BIOINFORMATICS

Vol. 00 no. 00 2012
Pages 1–8

RazerS 3: Faster, fully sensitive read mapping

David Weese*, Manuel Holtgrewe and Knut Reinert

Department of Mathematics and Computer Science,
Freie Universität Berlin, Takustr. 9, 14195 Berlin, Germany

Received on 2012-03-03; revised on 2012-08-06; accepted on 2012-08-09

Associate Editor: Dr. Michael Brudno

ABSTRACT

Motivation: During the last years NGS sequencing has become a key technology for many applications in the biomedical sciences. Throughput continues to increase and new protocols provide longer reads than currently available. In almost all applications, read mapping is a first step. Hence, it is crucial to have algorithms and implementations that perform fast, with high sensitivity, and are able to deal with long reads and a large absolute number of indels.

Results: RazerS is a read mapping program with adjustable sensitivity based on counting q-grams. In this work we propose the successor RazerS 3 which now supports shared-memory parallelism, an additional seed-based filter with adjustable sensitivity, a much faster, banded version of the Myers' bit-vector algorithm for verification, memory saving measures and support for the SAM output format. This leads to a much improved performance for mapping reads, in particular long reads with many errors. We extensively compare RazerS 3 with other popular read mappers and show that its results are often superior to them in terms of sensitivity while exhibiting practical and often competetive run times. In addition, RazerS 3 works without a precomputed index.

Availability and Implementation: Source code and binaries are freely available for download at http://www.seqan.de/projects/razers. RazerS 3 is implemented in C++ and OpenMP under a GPL license using the SeqAn library and supports Linux, Mac OS X, and Windows.

Contact: david.weese@fu-berlin.de

1 INTRODUCTION

Next generation sequencing allows researchers to produce billions of base pairs (bp) within days in the form of reads of length 100 bp and more. It has become an invaluable technology for a multitude of applications, e.g. the detection of SNPs (single nucleotide polymorphisms) and large structural genome variations, targeted or de-novo genome or transcriptome assembly, isoform prediction and quantification, identification of transcription factor binding sites or methylation patterns. In many of these applications mapping sequenced reads to their potential genomic origin is the first fundamental step for subsequent analyses.

A variety of tools have been designed specifically for the purpose of mapping short reads. In a recent publication, Li and Homer (2010) give a survey and categorize the existing tools into approaches using a q-gram index for a seed-and-extend strategy, e.g. (Alkan *et al.*, 2009; Weese *et al.*, 2009; Bauer *et al.*, 2010; Ahmadi *et al.*, 2011; David *et al.*, 2011), or recursively descending a suffix tree (Hoffmann *et al.*, 2009) or prefix tree (Langmead *et al.*, 2009; Li and Durbin, 2009; Langmead and Salzberg, 2012; Li *et al.*, 2009) of the reference genome.

Recursive approaches are usually designed for the fast search of one or a few locations where reads map with low error rates. These search algorithms are mostly based on heuristics and optimized for speed instead of enumerating all possible locations. Conversely, approaches based on the seed-and-extend strategy allow such an (often approximate) enumeration. The first class of approaches aims at directly finding the "best" location for mapping a read (*best-mappers*) while the second class aims at enumerating a comprehensive set of locations (*all-mappers*).

RazerS (Weese *et al.*, 2009) is an all-mapper that uses q-gram counting for read mapping with controllable sensitivity. This means it can guarantee to find all locations a read maps to in a reference sequence. At the same time, it works with practicable performance.

Since the original publication in 2009, sequencing technology has advanced to produce longer reads. The increasing length leads to a larger absolute number of errors to be considered, a problem that is aggravated by new technologies that have a higher error rate (e.g. PacBio). Older read mappers have difficulties mapping long reads with high number of errors with a high sensitivity.

In this paper we address this problem and propose a new read mapper RazerS 3 which is able to map reads of arbitrary length with a large number of indel errors. Our novel contributions are: 1) The use of OpenMP to provide a shared-memory parallelization with dynamic load balancing. 2) In addition to the q-gram counting filter used in RazerS, we implemented a pigeonhole based filter with controllable sensitivity, since it proved to be superior for low error rates. 3) An implementation of a banded version of Myers' bit-vector algorithm which we use for the verification, similar to (Hyyrö, 2003), which is up to 4 times faster than the previous, unbanded version.

These algorithmic improvements lead to a running time that is an order of magnitude faster than RazerS while keeping the guarantee for full sensitivity. Various, extensive benchmarks show higher sensitivity when compared to other approaches, especially best-mappers. Furthermore, the running time is superior to the considered all-mappers and competitive or superior to best-mappers on medium-sized genomes. On large genomes, the running time is still practical, and only about 3 times slower than that of BWA while being more sensitive. RazerS 3 does not rely on a precomputed index like the tree-based tools and is in this respect more flexible.

*to whom correspondence should be addressed

2 METHODS

The RazerS algorithm consists of a *filtration* and a *verification* part. In the filtration part, the genome is scanned for regions that possibly contain read matches. Results from the filtration are then subjected to a verification algorithm.

Formally, we consider the read mapping problem: The input is a reference sequence S, a set of reads R, a distance function δ, and a maximal distance k. The solution of the problem is the set of all locations (*matches*) in S where the read r is found with distance $\leq k$ under δ for each read r.

Common distance measures are Hamming and edit distance. The Hamming distance counts the minimal number of replacements while the edit distance allows insertions and deletions (*indels*). Under edit distance, matches can be ambiguous. The authors explained how to treat such ambiguities and how to derive a gold standard benchmark for read mapping in (Holtgrewe *et al.*, 2011). This is summarized in Section 2.3.

RazerS 3 supports both Hamming and edit distance read mapping. Both modes can be run with full or user-definable sensitivity on multiple CPU cores which allows for time-sensitivity trade-off.

2.1 Filtration

To make RazerS 3 applicable to a broad spectrum of use cases, we implemented two fast filtration algorithms, which differ in filtration specificity and processing speed. In Section 3.1, we analyze which filter performs best under different typical read mapping scenarios. The first filter, based on the SWIFT algorithm, was already used in RazerS and is hence only shortly described here. It is still operational in RazerS 3 since it is superior to the second filter for high error rates.

SWIFT. The first filter is a modified SWIFT algorithm (Rasmussen *et al.*, 2006) which divides the dot plot between genome and reads into overlapping parallelograms. In a linear scan over the reference sequence, the number of common exact q-grams between read and the reference subsequence is counted for each parallelogram. Parallelograms that contain a sufficient number of common exact q-grams are considered as candidate regions of semi-global alignments between reads and reference sequences with a tolerated number of errors. For more details, we refer the reader to (Weese *et al.*, 2009).

Pigeonhole principle. The second filter is new and is based on the pigeonhole principle which states that if a read is cut into $k+1$ pieces then in every approximate match of the read with at most k errors occurs at least one piece without error (Baeza-Yates and Navarro, 1999). If all reads have the same length m, they are cut into $\lfloor \varepsilon m + 1 \rfloor$ pieces of length $q = \lfloor m / \lfloor \varepsilon m + 1 \rfloor \rfloor$, where ε is the tolerated error rate. For reads of arbitrary length, the minimal q is chosen to build a q-gram index over the pieces of the reads. These pieces are then searched in a linear scan of the reference sequence. For every exact match the dot plot parallelogram, consisting of the diagonals that are at most k diagonals apart of the matching piece, is considered as a candidate region for a match within the tolerated edit distance. In Hamming distance mode, only the diagonals that cover matching pieces are considered as candidate regions. The candidate parallelograms of all matching pieces are recorded and verified in the subsequent verification step. Compared to the SWIFT filter this filter requires less processing time and, due to non-overlapping seeds, less indexed q-grams at the expense of less filtration specificity and more verifications (more results can be found in Table S1).

2.2 Lossy filtration and prediction of sensitivity

Both filters are fully sensitive if parameterized as described above, i.e. every occurrence of a read within the tolerated edit or Hamming distance will be detected as a candidate region and positively verified in the verification step. In our previous work the use of a lossy filter could improve the overall running time by an order of magnitude while still detecting 99 % of all matches, see (Weese *et al.*, 2009). Our approach is based on given positional error probabilities p_i, i.e. the probability that in a randomly chosen true match of any read there is an error at position i. As errors we consider base miscalls and mutations, and before mapping compute the error profile p_i based on base-call quality values and a user-specific mutation rate. Given the error profile and specific filtration parameters, we propose how to estimate the probability to miss a random match and vice versa how to choose more efficient filtration parameters that guarantee a specific minimal sensitivity. We sketch the procedure shortly for the SWIFT filter and elaborate then the method for the pigeonhole filter.

Predicting SWIFT sensitivity. The SWIFT filter has 2 parameters, the q-gram shape Q and the threshold t. The shape is a set of relative positions of the q considered characters. For example, ##-# corresponds to a 3-gram with shape $Q = \{0, 1, 3\}$ and the two 3-grams with shape Q in the string `GTTCA` are `GTC` and `TTA`. Of all overlapping q-grams with shape Q contained in a read, the threshold t is the minimal number of q-grams occurring without error in the reference a candidate region must have. By increasing q or t the number of candidate regions and also the overall running time can be reduced at the expense of filtration sensitivity.

To decide whether an arbitrary Hamming distance match is detected as a candidate region, it suffices to consider the positions of replacements between read and reference instead of whole sequences and count the number of q-grams without a replacement. Assuming the independence of errors, the occurrence probability of this match can be computed by the given positional error probabilities. In (Weese *et al.*, 2009), we devised an algorithm to efficiently compute the sensitivity using dynamic programming instead of exhaustive search for both, Hamming and edit distance.

Predicting pigeonhole sensitivity. A lossless pigeonhole filter divides a read into at least $k+1$ fragments and uses them as seeds to detect all k-error matches. As fragments we use the first $k+1$ non-overlapping read q-grams where q is chosen as large as possible. In expectation, every read q-gram has $n/4^q$ occurrences in a genome of length n. To reduce the number of random candidates and to reduce the overall running time, we increase q and allow the seeds to overlap. However, with overlapping seeds some of the mismatch patterns will be missed by the filter, e.g. if every odd seed overlap contains an error. With a (q, Δ)-seed filter we denote a filter that uses all q-grams starting at multiples of Δ in the read as seeds, with $q/2 \leq \Delta \leq q$, such that adjacent q-grams overlap by $q - \Delta$ characters. To compute the sensitivity of such a filter, we consider mismatch patterns between a read of length m and all of its true matches. (A mismatch pattern is a binary string, with `0`'s at matching and `1`'s at mismatching positions.) The sensitivity for matches with $e = 0, 1, \ldots, k$ errors is the sum of occurrence probabilities of e-error mismatch patterns that

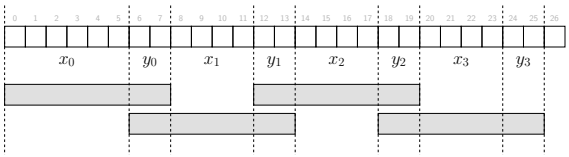

Fig. 1. A (q, Δ)-seed filter, with $q = 8$ and $\Delta = 6$, for searching matches with up to $k = 3$ errors (seed i consists of segments y_{i-1}, x_i, and y_i, except for $i = 0$).

are detected by the filter divided by the probability that an e-error mismatch patterns occurs. Instead of enumerating all possible e-error mismatch patterns we devised a DP algorithm that virtually split the mismatch pattern into segments at q-gram boundaries $\Delta, q, 2\Delta, \Delta + q, \ldots, (k+1)\Delta, k\Delta + q$ and denote the first $2(k+1)$ segments from left to right as $x_0, y_0, x_1, y_1 \ldots, x_k, y_k$ (see Figure 1). Our approach is analogously applicable to edit distance as insertions or deletions behave like mismatches in relation to destroyed seeds.

The probability $P\big(\|M[i..j)\|_1 = e\big)$ that a random mismatch pattern M contains e errors in a segment from position i to $j-1$ can be computed as follows using positional error probabilities p_i:

$$P\big(\|M[i..i)\|_1 = e\big) = \begin{cases} 1, & \text{if } e = 0 \\ 0, & \text{else.} \end{cases}$$

$$P\big(\|M[i..i+1)\|_1 = e\big) = \begin{cases} 1 - p_i, & \text{if } e = 0 \\ p_i, & \text{if } e = 1 \\ 0, & \text{else.} \end{cases}$$

$$\begin{aligned} P\big(\|M[i..j)\|_1 = e\big) &= (1 - p_{j-1}) \cdot P\big(\|M[i..j-1)\|_1 = e\big) \\ &\quad + p_{j-1} \cdot P\big(\|M[i..j-1)\|_1 = e-1\big) \ . \end{aligned}$$

We define $L(i, e, y)$ to be the probability of the event that the first $i+1$ seeds contain overall e errors, each at least one error, and y_i contains y errors. Let X_i and Y_i be random variables for the number of errors in the segments x_i and y_i, then L can recursively be computed as follows:

$$L(0, e, y) = \begin{cases} 0, & \text{for } e = 0 \\ P(X_0 = e - y) \cdot P(Y_0 = y), & \text{else.} \end{cases}$$

$$L(i, e, y) = \sum_{s=1}^{e} \sum_{y'=0}^{s-y} L(i-1, e-s+y', y') \cdot P(X_i = s-y-y') \cdot P(Y_i = y) \, .$$

The probability that all seeds are destroyed with overall e errors is:

$$L_{\text{all}}(e) = \sum_{y=0}^{e} \sum_{x=0}^{e} L(k, e-x, y) \cdot P(\|M[k\Delta + q \,..\, n)\|_1 = x) \ ,$$

and consequently the sensitivity of the (q, Δ)-seed filter for matches with at most k errors is:

$$S(q, \Delta, k) = 1 - \sum_{e=0}^{k} \frac{L_{\text{all}}(e)}{P(\|M\|_1 = e)} \ .$$

Before starting the mapping, RazerS 3 estimates the sensitivities of different filter settings and maximizes the seed length q as it has the greatest influence on the overall running time. Beginning with the lossless setting $q = \Delta = \lfloor m/(k+1) \rfloor$, it step-wise increases q as long as the estimated sensitivity is higher than required, q does not exceed the maximal seed length of 31 and not more than two seeds overlap ($q \leq 2\Delta$). The corresponding step sizes $\Delta = \lfloor (m-q)/k \rfloor$ are chosen such that each read contains $k+1$ overlapping seeds.

2.3 Verification

The result of the above described filtration part is a set of candidate regions and reads potentially matching there. A candidate region is a parallelogram in the dot plot that might contain the alignment trace of a match and hence has to be verified by the verification part explained in the following.

Hamming distance verification. In Hamming mode, a match covers solely one dot plot diagonal. Hence, the candidate parallelogram can be verified by scanning each diagonal while counting the number of mismatches between read and reference sequence. A diagonal can be skipped as soon as the counter exceeds the number of tolerated errors. Otherwise, a match has been found.

Edit distance verification. For edit distance verification, we implemented a banded version of Myers (1999) bit-vector algorithm as it was proposed in (Hyyrö, 2003) with small adaptions. The original algorithm by Myers can be used to search a read with at most k edit errors in the reference sequence. The underlying idea is the same as in (Needleman and Wunsch, 1970) but the implementation is much more efficient as it encodes a whole DP column in two bit-vectors and computes the adjacent column in a constant number of 12 logical and 3 arithmetical operations. For reads up to length 64 bp, CPU registers can be used directly. For longer reads bit-vectors and operations must be emulated using multiple words where only words affecting a possible match needs to be updated (Ukkonen, 1985). However, the additional processing overhead results in a performance drop for reads of length 65 bp and longer. The variant proposed by Hyyrö computes a banded semi-global alignment between read and reference, i.e. it only computes DP cells that are covered by a parallelogram. Hence, only the columns of the parallelogram needs to be encoded by bit-vectors which makes it applicable to parallelograms of width up to 63 without the need for bit-vector emulation. However, the banded variant proposed in (Hyyrö, 2003) requires bitmasks consisting of multiple words for each read as preprocessing information. We implemented a banded variant of Myers' algorithm that requires no preprocessing information at all as we update the 5 single-word bitmasks (we consider the alphabet $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}, \mathtt{N}\}$) during each verification. This strategy is faster than that of Hyyrö and saves memory. Further improvements like the support of clipped parallelograms are explained in Section S1.

In contrast to Hamming distance verification, where the difference between begin and end position of every match equals the read length, Myers' algorithm outputs only the end of a match. More precisely, it determines the minimal number of errors for a fixed end position and a free begin position. To determine a corresponding begin position we search the read backwards with a fixed end position. As edit distance scores mismatches and indels equally, there can be multiple best match beginnings. We choose the largest best match to optionally shrink it later using an alignment algorithm for affine gap costs (Gotoh, 1982) where we penalize gaps slightly more than mismatches and penalize an opened gap more than an extended a gap. More implementation details of our banded variant of Myers' algorithm can be found in the supplement in Section S1.

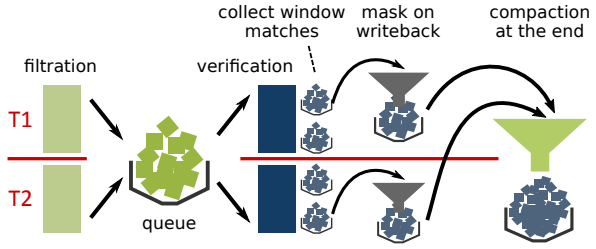

Fig. 2. Overview of RazerS 3. Large green/blue rectangles represent the filter/verification states (in the Figure with two threads T1 and T2). Small green/blue squares represent filtration/verification work packages. Gray/green funnels represent the masking/compaction step.

Island criterion. Another improvement in RazerS 3 addresses the problem of defining the term *match* for read mapping. This is discussed in detail by Holtgrewe *et al.* (2011) when defining the Rabema benchmark. We will give a summary of this here.

Read alignments under edit distance can be ambiguous if more than one error is allowed. Say, for example, a read aligns perfectly except for the first base where we observe a mismatch. Let the beginning of the read be `ACT`... and align with the genome stretch ...`ACCT`.... Then there may be two optimal alignments where the read starts with either `ACT`... or `A-CT`....

Such and other ambiguities lead to possibly several local minima (in terms of edit distance) around a match. The model for matches defined in (Holtgrewe *et al.*, 2011) describes a relaxation of the naïve requirement to enumerate all edit distance alignments. RazerS 3 uses this model and writes out at least one result record for each Rabema match.

2.4 Parallelization

Match management. The overlapping parallelograms of the SWIFT filter or the multiple seeds the pigeonhole filter may find in a single match, result in multiple identical or nearly identical matches found in the verification step. To filter these *duplicates*, we regularly search for matches of the same read that have an identical begin or end position and keep only those with a minimal number of errors. Additionally we use a heuristic in the pigeonhole filter, that for multiple seeds on the same diagonal only one candidate region is generated.

If the user specifies a maximal number M of matches per read, we sort all matches ascendingly by the number of errors and remove all but the first M matches of each read. For a read, the number of errors e in the M-th match is used to dynamically adjust the filter and verifier in order to search only for matches with less than e errors. If e equals 0 the read can be disabled completely.

Processing in window and batches. RazerS 3 collects all candidates from windows of a configured size (default: 500 kbp). The resulting candidates are split into work packages of a configured size (default: 100) or a larger size if a configured number of packages is exceeded for a window (default: 100 packages). Each package is then verified by a single thread.

Thus, the filtration is performed in a window-based fashion and verification is performed in batches. Locks for shared data structures only have to be obtained once for each window or batch. This way, lock contention and overhead are kept small while still allowing for fine-granular load balancing.

Load balancing scheme. We implemented a mixture of static and dynamic load balancing: For filtration, reads are statically assigned to threads in subsets of equal size. Each thread has a filter for its owned reads as well as a verifier, shown as large green/blue rectangles in Figure 2. Filtration results (green squares) are written to a global work queue.

After thread T_i completes the filtration in its current window, it takes candidate packages from the global queue until empty and verifies it. Thus, the verification work is distributed over threads and dynamically load balanced.

Each thread has a queue for each other thread and itself (labeled with the thread id) acting as a post box. Thread T_i then writes the verification results (blue square) to the post box for the owner of its current work package. It then writes the longest consecutive stretch of globally available verification results addressed to itself back into its local result container. (The arrays of matches are subdivided into work packages, of which each has an index. A *consecutive* sequence of packages is a sequence of packages whose indices are consecutive.) Matches are masked when written back. At the end of the program run, each thread performs a global compaction step on its result. A detailed analysis of influence of the chosen filter on load balancing is given in the supplement Section S2.

Further improvements. Another optimization in RazerS 3 is a reduction of running time of the masking step by conducting local sorting instead of global sorting. As a memory optimization each filter uses an open addressing q-gram index whose memory footprint is linear in the number of stored q-grams (see Sections S3 and S4 for details).

3 EXPERIMENTAL RESULTS

We compared RazerS 3 with the best-mappers Bowtie 2, BWA, and Soap 2 as well as the all-mappers Hobbes, mrFAST, and SHRiMP 2. For running time comparison, we ran the tools with 12 threads and used local disks for I/O. We used default parameters, except where stated otherwise. Read mappers that accept a maximal number of errors (mrFAST, Hobbes, Soap 2) were configured with the same error rate as RazerS 3. For a fair comparison with best-mappers, we configured RazerS 3 in a second variant to also output one best match per read. The exact parametrization is described in Section S6.

All read sets are given by their SRA/ENA id. As references we used whole genomes of E. coli (NCBI NC_000913.2), C. elegans (WormBase WS195), D. melanogaster (FlyBase release 5.42), and human (GRCh37.p2). The mapping times were measured on a cluster of nodes with 72 GB RAM and 2 Intel Xeon X5650 processors (each with 6 cores) per node running Linux 3.2.0.

3.1 Comparing the SWIFT and pigeonhole filters

RazerS 3 provides support for two string metrics (Hamming and edit distance) and two filter variants (SWIFT and pigeonhole filter). To investigate which filter performs best on which kind of input and metric, we conducted an experimental evaluation of the time required to map different real datasets for varying mapping settings.

For this reason, we ran RazerS 3 in both filtration modes for reads of lengths 30, 50, 70, and 100 bp for the references of E. coli, C. elegans and chr. 2 of human with error rates of 0 to 10 %. Figure 3 shows an excerpt of the resulting experimental map for mapping

reads to chr. 2 of human using Hamming and edit distance at 100 % sensitivity. Figure S5 shows the full result set and Table S2 describes the datasets we used.

The result shown in Figure 3 is representative for our overall results and shows the running time ratios between mapping with the pigeonhole and SWIFT filter. We observe, that for edit distance, the pigeonhole filter always leads to shorter running times than the SWIFT filter. For Hamming distance, the pigeonhole filter is well suited for low error rates (up to 6 %), while the SWIFT filter yields better mapping times for higher error rates. Astonishingly, the factors between the two methods range from 1:32 to 32:1.

The differences in mapping times can be explained by the different characteristics of both filters. Compared to SWIFT, the simpler but less specific pigeonhole filter requires no counting and hence less processing overhead which compensates the increased number of verifications for low error rates. With an increase in error rate the specificity of both filters deteriorates equally for edit distance. For Hamming distance, gapped shapes compensate this degradation and make the SWIFT filter much more specific than the pigeonhole filter which is based on ungapped q-grams. Section S2 gives a detailed comparison of the influence of the filter choice on the running time and load balancing.

In the following, we will denote RazerS 3 in edit distance mode using the pigeonhole filter with given sensitivity rate as R3-100 and R3-99. Similarly, we will denote RazerS 3 using SWIFT in edit distance mode with R3-SW-99 and R3-SW-100.

3.2 Verification of expected pigeonhole sensitivity

In this experiment, we examine the accuracy of the sensitivity estimation for our new pigeonhole filter.

We used 10 M fly reads of length 75 bp (SRR060093) and 10 M human reads of length 100 bp (ERR012100). For each read set, we computed a reference set consisting of all reads that can be mapped uniquely with up to 5 % errors. From the mapping results we determined positional error probabilities and used them to estimate the fraction of k-error matches lost by a (q, Δ)-seed filter (loss rate) while varying the q-gram length $q = 16, \ldots, 31$ and the q-gram overlap $q - \Delta = 0, \ldots, 10$. The estimated loss rates were compared with the loss rates observed after mapping the reference read sets with the same filtration settings and are shown on the left-hand side

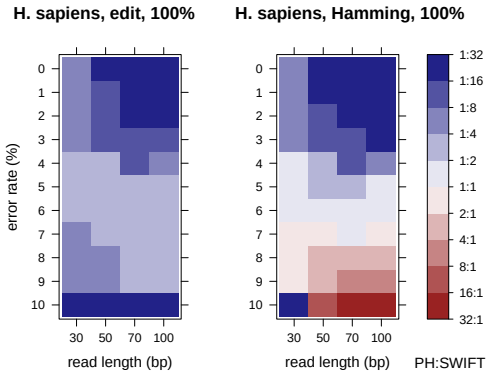

Fig. 3. Experimental map for human chr. 2 with different read lengths and error rates. Ratios between the mapping times with pigeonhole and SWIFT are color coded in the plots.

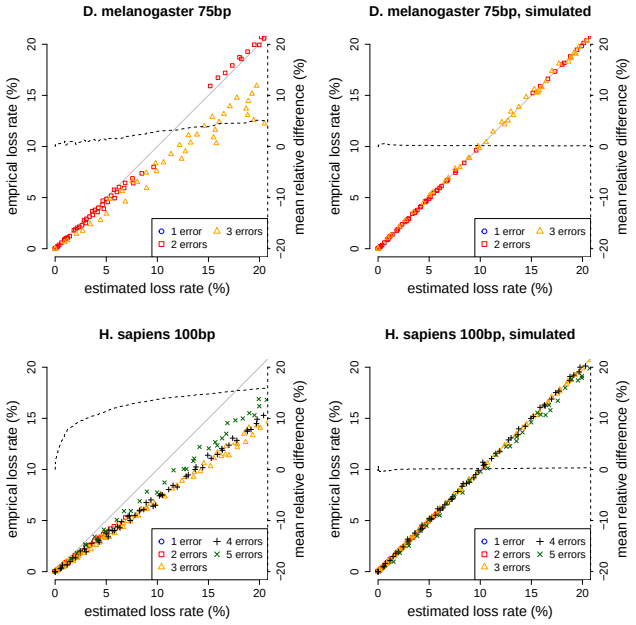

Fig. 4. Validation of the estimated sensitivity. We compared the estimated with the observed loss rate (1−sensitivity) of unique matches with different numbers of errors for (q, Δ)-seed filters with $q = 16, \ldots, 31$ and overlaps between 0 and 10. We evaluated real (a) and simulated reads (b) using the observed error profile.

in Figure 4. As a sanity check, we simulated 10 M reads of length 75 bp and 100 bp from the fly and human genome and implanted errors with the same positional error profile and repeated the whole comparison.

The results are shown on the right-hand side in Figure 4. Dots below the diagonal correspond to experiments with an empirical sensitivity higher than estimated and above the diagonal the empirical sensitivity was overestimated. As a measure of accuracy we use the relative difference between empirical and estimated loss rate. The dashed line shows the mean relative difference of all experiments up to a certain estimated loss rate. We observe a high level of agreement for simulated reads with a mean relative difference below 1 % for loss rates between 0 and 10 %. On real data the predicted loss rates between 0 and 10 % show a mean relative difference of 3 % on the fly and 14 % on the human read set.

We explain this deviation by a correlation of sequencing errors at adjacent positions, whereas our model assumes independence of errors. This error correlation has also been observed in (Dohm *et al.*, 2008) and may be the result of molecules which are out of phase for multiple cycles in the sequencing process and lead to interferences with signals of adjacent bases. However, this correlation shows no negative influence as in none of our experiments the effective sensitivity was overestimated by our model.

3.3 Rabema benchmark results

Next, we used the Rabema benchmark (Holtgrewe *et al.*, 2011) (v1.1) for a thorough evaluation and comparison of read mapping sensitivity. As datasets we simulated 100 k reads of length 100 bp

from the whole human genome with Mason (Holtgrewe, 2010) and distributed sequencing errors like in a typical Illumina experiment (see Section S9 for more details).

The benchmark contains the categories *all*, *all-best*, *any-best*, and *recall*. In the categories all, all-best, and any-best a read mapper has to find all, all of the best, or any of the best edit distance matches for each read. The category recall requires a read mapper to find the *original* location of each read, which is a measure independent of the used scoring model (edit-distance or quality-based). The benchmark was performed for an error rate of 5 %.

To compare the sensitivity fairly, we configured read mappers as best-mappers and as all-mappers if possible (BWA, Bowtie 2, and RazerS 3). We parametrized the best-mappers for high sensitivity and multiple matches. We do not consider running time here, since best-mappers are not designed for finding all matches and consequently consume more time (up to 3 hours in a run compared to several minutes). The aim here was to investigate sensitivity and recall.

The results are shown in the left part of Table 1. As expected, the all-mappers generally perform better than the best-mappers. Also, as expected, mappers lose more of the high-error locations than low-error locations. Surprisingly, Bowtie 2 and BWA are better than the all-mapper Hobbes. Soap 2 is low sensitive to reads with more than 2 errors as it allows at most 2 mismatches in total and by chance aligns some of the reads with more errors by replacing all `N`'s in the reads by a `G`'s. R3-100 is the most sensitive method, followed by mrFAST (which is not fully sensitive for higher error rates), SHRiMP 2, and Bowtie 2. Even when configured as a best-mapper (i.e. only reporting one best match), RazerS 3 achieves the best scores.

3.4 Variant detection

The next experiment analyzes the applicability of RazerS 3 and other read mappers in sequence variation pipelines. Similarly to the evaluation in (David *et al.*, 2011), we generated 5 million read pairs of length 2×100 bp with sequencing errors, SNPs, and indels from the whole human genome such that each read has an edit distance of at most 5 to its genomic origin. To distribute sequencing errors according to a typical Illumina run, we used the read simulator Mason with the default profile settings. The reads (pairs) were grouped according to the numbers of contained SNPs and indels, where the group (s, i) consists of reads (pairs) with s SNPs and i indels in total. We mapped the reads both as single and paired-end reads and measured the sensitivities separately for each class and read mapper.

A read (pair) was mapped *correctly* if an alignment (paired alignment) has been found within 10 bp of the genomic origin. It is considered to map *uniquely* if only one alignment was reported by the mapper. For each class we define *recall* to be the fraction of all contained reads (pairs) and *precision* the fraction of uniquely mapped reads (pairs) that were mapped correctly. The right side of Table 1 shows the results for each read mapper and class, where the upper and lower table contain the single-end and paired-end results. An extended version of this table is given in Section S11.

Comparing the all-mappers results, R3-100 shows the highest recall and precision values on both the single and paired-end datasets. mrFAST is also full-sensitive on the single-end dataset but has a low recall value of 8% for pairs with 5 bp indels. SHRiMP 2 shows full precision in all classes and experiments but misses some non-unique alignments. Hobbes appears to have problems with indels and shows the lowest sensitivities in the all-mapper comparison.

Surprisingly, R3-100 is the most sensitive best-mapper even in the non-variant class (0,0) where the simulated qualities could possibly give quality-based mappers an advantage. For paired-end reads where matches are also ranked by their deviation from the library size, it is even more sensitive than the all-mappers Hobbes and mrFAST. As observed in (David *et al.*, 2011), quality-based mappers like Bowtie 2, BWA, and Soap 2 are not suited to reliably detect the origin of reads with variants. Their recall values deteriorate with more variants as they prefer alignments where mismatches can be explained by sequencing errors instead of natural sequence variants. The low sensitivity of Soap 2 is again due to its limitation to at most 2 mismatches.

3.5 Performance comparison

In the last experiment we compare the real-world performance of RazerS 3 with other read mappers. To this end, we mapped four different sets of 10 million Illumina read pairs of length 2×100 bp from E. coli, C. elegans, fly, and human, as well as six simulated datasets consisting of 1 million simulated read pairs of length 2×200 bp, 2×400 bp, and 2×800 bp from fly and human to their reference genomes. We mapped the reads both as single and paired-end reads with 4% error rate and measured running times, peak memory consumptions, mapped reads (pairs), and reads (pairs) mapped with minimal edit distance. We compared RazerS 3 in default mode with other all-mappers and configured it to output only one best match per read for the best-mapper comparison. Since mrFAST supports no shared-memory parallelization we split the reads into packages of 500 k reads and mapped them with 12 concurrent processes of mrFAST. Hobbes' large memory consumption also required to map the reads package-wise but with a single process and 12 threads.

For the evaluation we use the commonly used measure of percentage of *mapped reads (pairs)*, i.e. the fraction of reads (pairs) that are reported as aligned in the result file of the mapper. However, as some mappers report alignments without constraints on the number of errors, we also determine the fraction of reads (pairs) whose best match has an error rate of at most 0%, ..., 4% (small numbers in the mapped reads (pairs) column in Tables 2 and 3).

We call a read (pair) *ε-mappable*, if it can be aligned with an error rate of ε (by any mapper). As a more stringent measure for edit distance mappers, we call an ε-mappable read (pair) *correctly mapped* if at least one (paired) alignment has been found with an error rate of ε. For each mapper we measured the percentage of *correctly mapped reads (pairs)*, i.e. the fraction of ε-mappable reads (pairs) for $\varepsilon \in [0, 4\%]$ that are correctly mapped. For a more detailed analysis we additionally give the percentages separately for sets of $\varepsilon = 0, \varepsilon \in (0, 1\%], \ldots, \varepsilon \in (3, 4\%]$.

The results for the fly and human Illumina datasets as well as the simulated 800 bp fly dataset are shown in Tables 2 and 3. More detailed tables of all datasets are given in Section S12.

As can be seen, R3-100 aligns all reads with the minimal number of errors and achieves the best percentage of correctly mapped reads followed by R3-95 in all experiments. A decrease in the specified sensitivity results in a decrease in running time and on the human

genome R3-95 is up to twice as fast as R3-100. As in the previous experiments, the actual sensitivity is always higher than specified.

All-mapper comparison. For the single-end 100 bp datasets mrFAST is as sensitive but 4 times slower than R3-100. On paired-end reads it is less sensitive and apparently has problems to map long reads with an increased number of absolute errors. In the results of the Illumina paired-end datasets we found some alignments with actual more errors than asserted by mrFAST and an error rate above 4%. Thus the number of totally mapped pairs is slightly higher compared to R3-100 on the Illumina paired-end reads. On single-end reads Hobbes is about 2 times slower and only on human paired-end reads faster (up to 2 times) than R3-100. It maps 5–15% less reads correctly and also the total number of mapped reads is less. Hobbes is not able to map reads longer than 100 bp and some single-end read packages could not be mapped due to repeated crashes (4 of 20 for C. elegans and 1 of 20 for human). As SHRiMP 2 does not use a maximal error rate it outputs more mapped reads than R3-100 in total. However, the percentages of correctly mapped reads is less in all experiments. This could be due to its different scoring scheme, where two mismatches costs less than opening a gap, but does not explain why it misses reads with 0 errors. SHRiMP 2 is 5–23 times slower than R3-100 on the Illumina datasets and up to 600 times slower on the 800 bp datasets.

Best-mapper comparison. Compared to other best-mappers, R3-95 is faster or equally fast on all E. coli, C. elegans, and fly datasets. For human reads of length 100–200 bp it is 2–3 times slower than BWA and equally fast or faster for longer reads. BWA and Bowtie 2 could not be run with a maximal error rate and hence map more reads than R3-100 in total, but less correctly (in terms of edit distance) as they optimize for errors at low-quality bases. With longer reads BWA becomes less sensitive and BWA-SW might be the better choice. However, we could not compare BWA-SW as it does not align the reads from end to end. As seen before, Soap 2 is low sensitive to reads with more than 2 errors.

Memory requirement. In all-mode (best-mode), RazerS 3 requires 15 GB (9 GB) for mapping 10 M reads of length 100 bp to hg18. The memory requirement is proportional to the number of reads and matches, about 10 GB are required for each additional 10 M × 100 bp reads. For the same input set, Bowtie 2 uses 3.3 GB, BWA uses 4.5 GB, Soap 2 uses 5.4 GB, SHRiMP 2 uses 38 GB. Due to the lack of parallelization or a high memory consumption we ran mrFAST and Hobbes on packages of 500 k reads where they required 11 GB and 70 GB of memory. Section S7 contains

Table 1. Rabema results (left): Rabema scores in percent (average fraction of matches found per read). Large numbers are the total scores in each Rabema category and small numbers show the category scores separately for reads with $\left(\begin{smallmatrix}0&1&2\\3&4&5\end{smallmatrix}\right)$ errors. **Variant detection results (right):** For single-end (top) and paired-end reads (bottom) we show the percentages of found origins (recall) and fraction of unique reads mapped to their origin (precision) grouped by reads with s SNPs and i indels (s, i).

	method	all				all-best				any-best				recall			
best-mappers	Bowtie 2	92.04	99.18	98.72	96.80	96.16	97.79	97.85	95.80	98.08	100.00	99.96	97.55	95.94	98.01	97.72	95.55
			93.44	81.94	40.19		94.83	93.37	88.86		96.62	94.93	90.46		94.24	92.79	89.52
	BWA	92.18	99.18	98.72	97.81	96.81	97.79	97.87	97.88	98.81	100.00	99.95	99.81	96.41	97.93	97.69	97.25
			94.25	80.92	37.65		96.59	92.63	83.47		98.55	94.28	85.37		95.77	91.98	84.61
	Soap 2	65.93	99.18	95.55	91.34	69.89	97.79	94.74	91.37	71.37	100.00	96.78	93.18	69.91	98.05	94.62	91.20
			8.67	0.70	0.00		8.98	0.79	0.00		9.21	0.81	0.00		11.85	1.41	0.36
	R3-100	93.30	99.18	98.73	97.93	97.96	97.79	97.88	98.03	100.00	100.00	100.00	100.00	97.80	98.00	97.85	97.75
			95.60	85.81	44.15		98.00	98.27	97.93		100.00	100.00	100.00		97.65	97.70	97.69
	R3-95	93.10	99.18	98.73	97.93	97.75	97.79	97.88	98.03	99.79	100.00	100.00	100.00	97.60	98.03	97.85	97.74
			95.49	84.76	42.82		97.88	97.03	94.97		99.89	98.74	97.00		97.52	96.56	94.99
all-mappers	Bowtie 2	95.69	99.98	99.91	99.45	98.85	99.74	99.79	98.61	99.16	100.00	99.98	99.01	98.54	99.74	99.58	98.27
			97.99	90.69	55.14		98.21	97.55	93.84		98.63	97.94	94.17		97.64	96.87	94.40
	BWA	95.89	99.96	99.88	99.49	97.98	98.81	99.01	99.02	98.82	100.00	99.95	99.82	97.80	99.03	98.96	98.75
			97.13	87.79	64.11		97.83	93.95	85.20		98.56	94.34	85.37		97.35	93.43	86.36
	Hobbes	96.56	99.41	99.00	98.76	97.08	97.23	96.59	97.01	98.01	97.92	97.51	97.96	96.41	95.49	95.84	96.54
			97.80	93.20	73.05		97.38	98.16	97.42		98.43	99.12	98.46		97.03	97.98	97.79
	mrFAST	99.97	100.00	100.00	100.00	99.97	100.00	100.00	100.00	99.97	100.00	100.00	100.00	99.97	100.00	100.00	100.00
			100.00	99.99	99.53		100.00	100.00	99.10		100.00	100.00	99.13		99.99	100.00	99.18
	SHRiMP 2	96.53	99.87	99.82	99.53	99.50	99.34	99.50	99.60	99.85	99.87	99.90	99.91	99.25	99.35	99.30	99.24
			98.37	92.58	64.63		99.64	99.65	98.32		99.89	99.84	98.57		99.30	99.09	98.48
	R3-100	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
			100.00	100.00	100.00		100.00	100.00	100.00		100.00	100.00	100.00		100.00	100.00	100.00
	R3-95	99.54	100.00	100.00	100.00	99.79	100.00	100.00	100.00	99.79	100.00	100.00	100.00	99.79	100.00	100.00	100.00
			99.89	98.67	95.11		99.89	98.71	96.96		99.89	98.74	97.00		99.89	98.77	97.17

	method	(0,0)		(2,0)		(4,0)		(1,1)		(1,2)		(0,3)	
		prec.	recl.	prec.	recl.	prec.	recl.	prec.	recl.	prec.	recl.	prec.	recl.
best-mappers	Bowtie 2	97.6	97.3	94.6	92.0	92.6	82.5	95.3	93.3	93.5	92.3	96.1	95.4
	BWA	98.2	97.9	97.6	95.3	94.9	85.1	97.4	90.9	97.1	80.3	96.3	66.5
	Soap 2	98.1	82.9	97.4	31.0	0.0	0.0	90.6	6.2	0.0	0.0	0.0	0.0
	R3-100	98.4	98.4	98.2	98.2	96.3	96.3	98.1	98.1	97.9	97.9	97.6	97.6
	R3-95	98.4	98.3	98.2	97.3	96.1	91.7	98.2	97.6	97.9	97.6	97.5	97.5
all-mappers	Hobbes	99.9	99.9	99.9	99.9	100.0	100.0	100.0	99.8	100.0	93.6	99.6	90.5
	mrFAST	100.0	99.9	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
	SHRiMP 2	100.0	99.4	100.0	99.7	100.0	99.7	100.0	99.5	100.0	99.2	100.0	99.6
	R3-100	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
	R3-95	100.0	99.9	100.0	99.0	100.0	95.4	100.0	99.4	100.0	99.6	100.0	99.9

	method	(0,0)		(4,0)		(8,0)		(2,2)		(2,4)		(0,5)	
		prec.	recl.	prec.	recl.	prec.	recl.	prec.	recl.	prec.	recl.	prec.	recl.
best-mappers	Bowtie 2	98.8	98.8	98.3	96.4	100.0	92.0	98.5	97.6	98.3	97.5	98.5	98.3
	BWA	99.0	98.4	99.2	93.5	100.0	80.0	98.5	79.9	100.0	65.2	99.1	69.0
	Soap 2	98.7	95.5	98.9	53.2	0.0	0.0	97.3	53.3	96.6	46.3	98.5	79.7
	R3-100	99.0	99.0	99.0	99.0	100.0	100.0	99.7	99.7	99.6	99.6	99.0	99.0
	R3-95	99.0	98.9	99.1	96.3	100.0	88.0	99.1	97.9	100.0	99.2	98.7	98.6
all-mappers	Hobbes	97.5	93.2	98.0	94.6	100.0	100.0	96.5	80.1	99.5	86.0	97.7	85.2
	mrFAST	98.8	98.8	98.9	98.9	100.0	100.0	99.1	99.1	98.8	98.8	91.8	7.8
	SHRiMP 2	100.0	99.7	100.0	99.9	100.0	100.0	100.0	99.7	100.0	99.6	100.0	99.7
	R3-100	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
	R3-95	100.0	99.9	100.0	97.3	100.0	88.0	100.0	98.8	100.0	99.2	100.0	99.9

Table 2. Mapping times and accuracy of **single-end mapping**. The left side shows the results for the first 10 M × 100 bp single-end reads of two Illumina datasets. The dataset on the right consists of 1 M × 800 bp simulated single-end reads with a stretched Illumina sequencing error profile. Hobbes could not be run on reads longer than 100 bp. In large we show the percentage of totally mapped reads and in small the percentages of reads that are mapped with up to $\left(\begin{smallmatrix}0&1\%&2\%\\3\%&4\%\end{smallmatrix}\right)$ errors. Correctly mapped reads show the fractions of reads that were mapped with the overall minimal number of errors.

	dataset	SRR497711 D. melanogaster			ERR012100 H. sapiens			simulated, $m = 800$ D. melanogaster		
	method	time [min:s]	correctly mapped reads [%]	mapped reads [%]	time [min:s]	correctly mapped reads [%]	mapped reads [%]	time [min:s]	correctly mapped reads [%]	mapped reads [%]
best-mappers	Bowtie 2	2:00	99.65 100.00 99.77 99.02 97.65 94.92	85.71 52.08 67.27 73.62 76.88 78.81	5:37	99.62 100.00 99.75 96.02 92.88 87.86	96.72 75.99 87.81 90.54 91.85 92.76	13:48	96.73 97.47 99.67 98.05 87.95 85.17	99.99 0.03 41.07 73.95 82.31 90.03
	BWA	5:35	98.96 100.00 99.57 98.40 90.72 82.08	79.37 52.08 67.24 73.57 76.62 78.31	13:45	99.66 100.00 99.50 98.01 93.39 88.92	93.53 75.99 87.78 90.59 91.91 92.82	5:38	74.96 97.47 98.62 82.47 0.00 0.00	68.09 0.03 40.61 68.09 68.09 68.09
	Soap 2	1:55	91.78 100.00 96.24 89.35 0.09 0.02	72.49 52.08 66.73 72.48 72.49 72.49	2:34	96.45 100.00 94.94 86.54 0.32 0.16	89.73 75.99 87.24 89.73 89.73 89.73	0:54	41.21 97.47 67.99 28.10 0.22 0.00	38.14 0.03 28.17 37.88 38.14 38.14
	R3-100	1:28	100.00 100.00 100.00 100.00 100.00 100.00	78.92 52.08 67.31 73.69 76.97 78.92	85:56	100.00 100.00 100.00 100.00 100.00 100.00	92.99 75.99 87.84 90.67 92.02 92.99	1:17	100.00 100.00 100.00 100.00 100.00 100.00	90.43 0.03 41.13 74.13 82.65 90.43
	R3-95	1:26	99.87 100.00 100.00 100.00 99.11 96.34	78.82 52.08 67.31 73.69 76.94 78.82	43:16	99.96 100.00 100.00 100.00 99.30 96.92	92.95 75.99 87.84 90.67 92.01 92.95	1:15	100.00 100.00 100.00 100.00 100.00 100.00	90.43 0.03 41.13 74.13 82.65 90.43
all-mappers	Hobbes	4:51	96.49 96.55 96.46 96.94 96.28 93.86	76.16 50.28 64.98 71.16 74.33 76.16	265:48	95.97 95.94 96.14 96.39 96.10 94.63	89.24 72.90 84.30 87.02 88.33 89.24	–	–	–
	mrFAST	4:01	100.00 100.00 100.00 100.00 100.00 100.00	78.92 52.08 67.31 73.69 76.97 78.92	413:40	100.00 100.00 100.00 100.00 100.00 100.00	92.99 75.99 87.84 90.67 92.02 92.99	5:16	65.25 93.14 95.65 59.59 0.00 0.00	69.32 0.03 39.34 69.32 69.32 69.32
	SHRiMP 2	23:40	99.83 99.99 99.99 99.74 98.71 96.33	89.91 52.07 67.30 73.66 76.92 78.83	1312:09	99.81 99.89 99.83 99.39 98.29 96.81	99.06 75.90 87.74 90.56 91.91 92.87	796:06	95.70 97.47 99.75 97.60 82.95 80.14	99.31 0.03 41.04 73.67 81.63 89.14
	R3-100	1:51	100.00 100.00 100.00 100.00 100.00 100.00	78.92 52.08 67.31 73.69 76.97 78.92	118:26	100.00 100.00 100.00 100.00 100.00 100.00	92.99 75.99 87.84 90.67 92.02 92.99	1:20	100.00 100.00 100.00 100.00 100.00 100.00	90.43 0.03 41.13 74.13 82.65 90.43
	R3-95	1:45	99.87 100.00 100.00 100.00 99.11 96.34	78.82 52.08 67.31 73.69 76.94 78.82	58:13	99.96 100.00 100.00 100.00 99.30 96.92	92.95 75.99 87.84 90.67 92.01 92.95	1:20	100.00 100.00 100.00 100.00 100.00 100.00	90.43 0.03 41.13 74.13 82.65 90.43

Table 3. Mapping times and accuracy of **paired-end mapping** with the same setting as in Table 2. As datasets we used $10\,\mathrm{M} \times 2 \times 100$ bp paired-end Illumina reads (left) and $1\,\mathrm{M} \times 2 \times 800$ bp simulated paired-end reads with a stretched Illumina sequencing error profile (right). There were none of the 2×800 bp pairs without error (denoted by a "–" in the 0-error class).

	dataset	SRR497711 D. melanogaster			ERR012100 H. sapiens			simulated, $m = 800$ D. melanogaster		
	method	time [min:s]	correctly mapped pairs [%]	mapped pairs [%]	time [min:s]	correctly mapped pairs [%]	mapped pairs [%]	time [min:s]	correctly mapped pairs [%]	mapped pairs [%]
best-mappers	Bowtie 2	6:32	98.94 100.00 98.82 96.96 95.26 90.21	81.94 32.50 60.48 69.88 72.47 72.94	10:51	99.51 99.97 99.80 97.70 94.37 84.85	94.19 15.04 77.57 85.16 86.58 86.89	39:07	93.64 – 99.20 93.91 83.32 73.14	99.70 0.00 24.15 57.94 69.85 71.10
	BWA	13:33	97.47 100.00 98.48 91.02 82.51 68.36	73.41 32.51 60.41 69.30 71.57 71.92	34:35	98.84 99.99 99.66 93.72 84.75 63.84	88.06 15.04 77.50 84.86 86.16 86.39	11:26	56.28 – 95.85 49.28 0.00 0.00	46.44 0.00 23.32 40.44 40.44 40.44
	Soap 2	5:29	88.67 100.00 93.05 59.12 17.90 0.01	72.77 32.58 59.65 65.93 66.51 66.62	8:24	91.58 99.99 97.68 43.05 9.61 0.01	87.47 15.07 77.33 81.46 81.70 81.77	12:36	23.55 – 49.58 13.91 0.002 0.00	28.23 0.00 12.38 17.64 17.83 18.00
	R3-100	9:01	100.00 100.00 100.00 100.00 100.00 100.00	72.95 32.50 60.63 70.04 72.52 72.95	176:29	100.00 100.00 100.00 100.00 100.00 100.00	86.93 15.04 77.65 85.27 86.62 86.93	2:22	100.00 – 100.00 100.00 100.00 100.00	71.16 0.00 24.22 58.38 70.03 71.16
	R3-95	6:56	99.78 100.00 100.00 99.28 97.44 93.24	72.80 32.50 60.63 69.98 72.39 72.80	135:44	99.89 100.00 100.00 99.47 97.74 91.70	86.84 15.04 77.65 85.23 86.55 86.84	2:19	100.00 – 100.00 100.00 100.00 100.00	71.16 0.00 24.22 58.37 70.02 71.16
all-mappers	Hobbes	8:43	84.78 84.27 86.02 84.71 78.85 77.84	62.48 27.39 51.81 59.99 62.08 62.48	89:35	95.11 95.68 95.57 92.20 85.12 89.86	84.05 14.39 74.46 81.95 83.53 84.05	–	–	–
	mrFAST	8:26	100.00 100.00 99.99 99.99 99.99 99.98	73.16 32.50 60.63 70.04 72.52 72.95	779:12	99.94 99.98 99.96 99.82 99.56 98.73	87.79 15.04 77.64 85.26 86.61 86.91	10:47	44.19 – 91.35 27.29 0.00 0.00	49.69 0.00 24.50 43.35 43.35 43.35
	SHRiMP 2	47:07	99.67 100.00 99.93 98.65 97.39 93.03	87.36 32.50 60.62 69.95 72.48 72.93	2762:32	99.74 99.91 99.88 99.07 97.44 90.67	97.51 15.03 77.57 85.15 86.53 86.83	1617:26	91.64 – 99.35 91.81 77.75 64.58	98.62 0.00 24.12 57.14 68.89 70.27
	R3-100	7:59	100.00 100.00 100.00 100.00 100.00 100.00	72.95 32.50 60.63 70.04 72.52 72.95	184:27	100.00 100.00 100.00 100.00 100.00 100.00	86.93 15.04 77.65 85.27 86.62 86.93	2:30	100.00 – 100.00 100.00 100.00 100.00	71.16 0.00 24.22 58.38 70.03 71.16
	R3-95	7:36	99.78 100.00 100.00 99.28 97.44 93.24	72.80 32.50 60.63 69.98 72.39 72.80	166:22	99.89 100.00 100.00 99.47 97.74 91.70	86.84 15.04 77.65 85.23 86.55 86.84	2:29	100.00 – 100.00 100.00 100.00 100.00	71.16 0.00 24.22 58.37 70.02 71.16

a detailed discussion of the memory requirements of RazerS 3 and Section S12 contains tables that also show the full memory requirements. A large read set, e.g. an Illumina HiSeq run, can be mapped on clusters or low memory machines by splitting it into blocks of appropriate size and mapping them separately.

4 DISCUSSION

We presented a read mapping program that is both faster than (or at least competitive to) existing, popular tools while guaranteeing full sensitivity following a sensible and formal definition for both Hamming and edit distance. Furthermore, it allows the user to lower the sensitivity in a controlled fashion to further lower the running time. Thirdly, RazerS 3 can deal with reads of arbitrary length and large error rates.

We showed that RazerS 3 has a superior performance in the presence of sequence variations. Together with some other recent publications our work shows that the use of the BWT together with more or less exhaustive backtracking strategies has its limitations if the number of absolute indel errors is large. In addition, the above strategies do not need a precomputed index.

The banded edit distance verification algorithm presented here should also be considered as a fast algorithmic ingredient for future read mappers. Finally, our tool is able to fully leverage the in-core parallelism of modern processors.

RazerS 3 was implemented using SeqAn (Döring *et al.*, 2008) and is publicly available at http://www.seqan.de/projects/razers.

5 ACKNOWLEDGEMENTS

This work was supported by Deutsche Forschungsgemeinschaft [RE-1712/3-1 to MH]; and the Federal Ministry of Education and Research [16V0080].

REFERENCES

Ahmadi, A., Behm, A., Honnalli, N., Li, C., Weng, L., and Xie, X. (2011). Hobbes: optimized gram-based methods for efficient read alignment. *Nucleic Acids Res.*

Alkan, C., Kidd, J. M., Marques-Bonet, T., Aksay, G., Antonacci, F., Hormozdiari, F., Kitzman, J. O., Baker, C., Malig, M., Mutlu, O., Sahinalp, S. C., Gibbs, R. A., and Eichler, E. E. (2009). Personalized copy number and segmental duplication maps using next-generation sequencing. *Nat. Genet.*, **41**(10), 1061–1067.

Baeza-Yates, R. A. and Navarro, G. (1999). Faster approximate string matching. *Algorithmica*, **23**(2), 127–158.

Bauer, M. J., Cox, A. J., and Evers, D. J. (2010). ELANDv2 - fast gapped read mapping for Illumina reads. In *ISMB*. ISCB.

David, M., Dzamba, M., Lister, D., Ilie, L., and Brudno, M. (2011). SHRiMP2: sensitive yet practical short read mapping. *Bioinformatics*, **27**(7), 1011–1012.

Dohm, J., Lottaz, C., Borodina, T., and Himmelbauer, H. (2008). Substantial biases in ultra-short read data sets from high-throughput dna sequencing. *Nucleic Acids Res.*, **36**.

Döring, A., Weese, D., Rausch, T., and Reinert, K. (2008). SeqAn an efficient, generic C++ library for sequence analysis. *BMC Bioinformatics*, **9**, 11.

Gotoh, O. (1982). An improved algorithm for matching biological sequences. *J. Mol Biol.*, **162**(3), 705–708.

Hoffmann, S., Otto, C., Kurtz, S., Sharma, C. M., Khaitovich, P., Vogel, J., Stadler, P. F., and Hackermüller, J. (2009). Fast mapping of short sequences with mismatches, insertions and deletions using index structures. *PLoS Comput. Biol.*, **5**(9), e1000502.

Holtgrewe, M. (2010). Mason – a read simulator for second generation sequencing data. Technical Report TR-B-10-06, Institut für Mathematik und Informatik, Freie Universität Berlin.

Holtgrewe, M., Emde, A.-K., Weese, D., and Reinert, K. (2011). A novel and well-defined benchmarking method for second generation read mapping. *BMC Bioinformatics*, **12**, 210.

Hyyrö, H. (2003). A bit-vector algorithm for computing levenshtein and damerau edit distances. *Nord. J. Comput.*, **10**, 29–39.

Langmead, B. and Salzberg, S. L. (2012). Fast gapped-read alignment with Bowtie 2. *Nat. Methods*, **9**(4), 357–359.

Langmead, B., Trapnell, C., Pop, M., and Salzberg, S. L. (2009). Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol.*, **10**(3), R25.

Li, H. and Durbin, R. (2009). Fast and accurate short read alignment with burrows-wheeler transform. *Bioinformatics*, **25**(14), 1754–1760.

Li, H. and Homer, N. (2010). A survey of sequence alignment algorithms for next-generation sequencing. *Brief. Bioinform.*, **11**(5), 473–483.

Li, R., Yu, C., Li, Y., Lam, T.-W., Yiu, S.-M., Kristiansen, K., and Wang, J. (2009). SOAP2: an improved ultrafast tool for short read alignment. *Bioinformatics*, **25**(15), 1966–1967.

Myers, G. (1999). A fast bit-vector algorithm for approximate string matching based on dynamic programming. *J. ACM*, **46**(3), 395–415.

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

Rasmussen, K. R., Stoye, J., and Myers, E. W. (2006). Efficient q-gram filters for finding all *epsilon*-matches over a given length. *J. Comput. Biol.*, **13**(2), 296–308.

Ukkonen, E. (1985). Finding approximate patterns in strings. *J. Algorithms*, **6**(1), 132–137.

Weese, D., Emde, A.-K., Rausch, T., Döring, A., and Reinert, K. (2009). RazerS–fast read mapping with sensitivity control. *Genome Res.*, **19**(9), 1646–1654.