NEW ALIGNMENT PROGRAMS HAVE BEEN RECENTLY DEVELOPED. THEIR MAIN REFERENCE GENOME. TO ENABLE EFFICIENT, FAST AND ACCURATE MAPPING, SINGLE RUN. A FIRST STEP IS GENERALLY TO MAP THESE SHORT READS OVER A WIDE RANGE OF APPLICATIONS, INCLUDING GENOME RESEQUENCING OR POLYMORPHISM DISCOVERY. A VERY LARGE AMOUNT OF SHORT SEQUENCES ARE GENERATED BY THESE NEW TECHNOLOGIES. FOR EXAMPLE, THE ILLUMINA-SEQUA EXA SYSTEM CAN PRODUCE OVER 50 MILLION 32-100 BP READS IN A SINGLE RUN. A FIRST STEP IS GENERALLY TO MAP THESE SHORT READS OVER A REFERENCE GENOME. TO ENABLE EFFICIENT, FAST AND ACCURATE MAPPING, NEW ALIGNMENT PROGRAMS HAVE BEEN RECENTLY DEVELOPED. THEIR MAIN GOALS ARE TO GLOBALLY ALIGN SHORT SEQUENCES TO LOCAL REGIONS OF COMPLETE GENOMES IN A VERY SHORT TIME. FURTHERMORE, TO INCREASE SENSITIVITY, A FEW ALIGNMENT ERRORS ARE PERMITTED.

THE SEED AND EXTEND TECHNIQUE IS MOSTLY USED FOR THIS PURPOSE. THE UNDERLYING IDEA IS THAT SIGNIFICANT ALIGNMENTS INCLUDE REGIONS HAVING EXACT MATCHES BETWEEN TWO SEQUENCES. FOR EXAMPLE, ANY 50 BP READ ALIGNMENTS WITH UP TO THREE ERRORS CONTAINS AT LEAST 12 IDENTICAL CONSECUTIVE BASES. THE LAST METHOD, USED IN CLOUDBURST, INDEXES BOTH THE GENOME AND THE READS. ALTHOUGH MORE MEMORY IS NEEDED, THE ALGORITHM EXHIBITS BETTER PERFORMANCE DUE TO MEMORY CACHE LOCALITY. ANOTHER SHORT READ ALIGNMENT TECHNIQUE, USED IN BOWTIE (LANGMEAD ET AL., 2009), SOAPV2 (LI ET AL., 2009) AND BWA (LI AND DURBIN, 2009), USES A METHOD CALLED BACKWARD SEED AND EXTEND TO SEARCH AN INDEX BASED ON THE BARRROWS–WHEELER TRANSFORM (BWT, BARROWS AND WHEELER, 1994). BASICALLY, IT ALLOWS MATCHES TO BE FOUND BEFORE USING A BACKTRACKING PROCEDURE THAT ALLOWS THE ADDITION OF SOME ERRORS. ALTHOUGH THIS TECHNIQUE REPORTS EXTREMELY FAST RUNNING TIMES AND SMALL MEMORY FOOTPRINTS, SOME DATA CONFIGURATIONS LEAD TO POOR PERFORMANCE.

Moreover, in order to speed-up computations, some methods restrict the type or the number of errors per alignment to a few mismatch and indel errors. In the building alignment process, computing the number of mismatches requires linear time, whereas indel errors require more costly algorithms such as the dynamic programming techniques used in the Smith–Waterman (Smith and Waterman, 1981) or Needleman–Wunsch (NW; Needleman and Wunsch, 1970) algorithms. For instance, MAQ, Eland and Bowtie do not allow gaps. EMBF (WENDI ET AL., 2009), SOAPV1 AND SOAPV2 ALLOW ONLY ONE CONSECUTIVE GAP, WHILE PASS, SHRIMP, BFAST AND SeqMap (Jiang and Wong, 2008) ALLOW ANY COMBINATION OF MISMATCH AND INDEL ERRORS. GASSST, AS WELL, CONSIDERS ANY COMBINATION OF MISMATCH, INSERTION OR DELETION ERRORS. IN MOST APPLICATIONS, WHEN READS ARE VERY SHORT, DEALING WITH A RESTRICTED
number of errors is acceptable. On the other hand, when longer reads are processed, or when more distant reference genomes are compared, this restriction may greatly affect the quality of the search.

In this article, we introduce GASSST, a new short read aligner for mapping reads with mismatch and indel errors at a very high speed. We show how a series of carefully designed filters allows false positive positions to be quickly discarded before the refinement extension step. In particular, GASSST is compared with similar state-of-the-art programs: BFAST, BWA, PASS and SSAHA2.

2 APPROACH

GASSST uses the seed and extend strategy and indexes the genome. The seed step provides all potentially homologous areas in the genome with a given query sequence. The traditional way to accomplish this step is through a hash-table of $\text{mers}$ for selecting regions sharing common $\text{mers}$ with query sequences. To include gaps, the extend step is carried out with a dynamic programming algorithm (NW). This approach provides a high degree of accuracy, but is prohibitively expensive due to the high number of costly NW extensions that must be performed.

To tackle this issue, several approaches have been proposed. The SHRMP implementation (Rumble et al., 2009) provides an improved seed step with spaced seeds and $Q$-gram filters to restrict the size of the candidate hit space. Then, a carefully optimized vectorized Smith–Waterman extension is run to perform the extend step. BFAST (Homer et al., 2009) uses long spaced seeds to limit the amount of candidate locations, yet still achieves high sensitivity through the use of multiple indexes. PASS (Campagna et al., 2009) implements another solution: a filter is introduced before the full extend step to rule out areas that have too many differences with the query sequence. A precomputed table of all possible short words, aligned against each other, is built to perform a quick analysis of the flanking regions adjacent to seed words.

The GASSST strategy is similar: a traditional anchoring method is used followed by a NW extension step. However, fast computation is achieved thanks to a very efficient filtering step. Candidate positions are selected through a new carefully designed series of filters of increasing complexity and efficiency. Two main types of filter are used. One is related to the computation of an Euler distance between nucleotide frequency vectors as defined by Wendi et al. (2009). The idea is this: if one sequence has, for example, three more ‘T’ nucleotides than another, then the alignment will have at least three errors (mismatches or indels).

The other includes precomputed tables, as in PASS, to produce a score based on the NW algorithm, but brings the strategy to a higher level; instead of addressing large tables, as PASS does, we designed an algorithm able to reuse small tables along the whole query sequence. In this way, the filter is much more selective and discards a very large number of false positives, thus drastically decreasing the time spent in the final extend step.

GASSST’s originality comes then from the use of a small lookup table. More precisely, the precomputed alignment scores of all possible pairs of words of length $w$ can be stored in a memory of size $4^w \times w$ bytes, if a score is memoized in a single byte. For PASS, the size of the lookup table is $4^{14} \times 256$ MB. This fits into any computer’s main memory, but not in the first-level CPU cache. Hence, random accessing of the table, even if it avoids many computations, may still be costly. On the contrary, the GASSST algorithm manipulates a small table of 64 KB which easily fits into the cache memory, providing fast filtering compared with non-precomputed calculations.

3 METHODS

GASSST algorithm has three stages: (i) searching for exact matching seeds between the reference genome and the query sequences; (ii) quickly eliminating hits that have more than a user-specified number of errors; and (iii) computing the full gap aligned with the NW algorithm. These three steps are referred to, respectively, as seed-filter-extend. The novelty of the GASSST approach relies on a new highly efficient filter method.

3.1 Seed

GASSST creates an index of all possible $\text{mers}$ in the reference sequence and then scans every query sequence to find matching seeds. When seeds longer than 14 are selected the algorithm uses a simple hash-table mechanism. For each seed position in the reference sequence, the index contains the sequence number, the position where it occurs and the flanking regions (binary encoded). Up to 16 nt are stored on the left and on the right of the seed in order to speed-up the next filtering steps. The size of the index is equal to $16 \times N$ bytes, with $N$ the size of the reference sequence. If large reference sequences which exceed the memory size are considered, a simple partitioning scheme is provided; the reference sequence is split into as many parts as necessary to fit in the available RAM. Each part is then processed iteratively.

3.2 Tiled-NW filter

A lookup table, called pre-computed score table $PST$, containing all the NW alignment scores of all possible pairs of $l$ nt long words is first computed. PASS uses a PST to analyze discrepancies near the 7 nt long flanking region adjacent to the seeds. The bigger the table, the better the filtering. But unfortunately, the table grows exponentially with $l$. To address this issue, GASSST analyzes discrepancies in a region of any length thanks to the reuse of a small PST$ l$ table. The goal is to provide a lower bound approximation of the real NW score along the whole alignment. If estimated lower bounds are greater than the maximum number of allowed errors, then alignments are eliminated. If not, alignments are passed to the next step.

The following values are used for the NW score computation: 0 for a match, and 1 for mismatch and indel errors. Consequently, the final score indicates the number of discrepancies in the alignment.

In the following, only the right side of the seed is considered, the other side being symmetrical.

**Definition 1.** We call $\text{TNW}(n)$ the score of a region of $n$ nucleotides which are adjacent to the right of the seed and which are computed with a PST$ l$. $\text{TNW}$ stands for Tiled NW. $\text{TNW}(n)$ operates on the maximum length available, i.e. from the right of the seed to the end of the query sequence.

**Definition 2.** If $S1$ and $S2$ are the two sequences directly adjacent to the right of the seed, we call $\text{PST}(s, j)$ the pre-computed NW score for the two $l$ nt long words $S1(i, i+l-1)$ and $S2(j, j+l-1)$.

If $G_a$ is the maximum number of allowed gaps, $\text{TNW}(n)$ is computed with the following recursion:

\[
\text{If } n \leq j \text{ then } \quad \text{TNW}(n) = \text{PST}(1, 1) \quad (1)
\]

\[
\text{Else } \quad \text{TNW}(n) = \min(A, B, C) \quad (2)
\]

With

\[
A = \text{TNW}(n-1) + \text{PST}(n-l+1, n-l+1) - \text{PST}(n-1, n-1)
\]

\[
B = \min_{1 \leq i \leq N} (\max(s, \text{TNW}(n-1)) + \text{PST}(n-l+1, n-l+1 - s))
\]
G.Rizk and D.Lavenier

A Traditional banded semi-global alignment

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| A | A | C | G | A | T | A |
| A | 1 | 2 | 1 | 1 | 1 | 1 |
| C | 1 | 1 | 1 | 1 | 1 | 1 |
| G | 1 | 1 | 1 | 1 | 1 | 1 |
| T | 1 | 1 | 1 | 1 | 1 | 1 |
| A | 1 | 1 | 1 | 1 | 1 | 1 |
| T | 1 | 1 | 1 | 1 | 1 | 1 |
| A | 1 | 1 | 1 | 1 | 1 | 1 |
| G | 1 | 1 | 1 | 1 | 1 | 1 |
| C | 1 | 1 | 1 | 1 | 1 | 1 |

Alignment result: 2 errors

B New Tiled Algorithm

Fig. 1. Computation of a semi-global alignment (only the query sequence needs to be globally aligned), in the case of a maximum of 1 error. (A) Dynamic Programming. With 12 nt long sequences and a traditional dynamic programming algorithm, cell calculations can be limited in a band, but there are still 34 cells to compute. (B) Tiled Algorithm. With a precomputed table score of size 4 x 4, the tiled algorithm is performed in only three steps. The first step requires one table access, while the second and third steps both require three table accesses. Here, the score given by the tiled algorithm is the same as the full dynamic programming algorithm—there are two errors in the alignment. In the general case, the tiled algorithm gives a lower bound of the number of errors present in the alignment.

For a sequence \( S = a_1a_2...a_p \) of characters in the alphabet \( \Sigma = \{a_1, a_2, ..., a_k\} \), the frequency vector \( F = \{f_1, f_2, ..., f_k\} \) is defined as:

\[
\forall c \in [1:n], f_c = \sum_{i \in S} a_i, c
\]

With

\[
b_{a_i, c} = \begin{cases} 
1 & \text{if } a_i = c \\
0 & \text{otherwise}
\end{cases}
\]

The Euler distance has to be computed on similar frequency vectors, i.e. referring to sequences of equal length. The distance is computed with:

\[
ECD(F, G) = \sum_{i \leq Gm} \frac{|a_i - b_i|}{2}
\]

In GASSST, frequency vectors of sequences of up to 16 nt on both sides of the seed are computed. Actually, this value is often limited by the length of the reads, and forbids vector of the reference sequence to be computed at runtime. Indeed, the size of subsequences depends on the size available on the read, which is often less than 16 and which is only known at compute time. The Euler distance is computed between the frequency vectors of the read and the genome. If the result is greater than the maximum number of errors then the alignment is discarded.

The computation of the frequency vectors is vectorized and quickly performed thanks to the binary format of the seed flanking regions. The frequency vectors and the Euler distance computations are both vectorized to benefit from vector execution units present in modern processors. Since a 16nt sequence is 32-bit encoded, counting the frequency of a single nucleotide can be done with bit-level logical operations that operate on traditional 32-bit words. The vectorization consists of simultaneously computing values for the four nucleotides of the alphabet and operates on 128-bit wide words. The frequency distance vectorized filter is referred to as FD-vec.

The goal of the filter step is to eliminate as many false positive alignments generated by the seed step as possible, and in the fastest possible way. This is done by ordering the individual filters from the fastest to the most complex and powerful. Algorithm 1 shows the main computation loop of GASSST with the series of filters. In GASSST, \( TNW(4) \) is first since it is the fastest. It is followed by the vectorized frequency filter that rules out remaining false positive alignments. Then a more thorough filter is used \( TNW(16) \). The last filter is a \( TNW(\text{full}) \) filter which is applied on the full length of the sequence and with a \( PST_4 \) in order to eliminate a maximum number of false positive alignments. It is computationally intensive, but it comes at the end when most alignments have already been ruled out.
Finally, the true NW alignment is computed on alignments that go through all filters. This combination of filters ensures an efficient and fast filtering in a wide range of configurations, from short to longer reads, with low or high polymorphism.

One important point is that these filters only discard alignments which are proven to have too many errors and that would have been eliminated by the NW algorithm. They never eliminate good alignments, hence they do not decrease sensitivity.

Moreover, to reduce running time, the search is stopped when a maximum number of occurrences of a seed in the reference sequence has been reached. This kind of limitation is present in most other aligners. The only difference here is that the seed is also checked in some stage of the filtering process. The threshold is automatically computed according to seed length and reference sequence size. Users can control the speed/sensitivity trade-off of this heuristic through a parameter $\alpha$ in $0.5$ which modulates this threshold.

**Algorithm 1 Main computation loop**

1. Input:
   1. reference genome, short query sequences.
   2. parameter: $\alpha$ (maximum errors allowed).
2. Pre-calculation:
   1. Compute the reference genome index
   2. for each query $q$ do
   3. for each overlapping seed $s$ in $q$ do
   4. for each occurrence $o$ of $s$ in reference genome $g$ do
   5. if $TNW(q,s,o) < \alpha$ then
   6. if $FD-vec(q,s,o) < \alpha$ then
   7. if $TNW(\text{full}(q,s,o) < \alpha$ then
   8. if $NW(q,s,o) < \alpha$ then
   9. Print Alignment $(q,s,o)$

3.5 Extend

The extend step receives alignments that passed the filter step. It is computed using a traditional banded NW algorithm. Significant alignments are then printed with their full description. It should be noted that if the filter step provides good efficiency, no optimization of the extend step is required. Indeed, if most false positive alignments have already been ruled out, the extend step should only take a negligible fraction of the total execution time.

4 RESULTS

4.1 Implementation

GASSST is implemented in C++ and runs on Linux, OS X and Windows. It benefits from vector execution units with the use of SSE (streaming SIMD extensions) instructions, and is multi-threaded. It performs accurate and fast gapped alignments and allows the user to specify the number of hits given per read (option -h). A tool is provided to convert GASSST results to SAM format and compute GASSST mapping quality. GASSST is distributed under the CeCILL software license (http://www.cecill.info). Documentations and source code are freely available from the following web site: http://www.irisa.fr/symbiose/projects/gassst/

4.2 Evaluated programs

The performance of GASSST is compared with four other programs: PASS 0.74 (Campagna et al., 2009) BFAST 0.6.3c (Homer et al., 2009), BWA 0.5.7 (Li and Durbin, 2009) and SSAHA 2.5.2 (Ning et al., 2001). PASS indexes the genome and scans reads. It uses a precomputed NW table to filter alignments before conducting the extension with a dynamic programming algorithm. BFAST is currently one of the most popular tools. It relies on large spaced seeds for a fast execution, and on many different indexes for sensitivity.

BWA uses another approach, based on the BWT, and is probably one of the fastest aligners to date for alignments with a low error rate. We also tested the BWA-SW variant intended to work best for longer reads. In the following, ‘BWA’ refers to the BWA short read mode and ‘BWA-SW’ to the long read variant.

The computer used for the tests is an Intel Xeon E5462 with 32 GB RAM running at 2.8 GHz. Although all programs tested are able to benefit from multi-threaded computations, we choose to compare performance on a single thread, as it is enough to assess their respective strong or weak points.

We ran experiments with real datasets to give an indicative behavior. Detailed program analysis was conducted on simulated data where alignment correctness could be assessed.

4.3 Evaluation on real data

Performance was evaluated on three real datasets of short reads obtained from the NCBI Short Reads Trace Archive. The three sets contain, respectively, 11.9 million sequences of 36 bases, 6.8 million sequences of 50 bases and 8.5 million sequences of 76 bases of accession numbers SRR002320, SRR039633 and SRR017179. They are all aligned with the whole human genome.

BWA short read aligner was run with default options, BFAST was run with its 10 recommended indexes and default options, GASSST and PASS were set to search for alignments with at most 10% errors. We measured the execution time and the percentage of mapped reads having a mapping quality greater than or equal to 20. The results are presented in Table 1.

Evaluation on real data is difficult since true alignment locations are unknown. However, it is possible to compare results of different aligners to estimate accuracy, as it is done by Li and Durbin (2010) for their evaluation of BWA-SW. If an aligner $A$ gives a high mapping quality to a read and another aligner $B$ finds an alignment at another position for that same read with an alignment score better or just slightly worse, then $A$ alignment is probably wrong. A score for each read is computed as the number of matches minus three multiplied by the number of differences (mismatches and gaps). We say that $A$ alignment is questionable if the score derived from $A$ minus the score derived from $B$ is less than 20. Since this evaluation method is approximate, a full evaluation was conducted on simulated data in Section 4.5.

On short 36bp reads, GASSST performance is comparable to BWA. On longer 76 bp reads both PASS anf BFAST becomes very slow. On the other hand, the GASSST combination of filters still works well for the 76 bp dataset.

GASSST and PASS currently cannot store the index on disk, yet index computation time is amortized when working on very large sets of reads. BFAST index time is very long because it uses 10 different indexes, however they are computed only once.

4.4 Filter behavior analysis

We measured the filtering power of the filter combination on the same three datasets of the previous section to validate their efficiency on different configurations. The allowed similarity rate was set to 90% minimum so the number of allowed errors was 3, 4 and 7,
Table 1. Evaluation on real data

| Read length | Software | Index (s) | Align (s) | Q20% | Q20 Error rate (%) |
|-------------|----------|-----------|-----------|------|-------------------|
| 36 bp       | GASSST   | 1712      | 3211      | 34.5 | 0.14              |
|             | BFAST    | 520/800   | 17,520    | 41.8 | 0.12              |
|             | BWA      | 5158      | 3739      | 35.4 | 0.17              |
|             | PASS     | 2312      | 5072      | 41.4 | –                 |
| 50 bp       | GASSST   | 1719      | 4090      | 73.7 | 0.04              |
|             | BFAST    | 520/800   | 22,799    | 80.4 | 0.10              |
|             | BWA      | 5158      | 3043      | 74.5 | 0.17              |
|             | PASS     | 2144      | 5384      | 79.3 | –                 |
| 76 bp       | GASSST   | 1701      | 8483      | 81.3 | 0.04              |
|             | BFAST    | 520/800   | 161,220   | 85.4 | 0.28              |
|             | BWA      | 5158      | 3101      | 86.4 | 0.53              |
|             | PASS     | 1951      | 118,541   | 87.7 | –                 |

Datasets consisted, respectively, of 11.9, 6.8 and 8.5 millions of reads of size 36, 50 and 76 bp, of accession numbers SRR002320, SRR039633 and SRR017179. The three datasets were aligned with the whole human genome. The time required to compute the genome index is shown in column 3. For BFAST and BWA, the index was computed only once and stored on disk. Column 4 shows the time required to align reads, running on a single core of a 2.8 GHz Xeon E5642. Column 5 shows the percentage of reads with a unique best alignment, and error rate is not computed.

Second, one interesting thing to observe is the number of false positive alignments that passed the filters and still arrive at the NW step. The NW alignment is seen as the last filter and its result is also included. Table 2 shows the results. The first thing that can be noted is that in all cases the percentage of hits filtered at the NW step is limited to 0.2% of the number of hits generated by the seed phase. This means that without filters, this step would take at least 500 times longer.

One interesting thing to observe is the number of false positive alignments that passed the filters and still arrive at the NW step. The NW alignment is seen as the last filter and its result is also included. Table 2 shows the results. The first thing that can be noted is that in all cases the percentage of hits filtered at the NW step is limited to 0.2% of the number of hits generated by the seed phase. This means that without filters, this step would take at least 500 times longer.

To complete this analysis, we measured the total time spent inside each filter. The program was profiled for a typical execution with the 50 bp dataset. For each step, both the computation time of a single hit in nanoseconds and the total execution time of the step in percentage of the total time of the program were measured. The results are shown in Table 3. The General Overhead mostly represents the cost of line 8 of Algorithm 1, which iterates through hits and loads the associated information. The TNW(4) filter takes only 5.8 ns, which represents, as expected, only a few clock cycles of the processor.

### Table 2. Filtering rate on three datasets

| Data Set | Filter Step | Step filter (%) | Total filter (%) |
|----------|-------------|-----------------|------------------|
| 36 bp    | TNW(4)      | 64.9            | 64.9             |
|          | FD−vec      | 72.8            | 90.5             |
|          | TNW(16)     | 96.6            | 99.7             |
|          | TNW(4)(full)| 85.9            | 99.8             |
|          | NW          | 63.8            | 99.9             |
| 50 bp    | TNW(4)      | 41.7            | 41.7             |
|          | FD−vec      | 80.4            | 88.6             |
|          | TNW(16)     | 97.5            | 99.7             |
|          | TNW(4)(full)| 88.5            | 99.97            |
|          | NW          | 67.5            | 99.99            |
| 76 bp    | TNW(4)      | 0.1             | 0.1              |
|          | FD−vec      | 49.8            | 49.9             |
|          | TNW(16)     | 93.7            | 96.8             |
|          | TNW(4)(full)| 94.8            | 99.8             |
|          | NW          | 70.2            | 99.95            |

The first column gives the percentage of hits filtered by a given filter. The second column gives the total percentage of hits filtered by the sequence of filters. Reads were aligned with 10% errors maximum, which means a maximum of 3, 4 and 7 errors, gaps or mismatches, respectively.

### Table 3. Filters execution time

| Step       | Execution time for one hit (ns) | Total percent time spent in this step |
|------------|---------------------------------|--------------------------------------|
| General Overhead | 13.9                       | 33                                   |
| TNW(4)     | 5.8                            | 14                                   |
| FD−vec     | 22.4                           | 32                                   |
| TNW(16)    | 78.5                           | 15                                   |
| TNW(4)(full)| 356.7                       | 3                                   |
| NW         | 4480.5                         | 3                                   |

Average time taken by each filter step for one hit of a 50bp query, in nanoseconds. Note that only the times of TNW(4)(full) and NW actually depends on query size since the others only work on a bounded size subsequence.

We can also verify that the filters are indeed ordered from the most general to the most specific. The NW step takes only 3% of the total time, which validates the assessment formulated in Section 3.5 that we do not need to be concerned about its optimization.

### 4.5 Evaluation on simulated data

We simulated 12 datasets containing one million reads each from the entire human genome, with lengths of 50, 100, 200 and 500 bp and with error rates of 2, 5 and 10%. The error rate is the probability of each base being an error, 20% of errors are indel errors with indel length / drawn from a geometric distribution of density 0.7 × 0.3^−1. BFAST is run with options −K 8 and −M 1280 with 10 indexes. GASSST was tested in two different configurations, a fast one with option −r0 and an accurate one with −r3, which controls the speed/sensitivity trade-off of the algorithm by setting a maximum number of occurrences per seed. It is roughly the equivalent of options −K and −M of BFAST. Other GASSST options such as seed length and maximum number of allowed errors were tuned accordingly to the different datasets. BWA and PASS options were also tuned to allow more mismatches and gaps when necessary, and BWA-SW was run with default configuration. Finding the most
Although this choice penalizes BWA, which is the only one not A read is considered to be mapped if it has a unique best alignment.

Table 4. Evaluation on simulated data

| Program | Mode | Metrics | 50bp | 100bp | 200bp | 500bp |
|---------|------|---------|------|-------|-------|-------|
|         |      |         | 2%   | 5%    | 10%   | 2%   | 5%    | 10%   | 2%   | 5%    | 10%   |
| GASSST  | fast | Align sec | 584  | 781   | 1720  | 794  | 981   | 1160  | 2030 | 2314  | 3051  |
|         |      | Accuracy | 99.2 | 98.5  | 93.8  | 90.5 | 89.4  | 86.9  | 91.9 | 91.5  | 89.7  |
| GASSST  | accurate | Align sec | 1709 | 2741  | 4706  | 1290 | 1887  | 3262  | 3452 | 6173  | 8744  |
|         |      | Accuracy | 96.7 | 44.6  | 37.5  | 51.0 | 50.3  | 43.5  | 53.4 | 52.8  | 51.7  |
| BFAST   | Align sec | 99.8  | 99.3  | 93.5  | 99.7  | 99.4  | 97.5  | 99.9  | 99.8  | 99.3  | 99.9  |
| BWA     | Align sec | 792   | 1392  | 1572  | 1862 | 2491  | 3364  | 4660 | 2145  | 185  |
| BWA-SW  | Align sec | 2279  | 2044  | 1756  | 1526 | 1578  | 11452 | 15285 | 2275 | 3710  | 24290 |
| SSAHA2  | Align sec | 1709 | 2741  | 4706  | 1290 | 1887  | 3262  | 3452 | 6173  | 8744  |
| PASS    | Align sec | 2012  | 2281  | 5085  | 14387 | 26033  | 30122 | 103138 | 139436 | 180943 |

Data sets containing one million reads each are simulated from the human genome with different lengths and error rate. Twenty percent of errors are indel errors with indel length drawn from a geometric distribution of density \(0.7 \cdot 0.3^{-l}\). The alignment time in seconds only includes the fraction of the total time proportional to the number of reads, i.e., not the time spent in computing or loading the index of the human genome, running on a single core of a 2.8 GHz Xeon E4562. Computed alignment coordinates are compared to the true simulated coordinates to find sensitivity and accuracy. Sensitivity is the percentage of reads correctly mapped, while accuracy is the percentage of mapped reads that are correctly mapped. A read is considered mapped if it has a unique best alignment. A mapped read is considered correct if it is within 10 bp of its true coordinates. We filtered alignments having a mapping quality less than 20, except for PASS, which does not give mapping quality.

Table S1 gives the list of options parameters used.

For each run, we reported accuracy, sensitivity and running time. A read is considered to be mapped if it has a unique best alignment. A mapped read is considered correct if it is within 10 bp of its true coordinates. We filtered all alignments that had a mapping quality less than 20, except for PASS which does not compute mapping quality. Sensitivity and accuracy are then, respectively, defined as the percentage of total or mapped reads that are correct. The running time reported does not include index loading or computing phase.

Tests were also conducted with SHRiMP 1.3.1 but not included here since running times were tens to hundreds of times larger than GASSST, rendering evaluation impractical. This corresponds to observations in other studies that had to extrapolate SHRiMP running time (Homer et al., 2009).

For datasets with a low 2% error rate, GASSST performs comparable with BWA. For example, on the 200 bp 10% error rate dataset, GASSST obtained results comparable to BWA-SW. With 10% error rate GASSST is eight times faster than SSAHA2 for similar results, whereas BWA-SW accuracy is dropping. BFAST is efficient on short 50 bp reads only, its execution time increases a lot for longer reads.

In this article, we introduced an original method to speed-up aligner programs. While the BFAST approach is to reduce the number of

5 DISCUSSION

In this article, we introduced an original method to speed-up aligner programs.
candidate alignment locations generated in the seed step through the use of large spaced seeds, our approach uses a simple indexing scheme generating many candidate alignment locations, but quickly discards most of them through a series of filters.

Experiments conducted on simulated data showed that the GASSST approach provides fast and high quality results, and that this new series of filters is indeed more efficient than trying to optimize the NW algorithm further. Even a GPU-based implementation of NW, with an optimistic 20-fold speed-up over the SSE vectorized implementation of SHRiMP, would still be slower than our filtering approach. Moreover, since sequencing technologies are evolving toward longer and longer generated reads, many algorithms designed to work best on short reads will become inefficient. On the other hand, our experiments showed that GASSST is already prepared for this evolution: it is efficient on short 50 bp reads as well as on long 500 bp reads.

Table 3 shows that the General Overhead, the cost of iterating through hits, can represent up to a third of the total execution time. This reveals that for the simple indexing scheme we use our filtering technique is close to the optimal solution possible, in the sense that further improvements of filters would not bring much overall speed-up. However, combining state-of-the-art indexing techniques with our series of filters should achieve even better performance. We could, for example, replace our simple unique index with the multiple indexing technique used by BFAST. It would reduce the number of hits to explore, probably increasing performance even further.

GASSST currently uses a simple gap and mismatch scoring scheme, whereas affine gap penalties are sometimes necessary. It could very easily be integrated in the extend step. Yet if the filter and extend step uses different scoring schemes, filters will no longer be guaranteed to discard only false positive alignments. This could be a problem that needs to be investigated. Another solution would be to also include affine gap penalties in the filter step. Although apparently problematic for our tiled NW algorithm, approximate solutions might be designed and will be the focus of future work.

The experiments conducted demonstrate the efficiency of our new filtering approach. This new approach is compatible with all algorithms based on the seed-filter-extend strategy, so other algorithms should be able to use it resulting in a significant performance increase without any decrease in sensitivity.

**ACKNOWLEDGEMENTS**

We are grateful to Jean-Jacques Codani for testing and providing helpful feedback about the software. We also thank Laurent Noel and Matthias Gallé for proofreading the manuscript.

**Funding:** French Agence Nationale de la Recherche BioWIC (ANR-08-SEGI-005).

**Conflict of Interest:** none declared.

**REFERENCES**

Burrows,M and Wheeler,D. (1994) A block-sorting lossless data compression algorithm. Technical report 124, Digital Equipment Corporation, Palo Alto, CA.

Campagna,D et al. (2009) PASS: a program to align short sequences. Bioinformatics, 25, 967.

Ferragina,P and Manzini,G. (2000) Opportunistic data structures with applications. In Proceedings of the 41st Symposium on Foundations of Computer Science (FOCS 2000), Redondo Beach, CA, USA, pp. 390-398.

Homer,N et al. (2009) Biafast: an alignment tool for large scale genome resequencing. PLoS ONE, 4, e7767.

Jiang,H and Wong,W. (2008) SeqMap: mapping massive amount of oligonucleotides to the genome. Bioinformatics, 24, 2395.

Langmead,B 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,H and Durbin,R. (2010) Fast and accurate long-read alignment with Burrows-Wheeler transform. Bioinformatics, 26, 589.

Li,H et al. (2008) Mapping short DNA sequencing reads and calling variants using mapping quality scores. Genome Res., 18, 1851.

Li,R et al. (2008) SOAP: short oligonucleotide alignment program. Bioinformatics, 24, 713.

Li,R et al. (2009) SOAP2: an improved ultrafast tool for short read alignment. Bioinformatics, 25, 1966-1967.

Needleman,S and Wunsch,C. (1970) A general method applicable to the search for similarities in the amino acid sequence of two proteins. J. Mol. Biol., 48, 443-453.

Ning,Z et al. (2001) SSAHA: a fast search method for large DNA databases. Genome Res., 11, 1725.

Rumble,S et al. (2009) SHRiMP: accurate mapping of short color-space reads. PLoS Comput. Biol., 5, e1000386.

Smith,T and Waterman,M. (1981) Identification of common molecular subsequences. J. Mol. Biol., 147, 195-197.

Wendle,W et al. (2009) Short read DNA fragment unchoosing algorithm. BMC Bioinformatics, 10 (Suppl. 1), S17.