellent sensivity and a reasonable execution time with the default parameters.

Reads with Ns are discarded by most of the tools unless the number of allowed mismatches is at least equal to the number of Ns. Although this may be a concern, reads with Ns usually have a poor quality and most would be unalignable anyway.

We reported the computation times using a single thread in order to compare the different tools in similar settings. Of course, several tools can use multi-threads and this advantage should be taken into account for real usage. Moreover, some tools, like GASSST and Bowtie, allow to reduce the computation time by decreasing the sensitivity, ie. by reporting only a subset of hits.

In this work, we did not consider indels mainly because only four tools can cope with indels (BWA, Novoalign, BFAST, and GASSST), but clearly we will now investigate this aspect. Finally, we left the

TABLE 8. MAPPING RESULTS FOR EACH SOFTWARE RUN ON THE 3 MISMATCHES BACTERIAL DATASET ($\mathcal{B}_3$)

Software	Non-mapped reads	Reads uniquely retrieved		Reads with multiple hits		
		Nb	Orig. pos. not retr.	Nb	Nb hits mean [sd]	Orig. pos. not retr.
BWA	212	7,301,261	66	2,698,527	8.99 [40.62]	2
Novoalign	89	7,333,124	31,764	2,666,787	8.41 [36.54]	20,154
Bowtie	214	7,301,200	3	2,698,586	8.99 [40.63]	14
BFAST	172,360	7,280,690	39,839	2,546,950	4.09 [2.31]	94,442
SSAHA2	17	6,994,428	2,377,585	3,005,555	6.98 [24.32]	1,367,027
GASSST	363,984	7,025,957	17,266	2,610,059	9.03 [40.90]	8,774
PerM	185	7,301,206	8	2,698,609	8.99 [40.63]	41

The number of unmapped reads, the number of reads mapped to a unique location and the number of reads mapped to several locations are displayed. Moreover, for the two latter categories, the number of reads whose original position has not been retrieved is reported. The mean number of hits per reads mapped more than once is also presented.

evaluation of pair-end or mate-pair sequencing data—a work worth a paper *per se*—as future direction for another study.

6. AVAILABILITY

The four sets of reads ($\mathcal{H}_0$, $\mathcal{H}_3$, $\mathcal{B}_0$, and $\mathcal{B}_3$), together with both reference genomes ($\mathcal{H}_{\text{ref}}$ and $\mathcal{B}_{\text{ref}}$), are available at http://genome.jouy.inra.fr/ngs/mapping/. This website also describes the commands and all the options that have been used to run the nine mapping tools producing the results presented in this article. Our goal is to further enrich this benchmark, by adding both tools and new read sets (typically longer reads, reads with indels, and paired-end reads). Additional results will then be posted on this website.

7. APPENDIX

Here are the results when mapping the read sets $\mathcal{B}_0$ and $\mathcal{B}_3$ on the bacterial genome reference $\mathcal{B}_{\text{ref}}$. We did not report the computation times but they are smaller than in the human dataset because the genome reference is shorter and contains less repeats.

A.1. Results for $\mathcal{B}_0$

Globally on this bacterial dataset, all mapping tools give very similar and good results (Table 7), except BFAST which fails to map 300,194 reads and returns on average few hits per reads. BWA, Bowtie and GASSST mapped all the reads without any Ns but BWA missed the original position for 36 mapped reads; GASSST could not retrieve the original position for only one read, whereas Bowtie succeeded for all the mapped reads. Novoalign, SOAP2, SSAHA2, MPscan and PerM map more reads (even some with Ns) but a few reads are not retrieved at the original position with SOAP2, PerM and MPscan. In this matter, SSAHA2 is the worst program for this dataset. Compared to the human dataset, BWA, SOAP2 and MPscan display here some minor difficulties. Finally, Novoalign seems to give the best results, but the 67 unmapped reads do not contain any Ns.

A.2. Results for $\mathcal{B}_3$

As for the human dataset $\mathcal{H}_3$, SSAHA2 shows very bad results: whatever the $-$best option, it misses the original position of about 3.7 millions reads (Table 8). BFAST fares only slightly better: it does not map 172,360 reads and among the mapped reads, 134,281 are not retrieved at their original location. Novoalign does better by failing to map only 89 reads, but the number of reads not retrieved at their original position is still very high (20,154). GASSST produces intermediate results: it fails to map a very large number of reads (363,984) but, comparatively to the above mentioned tools, the number of reads for which it cannot retrieve the original position is smaller (8774). Finally, Bowtie, BWA and PerM perform very well: respectively only 214, 212, and 185 reads are not mapped (note that $\mathcal{B}_3$ contains 219 reads with Ns) but these tools miss the original position of several reads (17 for Bowtie, 68 for BWA, and 36 for PerM). Contrarily to our expectation based on the test with the human dataset, the best tools failed to retrieve all the hits of all the reads, even those with no N.

ACKNOWLEDGMENTS

DISCLOSURE STATEMENT

No competing financial interests exist.

REFERENCES

Chen, Y., Souaiaia, T., and Chen, T. 2009. PerM: efficient mapping of short sequencing reads with periodic full sensitive spaced seeds. *Bioinformatics* 5, 2514–2521.

David, M., Dzamba, M., Lister, D., et al. 2011. SHRiMP2: sensitive yet practical short read mapping. *Bioinformatics* 27, 1011–1012.

Ferragina, P. and Manzini, G. 2000. Opportunistic data structures with applications. *Proc. 41st Symp. Found. Comput. Sci.* 390–398.

Homer, N., Merriman B., and Nelson SF. 2009. BFAST: an alignment tool for large scale genome resequencing *PLoS ONE* 4, e7767.

Jiang, H., and Wong, WH. 2008. SeqMap: mapping massive amount of oligonucleotides to the genome. *Bioinformatics* 24, 2395–2396.

Kersey, P., Bower, L., Morris, L., et al. 2005. Integr8 and genome reviews: integrated views of complete genomes and proteomes. *Nucleic Acids Res.* 33, D297–D302.

Langmead, B., Trapnell, C., Pop, M., et al. 2009. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol.* 10, R25.

Li, H., and Durbin, R. 2009. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics* 25, 1754–1760.

Li, R., Li, Y., Kristiansen, K., et al. 2008. SOAP: short oligonucleotide alignment program. *Bioinformatics* 24, 713–714.

Li, R., Yu, C., Li, Y., et al. 2009. SOAP2: an improved ultrafast tool for short read alignment. *Bioinformatics* 25, 1966–1967.

Li,H., Ruan, J., and Durbin, R. 2008. Mapping short DNA sequencing reads and calling variants using mapping quality scores *Genome Res.* 18, 1851–1858.

Lin, H., Zhang, Z., Zhang, M.Q., et al. 2008. ZOOM! Zillions of oligos mapped. *Bioinformatics* 24, 2431–2437.

Lunter, G., and Goodson, M. 2010. Stampy: a statistical algorithm for sensitive and fast mapping of Illumina sequence reads *Genome Res.* 21, 936–939.

Ning, Z., Cox, AJ., and Mullikin, J.C. 2001. SSAHA: a fast search method for large DNA databases *Genome Res.* 11, 1725–1729.

Novocraft. 2010. Available at: www.novocraft.com. Accessed February 25, 2012.

Rivals, E., Salmela, L., Kiiskinen, P., et al. 2009. MPscan: fast localisation of multiple reads in genomes. *Lect. Notes Comput. Sci.* 5724, 246–260.

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

Ruffalo, M., LaFramboise, T., and Koyutrk, M. 2011. Comparative analysis of algorithms for next-generation sequencing read alignment. *Bioinformatics* 27, 2790–2796.

Weese, D., Emde, A.-K., Rausch, T., et al. 2009. RazerS–fast read mapping with sensitivity control. *Genome Res.* 19, 1646–1654.

Address correspondence to:
Dr. Sophie Schbath
INRA
Unité Mathématique, Informatique et Génome UR1077
F-78350 Jouy-en-Josas, France

E-mail: Sophie.Schbath@jouy.inra.fr