BIOINFORMATICS

Vol. 00 no. 00 2009
Pages 1–7

Fast and accurate long read alignment with Burrows-Wheeler transform

Heng Li [1] and Richard Durbin [1,*]

[1]Wellcome Trust Sanger Institute, Wellcome Genome Campus, Cambridge, CB10 1SA, UK

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Associate Editor: XXXXXXX

ABSTRACT

Motivation: Many programs for aligning short sequencing reads to a reference genome have been developed in the last two years. Most of them are very efficient for short reads but inefficient or not applicable for reads longer than 200bp because the algorithms are heavily and specifically tuned for short queries with low sequencing error rate. However, some sequencing platforms already produce longer reads and others are expected to become available soon. For longer reads, hashing based software such as BLAT and SSAHA2 remain the only choices. Nonetheless, these methods are substantially slower than short-read aligners in terms of aligned bases per unit time.

Results: We designed and implemented a new algorithm, BWA Smith-Waterman alignment (BWA-SW), to align long sequences up to 1 megabases against a large sequence database (e.g. the human genome) with a few gigabytes of memory. The algorithm is as accurate as SSAHA2, more accurate than BLAT, and is several to tens of times faster than both.

Availability: http://bio-bwa.sourceforge.net

Contact: rd@sanger.ac.uk

1 INTRODUCTION

Following the development of sensitive local alignment software such as FASTA (Pearson and Lipman, 1988) and BLAST (Altschul et al., 1997) around 1990, a new generation of faster methods to find DNA sequence matches was developed since 2000, including MegaBLAST (Zhang et al., 2000; Morgulis et al., 2008), SSAHA2 (Ning et al., 2001), BLAT (Kent, 2002) and PatternHunter (Ma et al., 2002), greatly speeding up matching capillary sequencing reads against a large reference genome. When new sequencing technologies arrived that generated millions of short (under 100bp) reads, a variety of new algorithms were developed which were 10–1000 times faster, including SOAP (Li et al., 2008b), MAQ (Li et al., 2008a), Bowtie (Langmead et al., 2009) and BWA (Li and Durbin, 2009). However, Roche/454 sequencing technology has already produced reads longer than 400bp in production, Illumina gradually increases read length over 100bp, and Pacific Bioscience generates 1,000bp reads in early testing (Eid et al., 2009). Reads coming from the new sequencing technologies are not short any more, which effectively rules out many of the new aligners exclusively designed for reads no longer than 100bp. Efficiently aligning long reads against a long reference sequence like the human genome poses a new challenge to the development of alignment tools.

Long-read alignment has different objectives from short-read alignment. Firstly, in short-read alignment, we would usually like to align the full-length read to reduce the reference bias caused by the mismatches towards the ends of the read. Given this requirement, we can design spaced seed templates (Ma et al., 2002) spanning the entire read (Smith et al., 2008; Jiang and Wong, 2008; Lin et al., 2008), or quickly filter out poor matches, for example, by applying q-gram filtration (Rumble et al., 2009; Weese et al., 2009) or by bounding the search process (Li and Durbin, 2009), and thus accelerate the alignment. In long-read alignment, however, we would prefer to find local matches because a long read is more fragile to structural variations and misassemblies in the reference but is less affected by the mismatches close to the ends of a read. Secondly, many short-read aligners are only efficient when doing ungapped alignment or allowing limited gaps, e.g. a maximum of one gap. They cannot find more gaps or the performance quickly degrades when they are tuned for this task. Long-read aligners, however, must be permissive about alignment gaps because indels occur more frequently in long reads and may be the dominant source of sequencing errors for some technologies such as 454 and Pacific Bioscience.

When considering algorithms to speed up long read alignment, hash table indexing as is used in most current software is not the only choice. Meek et al. (2003) found a Smith-Waterman like dynamic programming that can be applied between a query sequence and the suffix tree of the reference, effectively aligning the query against each subsequence sampled from the suffix tree via a top-down traversal. As on a suffix tree identical sequences are collapsed on a single path, time is saved by avoiding repeated alignment of identical subsequences. Lam et al. (2008) furthered this idea by implicitly representing the suffix tree with an FM-index (Ferragina and Manzini, 2000), which is based on the Burrows-Wheeler transform (BWT) (Burrows and Wheeler, 1994), to achieve a small memory footprint. Their new algorithm, BWT-SW, is able to deliver identical results to the standard Smith-Waterman alignment, but thousands of times faster when aligning against the human genome sequence. While BWT-SW is still slower than BLAST on long query sequences, it finds all matches without heuristics. One can imagine that introducing heuristics would further accelerate BWT-SW. Our BWA-SW algorithm follows this route.

To some extent, BWA-SW, as well as BWT-SW, also follows the seed-and-extend paradigm. But different from BLAT and SSAHA2,

*to whom correspondence should be addressed

BWA-SW finds seeds by dynamic programming between two FM-indices and allows mismatches and gaps in the seeds. It extends a seed when the seed has few occurrences in the reference sequence. Speed is gained by reducing unnecessary extension for highly repetitive sequences.

In this article, we will describe this new alignment algorithm, BWA-SW, for long read alignment and evaluate its practical performance along with BLAT and SSAHA2 on both simulated and real data. We will also give a brief introduction to suffix array and FM-index, but readers are referred to Li and Durbin (2009) for more details.

2 METHODS

2.1 Overview of the BWA-SW algorithm

BWA-SW builds FM-indices for both the reference and query sequence. It implicitly represents the reference sequence in a prefix trie and represents the query sequence in a prefix directed acyclic word graph (prefix DAWG) (Blumer et al., 1985) which is transformed from the prefix trie of the query sequence (Section 2.3). A dynamic programming can be applied between the trie and the DAWG, by traversing the reference prefix trie and the query DAWG, respectively. This dynamic programming would find all local matches if no heuristics were applied, but would be no faster than BWT-SW. In BWA-SW, we apply two heuristic rules to greatly accelerate this process. Firstly, traversal on the query DAWG is carried in the outer loop, and therefore without finishing the dynamic programming, we know all the nodes in the reference prefix trie that match the query node with a positive score. Based on the observation that the true alignment tends to have a high alignment score, we can prune low-scoring matches at each node to restrict the dynamic programming around good matches only. The scale of dynamic programming can thus be dramatically reduced. It is possible for the true alignment to be pruned in this process, but in practice, this can be controlled by the use of heuristics and happens rarely, given long or high-quality query sequences. Secondly, BWA-SW only reports alignments largely non-overlapping on the query sequence instead of giving all the significant local alignments. It heuristically identifies and discards seeds contained in a longer alignment and thus saves computing time on unsuccessful seed extensions.

2.2 Notations and definitions

2.2.1 Suffix array and BWT Let $\Sigma = \{A, C, G, T\}$ be the alphabet of nucleotides and \$ be a symbol that is lexicographically smaller than all the symbols in Σ. Given a nucleotide sequence $X = a_1 \dots a_{n-1}$ with $a_{n-1} = \$$, let $X[i] = a_i$ be the i-th symbol, $X[i, j] = a_i \dots a_j$ a subsequence of X and $X_i = X[i, n-1]$ a suffix of X. The suffix array S of X is a permutation of integers $0, \dots, n-1$ such that $S(i) = j$ if and only if X_j is the i-th lexicographically smallest suffix. The BWT of X is a permutation of X, where $B[i] = \$$ if $S(i) = 0$ and $B[i] = X[S(i)-1]$ otherwise.

2.2.2 Suffix array interval Given a sequence W, the *suffix array interval* or *SA interval* $[\underline{R}(W), \overline{R}(W)]$ of W is defined as

$$\underline{R}(W) = \min\{k\text{: } W \text{ is the prefix of } X_{S(k)}\}$$
$$\overline{R}(W) = \max\{k\text{: } W \text{ is the prefix of } X_{S(k)}\}$$

In particular, if W is an empty string, $\underline{R}(W) = 1$ and $\overline{R}(W) = n-1$. The set of the positions of all the occurrences of W is $\{S(k) : \underline{R}(W) \leq k \leq \overline{R}(W)\}$.

Let $C(a) = \#\{0 \leq j \leq n-2 : X[j] < a\}$ and $O(a, i) = \#\{0 \leq j \leq i : B[j] < a\}$, where $\#\{\cdot\}$ calculates the cardinality (or size) of a set. Ferragina and Manzini (2000) proved that

$$\underline{R}(aW) = C(a) + O(a, \underline{R}(W) - 1) + 1$$
$$\overline{R}(aW) = C(a) + O(a, \overline{R}(W))$$

and that $\underline{R}(aW) \leq \overline{R}(aW)$ if and only if aW is a substring of X.

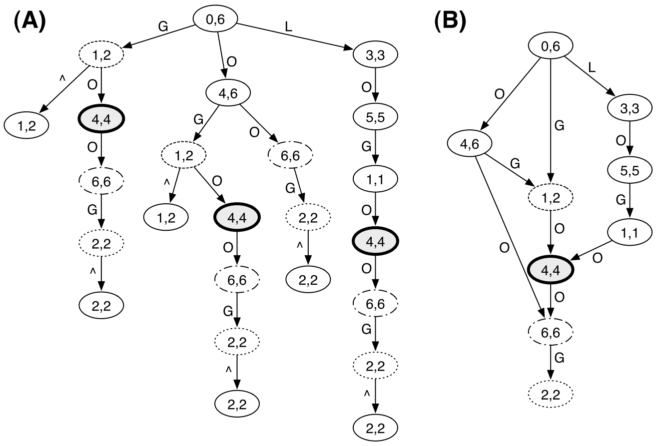

Fig. 1. Prefix trie and prefix DAWG of string '`GOOGOL`'. (A) Prefix trie. Symbol '$\wedge$' marks the start of a string. The two numbers in a node gives the SA interval of the node. (B) Prefix DAWG constructed by collapsing nodes with the identical SA interval. For example, in the prefix trie, three nodes has SA interval $[4, 4]$. Their parents have interval $[1, 2]$, $[1, 2]$ and $[1, 1]$, respectively. In the prefix DAWG, the $[4, 4]$ node thus has parents $[1, 2]$ and $[1, 1]$. Node $[4, 4]$ represents three strings '`OG`', '`OGO`' and '`OGOL`' with the first two strings being the prefix of '`OGOL`'. Figure 1A is modified from Figure 1 in Li and Durbin (2009).

2.2.3 FM-index The suffix array S, array C and O suffice for the exact search of a pattern in X. FM-index (Ferragina and Manzini, 2000) is a compressed representation of the three arrays, consisting of the compressed BWT string B, auxilliary arrays for calculating O, and part of the suffix array S. BWA-SW, however, uses a simplified FM-index where we do not compress B and store part of the occurrence array O without auxilliary data structures. The simplified version is more efficient for DNA sequences with a very small alphabet. Details on the construction are presented in our previous paper (Li and Durbin, 2009).

2.2.4 Alignment An *alignment* is a tuple (W_1, W_2, A) where W_1 and W_2 are two strings and A is a series of copying, substitution, insertion and deletion operations which transform W_2 into W_1. Insertions and deletions are *gaps*. Gaps and substitutions are *differences*. The *edit distance* of the alignment equals the total number of differences in A.

A score can be calculated for an alignment given a scoring system. We say W_1 matches W_2 if W_1 and W_2 can be aligned with a positive score, and in this case, we also say (W_1, W_2) is a match.

A match (W_1, W_2) is said to be *contained* in (W_1', W_2') on the first sequence if W_1 is a substring of W_1'. Similarly we can define the 'contained' relationship between alignments (a stronger condition) and between an alignment and a match.

2.3 Prefix trie and prefix DAWG

The *prefix trie* of string X is a tree with each edge labeled with a symbol such that the concatenation of symbols on the path from a leaf to the root gives a unique prefix of X. The concatenation of edge symbols from a node to the root is always a substring of X, called the string represented by the node. The *SA interval* of a node is defined as the SA interval of the string represented by the node. Different nodes may have an identical interval, but recalling the definition of SA interval, we know that the strings represented by these nodes must be the prefixes of the same string and have different lengths.

The *prefix directed acyclic word graph*, or *prefix DAWG*, of X is transformed from the prefix trie by collapsing nodes having an identical interval. Thus in the prefix DAWG, nodes and SA intervals have an one-to-one relationship, and a node may represent multiple substrings of X, falling in a sequence where each is a prefix of the next as is discussed in the previous paragraph. Figure 1 gives an example.

2.4 Aligning prefix trie against prefix DAWG

We construct a prefix DAWG $\mathcal{G}(W)$ for the query sequence W and a prefix trie $\mathcal{T}(X)$ for the reference X. The dynamic programming for calculating the best score between W and X is as follows. Let $G_{uv} = I_{uv} = D_{uv} = 0$ when u is the root of $\mathcal{G}(W)$ and v the root of $\mathcal{T}(X)$. At a node u in $\mathcal{G}(W)$, for each of its parent node u', calculate

$$\begin{aligned} I_{uv|u'} &= \max\{I_{u'v}, G_{u'v} - q\} - r \\ D_{uv|u'} &= \max\{D_{uv'}, G_{uv'} - q\} - r \\ G_{uv|u'} &= \max\{G_{u'v'} + S(u', u; v', v), I_{u'v'}, D_{u'v'}, 0\} \end{aligned}$$

where v' is the parent of v in $\mathcal{T}(X)$, function $S(u', u; v', v)$ gives the score between the symbol on the edge (u', u) and the one on (v', v), and q and r are gap open and gap extension penalties, respectively. G_{uv}, I_{uv} and D_{uv} are calculated with:

$$u^* = \underset{u' \in \mathrm{pre}(u)}{\mathrm{argmax}}\ G_{uv|u'}$$

$$(G_{uv}, I_{uv}, D_{uv}) = \begin{cases} (G_{uv|u^*}, I_{uv|u^*}, D_{uv|u^*}) & G_{uv|u^*} > 0 \\ (-\infty, -\infty, -\infty) & \text{otherwise} \end{cases}$$

where $\mathrm{pre}(u)$ is the set of parent nodes of u. G_{uv} equals the best score between the (possibly multiple) substrings represented by u and the (one) substring represented by v. We say a node v *matches* u if $G_{uv} > 0$.

The dynamic programming is performed by traversing both $\mathcal{G}(W)$ and $\mathcal{T}(X)$ in the reverse post-order (i.e. all parent nodes are visited before children) in a nested way. Noting that once u does not match v, u does not match any nodes descending from v, we only need to visit the nodes close to the root of $\mathcal{T}(X)$ without traversing the entire trie, which greatly reduces the number of iterations in comparison to the standard Smith-Waterman algorithm that always goes through the entire reference sequence.

2.5 Acceleration by the standard Smith-Waterman

In comparison to the standard Smith-Waterman alignment whose time complexity is $O(|X| \cdot |W|)$, BWA-SW has better time complexity since it is no slower than BWT-SW whose time complexity $O(|X|^{0.628}|W|)$ (Lam et al., 2008). This conclusion comes because for short sub-alignments we are considering multiple possible matches with a single uv comparison. However, the constant associated with each iteration is much larger due to the complex steps related to the traversal of prefix trie and prefix DAWG, which makes BWA-SW inefficient when we use BWA-SW to extend a unique alignment. A more efficient strategy would be to use BWA-SW to find partial matches and apply the Smith-Waterman algorithm to extend. In dynamic programming, we know the number of partial matches being considered at any pair because this can be calculated from the size of the SA interval. When G_{uv} is good enough and the SA interval size of v is below a certain threshold (3 by default), we save the (u, v) pair, called a *seed interval pair*, and do not go deeper from the v node in $\mathcal{T}(X)$. By looking up the suffix array of X and W, we can derive *seed matches*, or simply *seeds*, from seed interval pairs. These seeds are then extended by the Smith-Waterman algorithm later. If the entire query is a highly repetitive sequence, it will be aligned purely with the algorithm described in the last section without the Smith-Waterman extension.

Because we are stopping the dynamic programming early to generate seeds, the global best alignment may contain multiple seeds and in practice this will tend to be the case for long alignments. Typically for 1kb alignments there will be 10–20 seeds. Below we will take advantage of this observation to heuristically speed up the search.

BWT-SW deploys a similar strategy in performing the dynamic programming between a sequence and a prefix trie to find seed matches followed by Smith-Waterman extension. The main difference from our algorithm is that BWT-SW initiates the Smith-Waterman alignment once the score is high enough, regardless of the SA interval size. Sometimes a repetitive sequence may match to thousands of places in the human genome and extending partial matches each time may be slow.

2.6 Heuristic accelerations

2.6.1 Z-best strategy The algorithm described so far is exact in that it is able to deliver the same results as the Smith-Waterman algorithm. Although it is much faster than the standard algorithm given a long reference sequence, it is not fast enough for aligning large-scale sequencing data. Closer investigation reveals that even for a unique 500bp query sequence, a few million nodes in $\mathcal{T}(X)$ may match the query with a positive alignment score. The majority of these nodes are random matches or matches in short low-complexity regions. Visiting all of them is wasteful.

To accelerate alignment, we traverse $\mathcal{G}(W)$ in the outer loop and $\mathcal{T}(X)$ in the inner loop, and at each node u in $\mathcal{G}(W)$ we only keep the top Z best-scoring nodes in $\mathcal{T}(X)$ that match u, rather than keep all the matching nodes. This heuristic strategy is called *Z-best*. Of course when we apply the Z-best strategy, we could miss a seed contained in the true alignment when a false match has a higher score. But if the query is nearly identical to the reference, this happens less often. In addition, if the true alignment is long and contains many seeds, the chance of all seeds being false is very small. On both simulated and real data (see Results), we find even $Z = 1$ works well with high-quality 200bp reads ($< 5\%$ sequencing error rate). Increasing Z to 10 or higher marginally improves the accuracy but greatly reduces the alignment speed.

To reduce alignment errors, we also align the reverse query sequence to the reverse reference sequence, namely reverse-reverse alignment, in addition to the forward-forward alignment. Ideally, the forward-forward and the reverse-reverse alignments should yield identical outcomes, but if a seed in the true alignment has a low-scoring suffix (or prefix), the forward-forward (or reverse-reverse) alignment is likely to miss it, while combining the two rounds of alignment reduces the chance. Moreover, if the best alignment from the forward-forward alignment contains many seed matches, the chance of it being false is also small. In implementation, we do not apply the reverse-reverse alignment if the best alignment contains, by default, 5 or more seeds.

2.6.2 Filtering seeds before the Smith-Waterman extension Like BLAST, both BLAT and SSAHA2 report all significant alignments or typically tens of top-scoring alignments, but this is not the most desired output in read mapping. We are typically more interested in the best alignment or best few alignments, covering each region of the query sequence. For example, suppose an 1,000bp query sequence consists of a 900bp segment from one chromosome and an 100bp segment from another chromosome; 400bp out of the 900bp segment is a highly repetitive sequence. For BLAST, to know this is a chimeric read we would need to ask it to report all the alignments of the 400bp repeat, which is costly and wasteful because in general we are not interested in alignments of short repetitive sequences contained in a longer unique sequence. On this example, a useful output would be to report one alignment each for the 900bp and the 100bp segment, and to indicate if the two segments have good suboptimal alignments that may render the best alignment unreliable. Such output simplifies downstream analyses and saves time on reconstructing the detailed alignments of the repetitive sequence.

In BWA-SW, we say two alignments are *distinct* if the length of the overlapping region on the query is less than half of the length of the shorter query segment. We aim to find a set of distinct alignments which maximizes the sum of scores of each alignment in the set. This problem can be solved by dynamic programming, but as in our case a read is usually aligned entirely, a greedy approximation would work well. In the practical implementation, we sort the local alignments based on their alignment scores, scan the sorted list from the best one and keep an alignment if it is distinct from all the kept alignments with larger scores; if alignment a_2 is rejected because it is not distinctive from a_1, we regard a_2 to be a suboptimal alignment to a_1 and use this information to approximate the mapping quality (Section 2.7).

Because we only retain alignments largely non-overlapping on the query sequence, we might as well discard seeds that do not contribute to the final alignments. Detecting such seeds can be done with another heuristic before the Smith-Waterman extension and time spent on unnecessary extension can thus be saved. To identify these seeds, we chain seeds that are contained in a band (default band width 50bp). If on the query sequence a short chain is fully contained in a long chain and the number of seeds in the short chain is below one tenth of the number of seeds in the long chain, we discard all the seeds in the short chain, based on the observation that the short chain can rarely lead to a better alignment than the long chain in this case. Unlike the Z-best strategy, this heuristic does not have a noticeable effect on alignment accuracy. On 1,000 10kbp simulated data, it halves the running time with no reduction in accuracy.

2.7 Approximating mapping quality

Li et al. (2008a) introduced the concept of mapping quality to estimate the probability of a query sequence being placed at a wrong position. If an alignment algorithm guarantees to find all local alignments, mapping quality is determined by these local alignments only. However, as BWA-SW deploys heuristic rules, the chance of producing a wrong alignment is also related to the heuristics. To estimate the mapping quality of a BWA-SW alignment, we fit an empirical formula: $250 \cdot c_1 \cdot c_2 \cdot (S_1 - S_2)/S_1$, where S_1 is the score of the best alignment, S_2 the score of the 2nd best alignment, c_1 equals 1 if the alignment covers more than four seeds or 0.5 otherwise, and c_2 equals to 1 if the best alignment is found by both forward-forward and reverse-reverse alignment or 0.2 otherwise.

3 RESULTS

3.1 Implementation

The BWA-SW algorithm is implemented as a component of the BWA program (Li and Durbin, 2009) which is distributed under the GNU general public license (GPL). The implementation takes a BWA index and a query FASTA or FASTQ file as input and outputs the alignment in the SAM format (Li et al., 2009). The query file typically contain many sequences (reads). We process each query sequence in turn, using multiple threads if applicable. Memory usage is dominated by the FM-index, about 3.7 gigabytes (GB) for the human genome. Memory required for each query is roughly proportional to the sequence length. On typical sequencing reads, the total memory is less than 4GB; on one query sequence with 1 million base-pairs (Mbp), the peak memory is 6.4GB in total.

In the implementation, we try to automatically adjust parameters based on the read lengths and sequencing error rates to make the default settings work well for inputs of different characteristics. This behavior is convenient to users who are not familiar with the algorithm and helps performance given reads of mixed lengths and error rates.

3.2 Evaluation on simulated data

On simulated data, we know the correct chromosomal coordinates from the alignment and the evaluation is straightforward.

3.2.1 Overall performance Table 1 shows the CPU time, fraction of confidently aligned reads and alignment error rates for BLAT (v34), BWA-SW (version 0.5.3) and SSAHA2 (version 2.4) given different read lengths and error rates. Unless necessary, we tried to use the default command line options of each aligner. Fine tuning the options based on the characteristics of the input data may yield better performance.

From Table 1, we can see that BWA-SW is clearly the fastest, several times faster than BLAT and SSAHA2 on all inputs, and its speed is not sensitive to the read length or error rates. The accuracy of BWA-SW is comparable to SSAHA2 when the query is long or has low error rate. Given short and error-prone reads, SSAHA2 is more accurate, although it has to spend more time on aligning such reads. SSAHA2 is not tested on the 10kbp reads because it is not designed for this task initially and thus does not perform well. BLAT with the -fastMap option is faster than SSAHA2, but less accurate. Under the default option, BLAT is several to tens of times slower than SSAHA2. The accuracy is higher in comparison to the -fastMap mode, but still lower than that of BWA-SW in general (data not shown).

On memory, both BWA-SW and BLAT uses about 4GB memory. SSAHA2 uses 2.4GB for 500bp or longer reads with the default option, and 5.3GB for shorter reads with the -454 option which increases the number of seed sequences stored in the hash table and increases the memory as a result. In addition, BWA-SW supports multi-threading and thus may take less memory per CPU core if it is run on a multi-core computer. SSAHA2 and BLAT do not support multi-threading at present.

3.2.2 Chimera detection We first study the behavior of each aligner given a chimeric read. To do so, we fabricated two chimeric reads with both consisting of one 1000bp piece from one chromosomal position and one 300bp piece from another position. The main difference between the two reads is the 1000bp piece in the second read has a $\sim$750bp repetitive sequence while the first read is highly unique. When align the two chimeric reads to the human genome, BWA-SW reports four alignments, one for each piece, as is desired. The latest SSAHA2 fails to find the alignment of the 300bp pieces in both reads, although it is able to find the alignments if we align the 300bp piece as an individual read. An older version (1.0.9) is able to align the 300bp piece in the first read by default, but for the second read, we need to switch to a more thorough but much slower configuration that reports all the hits to the 750bp repeat. BLAT with -fastMap does not find the alignment of the 300bp piece for the second read. On the two examples, only BWA-SW has sufficient power to detect chimera.

Furthermore, BWA-SW rarely produces high-quality false chimeric alignments. For example, given the 10,000 1kbp reads with 10% errors but without chimera in simulation, BWA-SW predicts 18 chimeric reads. The mapping quality of the wrongly aligned pieces on these reads is only 2.4 (maximum 11), implying that BWA-SW is aware that these chimera are unreliable. As is expected, BWA-SW produces fewer false chimeric reads given lower base errors.

3.3 Evaluation on real data

Evaluation on real data is complicated by the lack of a ground truth. However, it is still possible to evaluate the relative accuracy by comparing the results from two aligners using the principle that the true alignment tends to have a considerably higher alignment score, because most errors arise from failing to find a seed.

Suppose we align a read using two aligners A and B and get different results. If both A and B give low mapping qualities, the

Table 1. Evaluation on simulated data.

Program	Metrics	100bp 2%	100bp 5%	100bp 10%	200bp 2%	200bp 5%	200bp 10%	500bp 2%	500bp 5%	500bp 10%	1,000bp 2%	1,000bp 5%	1,000bp 10%	10,000bp 2%	10,000bp 5%	10,000bp 10%
BLAT	CPU sec	685	577	559	819	538	486	1078	699	512	1315	862	599	2628	1742	710
	Q20%	68.7	25.5	3.0	92.0	52.9	7.8	97.1	86.3	21.4	97.7	96.4	39.0	98.4	99.0	94.0
	errAln%	0.99	2.48	5.47	0.55	1.72	4.55	0.17	1.12	4.41	0.01	0.52	3.98	0.00	0.00	1.28
BWA-SW	CPU sec	165	125	84	222	168	118	249	172	152	234	168	150	158	134	120
	Q20%	85.1	62.2	19.8	93.8	88.7	49.7	96.1	95.5	85.1	96.9	96.5	95.0	98.4	98.5	98.1
	errAln%	0.01	0.05	0.17	0.00	0.02	0.13	0.00	0.00	0.04	0.00	0.00	0.00	0.00	0.00	0.00
SSAHA2	CPU sec	4872	7962	9345	1932	2236	5252	3311	8213	6863	1554	1583	3113	–	–	–
	Q20%	85.5	83.8	78.2	93.4	93.1	91.9	96.6	96.5	96.1	97.7	97.6	97.4	–	–	–
	errAln%	0.00	0.01	0.19	0.01	0.00	0.01	0.00	0.01	0.04	0.00	0.00	0.00	–	–	–

Approximately 10,000,000bp data of different read lengths and error rates are simulated from the human genome. 20% of errors are indel errors with the indel length drawn from a geometric distribution (density: $0.7 \cdot 0.3^{l-1}$). These simulated reads are aligned back to the human genome with BLAT (option -fastMap), BWA-SW and SSAHA2 (option -454 for 100bp and 200bp reads), respectively. The aligned coordinates are then compared to the simulated coordinates to find alignment errors. In each cell in this table, the three numbers are the CPU seconds on a single-core of an Intel E5420 2.5GHz CPU, percent alignments with mapping quality greater than or equal to 20 (Q20), and percent wrong alignments out of Q20 alignments. SSAHA2 and BWA-SW report mapping quality; BLAT mapping quality is estimated as 250 times the difference of the best and 2nd best alignment scores divided by the best alignment score (essentially the same calculation as the one for BWA-SW).

Table 2. Summary of alignments inconsistent between the BWA-SW and SSAHA2 on real data.

Condition	Count	BWA-SW avgLen	BWA-SW avgDiff	BWA-SW avgMapQ	SSAHA2 avgLen	SSAHA2 avgDiff	SSAHA2 avgMapQ
BWA-SW$\geq$ 20; SSAHA2 unmapped	0	–	–	–	–	–	–
BWA-SW$\geq$ 20 plausible; SSAHA2$<$ 20	1188	398.2	1.3%	178.4	198.3	13.1%	3.9
BWA-SW$\geq$ 20 questionable	40	183.0	7.8%	41.2	280.3	9.4%	2.4
SSAHA2$\geq$ 20; BWA-SW unmapped	946	–	–	–	75.4	6.3%	51.2
SSAHA2$\geq$ 20 plausible; BWA-SW$<$ 20	47	129.0	9.3%	2.5	200.5	8.8%	34.4
SSAHA2$\geq$ 20 questionable	185	400.2	1.7%	13.4	399.2	2.9%	216.4

137,670 454 reads uniformly selected from `SRR003161` were mapped against the human genome with BWA-SW and SSAHA2, respectively. A read is said to be aligned inconsistently if the leftmost coordinates of the BWA-SW and SSAHA2 alignment differs by over 355bp, the average read length. A score, which equals to the number of matches minus three multiplied by the number of differences (mismatches and gaps) in the aligned region, is calculated for each alignment. A BWA-SW alignment is said to be *plausible* if the score derived from the BWA-SW alignment minus the one derived from the SSAHA2 alignment of the same read is greater than or equal to 20 (i.e. the BWA-SW alignment is sufficiently better); otherwise the BWA-SW alignment is said to be *questionable*. Plausible and questionable SSAHA2 alignments are defined in a similar manner. In the table, "BWA-SW$\geq$ 20" denotes the BWA-SW alignments with mapping quality higher than 20. In all, BWA-SW misses 993 (=946+47) alignments which SSAHA2 aligns well, while SSAHA2 misses 1188; 40 BWA-SW Q20 alignments and 185 SSAHA2 Q20 alignments are possibly wrong.

alignment is ambiguous and it does not matter if either alignment is wrong. If A gives high mapping quality and the A alignment score is worse than B, A alignment is probably wrong; even if A alignment score is just a little better than B, A alignment is not reliable and the high mapping quality given by A is still questionable. In practice, defining "a little better" alignment score requires to set a arbitrary threshold on the score difference and therefore this evaluation method is approximate.

Table 2 gives a summary of 454 reads which are mapped by only one aligner or mapped to different places, and are assigned a mapping quality greater or equal to 20 by either BWA-SW or SSAHA2. We can see that BWA-SW tends to miss short alignments with high error rates (946 of them), which agrees with the evaluation on simulated data. SSAHA2 misses alignments for a different reason. On 1,188 reads, SSAHA2 produces obviously wrong alignments. It is aware that these alignments are wrong by assigning low mapping quality, but the true alignments are missed anyway.

For both aligners, most wrong alignments are caused by overlooking alignments with a similar score to the best reported alignment. For example, SSAHA2 aligns read `SRR003161.1261578` to X chromosome with mapping quality 244 and BWA-SW aligns it to chromosome 2 with identical alignment length and edit distance. The existence of two best scoring alignments means the read cannot be uniquely placed and a mapping quality as high as 244 is inaccurate. SSAHA2 gives this high mapping quality probably because it overlooks the match on chromosome 2. And on this specific example, BWA-SW properly gives a mapping quality zero, although it may overlook alternative matches in other examples.

On simulated 100bp and 200bp reads, SSAHA2 with the -454 option delivers better alignments than BWA-SW. On this real data set, BWA-SW is more accurate possibly because the average read length is relatively long (355bp). To confirm this speculation, we compared the two aligners on 99,958 reads from run `SRR002644` with average read length 206bp. This time BWA-SW misses 1,092 SSAHA2 Q20 alignments and produces 39 questionable alignments; SSAHA2 misses 325 and produces 10 questionable ones. SSAHA2 is more accurate on this shorter data set, although it is nine times slower than BWA-SW and uses 40% more memory.

4 DISCUSSION

BWA-SW is an efficient algorithm for aligning a query sequence of a few hundred base-pairs or more against a long reference genome. Its sensitivity and specificity tend to be higher given a long query or a query with low error rate, and on such query sequences, the accuracy of BWA-SW is comparable to the most accurate aligner so far. Furthermore, BWA-SW is able to detect chimaera, potentially caused by structural variations or reference misassemblies, which may pose a challenge to BLAT and SSAHA2.

BWA-SW, BLAT and SSAHA2 all follow the seed-and-extend paradigm. The major difference comes from the seeding strategy. BLAT and SSAHA2 identify short exact matches as seeds, typically of length 11 or 12bp. For k-mer seeding between two sequences of length L and l, respectively, the expected number of seeds is $L \cdot l/4^k$, or of the order of 10^5 for alignment against the human genome. Extending these seeds each with the Smith-Waterman algorithm is expensive. To reduce unnecesary seed extension, both BLAT and SSAHA2 use non-overlapping seeds by default and require multiple seed matches, which should work well for random sequences, but still involves many seed extensions in highly repetitive regions. BWA-SW resolves this issue by using a few long gapped seeds in unique regions. On real biological data, it saves many unnecessary seed extensions and leads to a better overall performance. However, to reduce time when identifying long seeds, BWA-SW only maintains a very small fraction of the dynamic programming matrix, which may miss all seeds for true matches. This heuristic is the major source of alignment errors especially for short queries when there are only few valid unique seeds between the sequences to be aligned. On long alignments, fortunately, the chance of missing all seeds is small. We have shown BWA-SW works equally well as SSAHA2.

BWA-SW differs from BWT-SW in several aspects. First of all, BWT-SW guarantees to find all local matches while BWA-SW is a heuristic algorithm which may miss true hits but is much faster. Secondly, BWA-SW aligns two FM-indices while BWT-SW aligns one sequence and a FM-index. Building a prefix DAWG for the query sequences potentially helps to avoid repeatedly aligning identical substrings in the query, and thus improves the theoretical time complexity. Thirdly, BWA-SW traverses the reference prefix trie in the inner loop while BWT-SW loops through the query sequence in the inner loop. Without heuristics, the BWA-SW approach would hurt performance because we have to trade speed for memory in traversing the reference prefix trie, and it would be more efficient to traverse it in the outer loop. Nonetheless, applying the Z-best strategy requires to know the top-scoring reference nodes matching a query substring without finishing the dynamic programming and thus only works when the reference is traversed in the inner loop. Fourthly, BWA-SW only reports alignments largely non-overlapping on the query sequence, while BWT-SW, like BLAST, reports all statistically significant alignments. BWA-SW retains key information of alignments and generates much smaller and more convenient output. For BWT-SW, end users usually need to post-process the results to filter out many alignments of little interest to them. In all, BWA-SW is tuned towards practical usefulness given large-scale real data.

The high speed of BWA-SW largely comes from two strategies: the use of FM-indices and the suppression of short repetitive matches contained in a better match. While the first strategy is not applicable to hash table based algorithms such as SSAHA2 and BLAT, the second strategy could be implemented in such programs and may substantially accelerate them by saving much time on the construction of repetitive alignments. And although the use of BWT reduces unnecessary alignments in repeats, each BWT operation comes with a large constant in comparison to a hash table look up. It is still possible that hash table based algorithms could be faster than BWA-SW if they incorporated some of these features.

ACKNOWLEDGEMENTS

We are grateful to Zemin Ning for his helpful comments on the SSAHA2 algorithm, and to Kimmo Palin for providing the literature on directed acyclic word graph (DAWG).

Funding: Wellcome Trust 077192/Z/05/Z.

REFERENCES

Altschul, S. F. et al. (1997). Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res*, 25:3389–3402.

Blumer, A. et al. (1985). The smallest automaton recognizing the subwords of a text. *Theoretical Computer Science*, 40:31–55.

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

Eid, J. et al. (2009). Real-time DNA sequencing from single polymerase molecules. *Science*, 323:133–138.

Ferragina, P. and Manzini, G. (2000). Opportunistic data structures with applications. In *Proceedings of the 41st Symposium on Foundations of Computer Science (FOCS 2000)*, pages 390–398.

Jiang, H. and Wong, W. H. (2008). Seqmap: mapping massive amount of oligonucleotides to the genome. *Bioinformatics*, 24:2395–2396.

Kent, W. J. (2002). BLAT–the BLAST-like alignment tool. *Genome Res*, 12:656–664.

Lam, T. W. et al. (2008). Compressed indexing and local alignment of DNA. *Bioinformatics*, 24:791–797.

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. et al. (2008a). Mapping short dna sequencing reads and calling variants using mapping quality scores. *Genome Res*, 18:1851–1858.

Li, H. et al. (2009). The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, 25:2078–2079.

Li, R. et al. (2008b). SOAP: short oligonucleotide alignment program. *Bioinformatics*, 24:713–714.

Lin, H. et al. (2008). Zoom! zillions of oligos mapped. *Bioinformatics*, 24:2431–2437.

Ma, B. et al. (2002). Patternhunter: faster and more sensitive homology search. *Bioinformatics*, 18:440–445.

Meek, C. et al. (2003). OASIS: An online and accurate technique for local-alignment searches on biological sequences. In *Proceedings of 29th International Conference* *on Very Large Data Bases (VLDB 2003)*, pages 910–921.

Morgulis, A. et al. (2008). Database indexing for production megablast searches. *Bioinformatics*, 24:1757–1764.

Ning, Z. et al. (2001). SSAHA: a fast search method for large DNA databases. *Genome Res*, 11:1725–1729.

Pearson, W. R. and Lipman, D. J. (1988). Improved tools for biological sequence comparison. *Proc Natl Acad Sci U S A*, 85:2444–2448.

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

Smith, A. D. et al. (2008). Using quality scores and longer reads improves accuracy of solexa read mapping. *BMC Bioinformatics*, 9:128.

Weese, D. et al. (2009). RazerS–fast read mapping with sensitivity control. *Genome Res*, 19:1646–1654.

Zhang, Z. et al. (2000). A greedy algorithm for aligning dna sequences. *J Comput Biol*, 7:203–214.