Approximate All-Pairs Suffix/Prefix Overlaps

Niko Välimäki[1⋆], Susana Ladra[2⋆⋆], and Veli Mäkinen[1⋆⋆⋆]

[1] Department of Computer Science, University of Helsinki, Finland.
{nvalimak,vmakinen}@cs.helsinki.fi
[2] Department of Computer Science, University of A Coruña, Spain. sladra@udc.es

Abstract. Finding approximate overlaps is the first phase of many sequence assembly methods. Given a set of r strings of total length n and an error-rate ϵ, the goal is to find, for all-pairs of strings, their suffix/prefix matches (overlaps) that are within edit distance $k = \lceil \epsilon \ell \rceil$, where ℓ is the length of the overlap. We propose new solutions for this problem based on *backward backtracking* (Lam et al. 2008) and *suffix filters* (Kärkkäinen and Na, 2008). Techniques use $nH_k + o(n \log \sigma) + r \log r$ bits of space, where H_k is the k-th order entropy and σ the alphabet size. In practice, methods are easy to parallelize and scale up to millions of DNA reads.

1 Introduction

High-throughput *short read sequencing* is revolutionizing the way molecular biology is researched. For example, the routine task of measuring gene expression by microarrays is now being replaced by a technology called *RNA-seq* [4, 27]; the transcriptome is shotgun sequenced so that one is left with a set of short reads (typically e.g. of length 36 basepairs) whose sequence is known but it is not known from which parts of the genome they were transcribed. The process is hence reversed by mapping the short reads back to the genome, assuming that the reference genome sequence is known. Otherwise, one must resort to *sequence assembly* methods [24].

The *short read mapping* problem is essentially identical to an *indexed multiple approximate string matching* problem [21] when using a proper distance/similarity measure capturing the different error types (SNPs, measurement errors, etc.). Recently, many new techniques for short read mapping have come out building on the *Burrows-Wheeler transform (BWT)* [1] and on the *FM-index* [7] concept. The FM-index provides a way to index a sequence within space of compressed sequence exploiting BWT. This index provides so-called *backward search* principle that enables very fast exact string matching on the indexed sequence. Lam et al. [13] extended backward search to simulate backtracking on *suffix tree* [28], i.e., to simulate dynamic programming on all relevant paths of suffix tree; their tool `BWT-SW` provides an efficient way to do *local alignment* without the heuristics used in many common bioinformatics tools. The

⋆ Funded by the Helsinki Graduate School in Computer Science and Engineering.
⋆⋆ Funded by MICINN grant TIN2009-14560-C03-02.
⋆⋆⋆ Funded by the Academy of Finland under grant 119815.

same idea of *backward backtracking* coupled with search space pruning heuristics is exploited in the tools tailored for short read mapping: `bowtie` [14], `bwa` [16], `SOAP2` [5]. In a recent study [17], an experimental comparison confirmed that the search space pruning heuristics used in short read mapping software are competitive with the fastest index-based filters — *suffix filters* [11] by Kärkkäinen and Na — proposed in the string processing literature.

In this paper, we go one step further in the use of backward backtracking in short read sequencing. Namely, we show that the technique can also be used when the reference genome is not known, i.e., as part of *overlap-layout-consensus* sequence assembly pipeline [12]. The overlap-phase of the pipeline is to detect all pairs of sequences (short reads) that have significant approximate overlap. We show how to combine suffix filters and backward backtracking to obtain a practical overlap computation method that scales up to millions of DNA reads.

2 Background

A *string* $S = S_{1,n} = s_1 s_2 \cdots s_n$ is a *sequence* of *symbols* (a.k.a. characters or letters). Each symbol is an element of an *alphabet* $\Sigma = \{1, 2, \ldots, \sigma\}$. A *substring* of S is written $S_{i,j} = s_i s_{i+1} \ldots s_j$. A *prefix* of S is a substring of the form $S_{1,j}$, and a *suffix* is a substring of the form $S_{i,n}$. If $i > j$ then $S_{i,j} = \varepsilon$, the empty string of length $|\varepsilon| = 0$. A *text* string $T = T_{1,n}$ is a string terminated by the special symbol $t_n = \$ \notin \Sigma$, smaller than any other symbol in Σ. The *lexicographical order* "$<$" among strings is defined in the obvious way. *Edit distance* $ed(T, T')$ is defined as the minimum number of insertions, deletions and replacements of symbols to transform string T into T' [15]. *Hamming distance* $h(T, T')$ is the number of mismatching symbols between strings T and T'.

The methods to be studied are derivatives of the *Burrows-Wheeler transform (BWT)* [1]. The transform produces a permutation of T, denoted by T^{bwt}, as follows: (i) Build the *suffix array* [19] $\mathsf{SA}[1, n]$ of T, that is an array of pointers to all the suffixes of T in the lexicographic order; (ii) The transformed text is $T^{bwt} = L$, where $L[i] = T[\mathsf{SA}[i] - 1]$, taking $T[0] = T[n]$. The BWT is reversible, that is, given $T^{bwt} = L$ we can obtain T as follows [1]: (a) Compute the array $C[1, \sigma]$ storing in $C[c]$ the number of occurrences of characters $\{\$, 1, \ldots, c-1\}$ in the text T; (b) Define the *LF mapping* as follows: $LF(i) = C[L[i]] + rank_{L[i]}(L, i)$, where $rank_c(L, i)$ is the number of occurrences of character c in the prefix $L[1, i]$; (c) Reconstruct T backwards as follows: set $s = 1$, for each $n - 1, \ldots, 1$ do $t_i \leftarrow L[s]$ and $s \leftarrow LF[s]$. Finally put the end marker $t_n \leftarrow \$$.

The *FM-index* [7] is a self-index based on the BWT. It is able to locate the interval $\mathsf{SA}[sp, ep]$ that contains the occurrences of any given pattern P without having SA stored. The FM-index uses an array C and function $rank_c(L, i)$ in the so-called *backward search* algorithm, calling the $rank_c(L, i)$ function $O(|P|)$ times. Its pseudocode is given below.

Algorithm Count($P[1 \ldots m]$,$L[1 \ldots n]$)
(1) $i \leftarrow m$;
(2) $sp \leftarrow 1$; $ep \leftarrow n$;

(3) **while** $(sp \leq ep)$ **and** $(i \geq 1)$ **do**
(4) $\quad s \leftarrow P[i]$;
(5) $\quad sp \leftarrow C[s] + rank_s(L, sp-1)+1$;
(6) $\quad ep \leftarrow C[s] + rank_s(L, ep)$;
(7) $\quad i \leftarrow i-1$;
(8) **if** $(ep < sp)$ **return** "not found"
else return "found $(ep - sp + 1)$ occurrences".

The correctness of the algorithm is easy to see by induction: At each phase i, the range $[sp, ep]$ gives the maximal interval of SA pointing to suffixes prefixed by $P[i \ldots m]$.

To report the occurrence positions $\mathsf{SA}[i]$ for $sp \leq i \leq ep$ a common approach is to sample SA values and then use the LF-mapping to derive the unsampled values from the sampled ones.

Many variants of the FM-index have been derived that differ mainly in the way the $rank_c(L, i)$-queries are solved [22]. For example, on small alphabets, it is possible to achieve $nH_k + o(n \log \sigma)$ bits of space, for moderate k, with constant time support for $rank_c(L, i)$ [8]. Here H_k is the standard *k-th order entropy*, i.e., the minimum number of bits to code a symbol once its k-symbol context is seen. There holds $H_k \leq \log \sigma$.

Let us denote by t_{LF} and t_{SA} the time complexities of LF-mapping (i.e. $rank_c(L, i)$ computation) and $\mathsf{SA}[i]$ computation, respectively.

3 All-Pairs Suffix/Prefix Matching

Given a set $\mathcal{T}$ of r strings $T^1, T^2, \ldots, T^r$, of total length n, the *exact* all-pairs suffix/prefix matching problem is to find, for each ordered pair $T^i, T^j \in \mathcal{T}$, all nonzero length suffix/prefix matches (dubbed *overlaps*). The problem can be solved in optimal time by building a generalized suffix tree for the input strings:

Theorem 1 ([9, Sect. 7.10]). *Given a set $\mathcal{T}$ of r strings of total length n, let $r*$ be the number of exact suffix/prefix overlaps longer than a given threshold. All such overlaps can be found in $O(n + r*)$ time and in $\Theta(n \log n)$ bits of space.*

In the sequel, we concentrate on approximate overlaps and more space-efficient data structures. Instead of generalized suffix trees, the following techniques use a FM-index built on the concatenated sequence of strings in $\mathcal{T}$. Since all strings T^i contain the \$-terminator as their last symbol, the resulting BWT T^{bwt} contains all r terminators in some permuted order. This permutation is represented with an array D that maps from positions of \$s in T^{bwt} to strings in $\mathcal{T}$. Thus, the string T^i corresponding to a terminator $T^{bwt}[j] = \$$ is $i = D[rank_{\$}(T^{bwt}, j)]$. The array requires $d \log d$ bits.

Next subsection introduces a basic backtracking algorithm that can find approximate overlaps within a fixed distance k. The second subsection describes a filtering method that is able to find approximate overlaps when the maximum number of errors depends on length of the overlap.

3.1 Backward Backtracking

The backward search can be extended to *backtracking* to allow the search for approximate occurrences of the pattern [13]. To get an idea of this general approach, let us first concentrate on the *k-mismatches* problem: The pattern $P_{1,m}$ approximately matches a substring $X_{1,m}$ of some string $T^i \in \mathcal{T}$, if there are at most k indices i such that $P[i] \neq X[i]$ (i.e. Hamming distance $h(P, X) \leq k$). The following pseudocode finds the k-mismatch occurrences, and is analogous to the schemes used in [14, 16]. The first call to the recursive procedure is `kmismatches`$(P, T^{bwt}, k, m, 1, n)$.

Algorithm `kmismatches`(P, L, k, j, sp, ep)
(1) **if** $(sp > ep)$ **return** ;
(2) **if** $(j = 0)$
(3) Report $\mathsf{SA}[sp], \ldots, \mathsf{SA}[ep]$; **return** ;
(4) **for each** $s \in \Sigma$ **do**
(5) $sp' \leftarrow C[s] + rank_s(L, sp - 1)+1$;
(6) $ep' \leftarrow C[s] + rank_s(L, ep)$;
(7) **if** $(P[j] \neq s)$ $k' \leftarrow k - 1$; **else** $k' \leftarrow k$;
(8) **if** $(k' \geq 0)$ `kmismatches`$(P, L, k', j - 1, sp', ep')$;

The difference between the `kmismatches` algorithm and exact searching is that the recursion considers incrementally, from right to left, all different ways the pattern can be altered with at most k substitutions. Simultaneously, the recursion maintains the suffix array interval $\mathsf{SA}[sp \ldots ep]$ where suffixes match the current modified suffix of the pattern.

To find approximate overlaps of T^i having at most k mismatches, we call `kmismatches`$(T^i, T^{bwt}, k, |T^i|, 1, n)$ and modify the algorithm's output as follows. Notice that, at each step, the range $T^{bwt}[sp \ldots ep]$ contains \$-terminators of all strings prefixed (with at most k mismatches) by the suffix $T^i_{j,m}$ where $m = |T^i|$. Thus, each of the terminators correspond to one valid overlap of length j. Terminators and their respective strings $T^{i'}$ can be enumerated from the array D in constant time per identifier; the identifiers i' to output are in the range $D[rank_\$(T^{bwt}, sp) \ldots rank_\$(T^{bwt}, ep)]$.

The worst case complexity of backward backtracking is $O(|\Sigma|^k m^{k+1} t_{\mathsf{LF}})$. There are several recent proposals to prune the search space [14, 16] but none of them can be directly adapted to this suffix/prefix matching problem.

To find all-pairs approximate overlaps, the k-mismatch algorithm is called for each string $T^i \in \mathcal{T}$ separately. Thus, we obtain the following result:

Theorem 2. *Given a set $\mathcal{T}$ of r strings of total length n, and a distance k, let $r*$ be the number of approximate suffix/prefix overlaps longer than a given threshold and within Hamming distance k. All such approximate overlaps can be found in $O(\sigma^k \sum_{T \in \mathcal{T}} |T|^{k+1} t_{\mathsf{LF}} + r*)$ time and in $nH_k + o(n \log \sigma) + r \log r$ bits of space.*

From the above theorem, it is straightforward to achieve a space-efficient and easily parallelizable solution for the exact all-pairs suffix/prefix matching problem (cf. Theorem 1):

Corollary 1. *Given a set $\mathcal{T}$ of r strings of total length n, let $r*$ be the number of exact suffix/prefix overlaps longer than a given threshold. All such overlaps can be found in $O(nt_{\mathsf{LF}} + r*)$ time and in $nH_k + o(n \log \sigma) + r \log r$ bits of space.*[3]

When *k-errors* searching (edit distance in place of Hamming distance) is used instead of k-mismatches, one can apply dynamic programming by building one column of the standard dynamic programming table [26] on each recursive step. Search space can be pruned by detecting the situation when the minimum value in the current column exceeds k. To optimize running time, one can use Myers' bit-parallel algorithm [20] with the bit-parallel witnesses technique [10] that enables the same pruning condition as the standard computation. We omit the details for brevity.

3.2 Suffix filters

We build on *suffix filters* [11] and show two different ways to modify the original idea to be able to search for approximate overlaps. Let us first describe a simplified version of the original idea using an example of approximate matching of string P with edit distance k.

Suffix filter splits the string to be searched, here P of length m, into $k+1$ pieces. More concretely, let string P be partitioned into pieces $P = \alpha_1\alpha_2 \cdots \alpha_{k+1}$. Because the FM-index is searched backwards, it is more convenient to talk about *prefix* filters in this context. Now the set of *filters* to be considered is $\mathcal{S} = \{\alpha_1\alpha_2 \cdots \alpha_{k+1},\ \alpha_1\alpha_2 \cdots \alpha_k,\ \ldots,\ \alpha_1\}$ as visualized in Fig. 1. To find *candidate* occurrences of P within edit distance k, each filter $S \in \mathcal{S}$ is matched against T as follows. We use backward backtracking (Sect. 3.1) and match pieces of the filter S starting from the last one with distance $k' = 0$. When the backtracking advances from one piece to next one (i.e. the preceding piece), the number of allowed errors k' is increased by one. Figure 1 gives a concrete example on how k' increases. If there is an occurrence of P within distance k, at least one of the filters will output it as an candidate [11]. In the end, all candidate occurrences must be validated since the filters may find matches having edit distance larger than k. However, suffix filters have been shown to be one of the strongest filters producing quite low number of wrong candidates [11].

Approximate suffix/prefix matches of $T^i \in \mathcal{T}$ can be found as follows. Instead of a fixed distance k, we are given two parameters: an *error-rate* $\epsilon \leq 1$ and a minimum overlap threshold $t \geq 1$. Now an overlap of length ℓ is called *valid* if it is within edit distance $\lceil \epsilon\ell \rceil$ and $\ell \geq t$. Again, the string T^i is partitioned into

[3] Notice that a stronger version of the algorithm in [9, Sect. 7.10] (the one using doubly-linked stacks) can be modified to find $r' < r^2$ pairs of strings with *maximum* suffix/prefix overlap longer than a given threshold. We can simulate that algorithm space-efficiently replacing doubly-linked stacks with dynamic compressed bit-vectors [18] so that time complexity becomes $O(n(t_{\mathsf{SA}} + \log n) + r')$ and space complexity becomes $nH_k + o(n \log \sigma) + r \log r + n(1 + o(1))$. We omit the details, as we focus on the approximate overlaps. A stronger variant for approximate overlaps is an open problem.

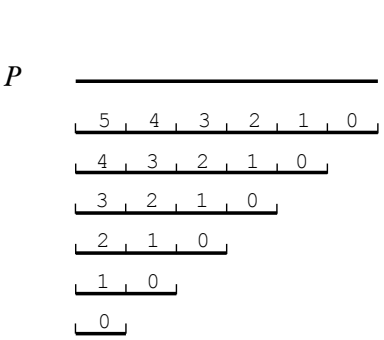

Fig. 1. Prefix filters for a string P that has been partitioned into even length pieces. Numbers correspond to maximum number of errors allowed during backward search.

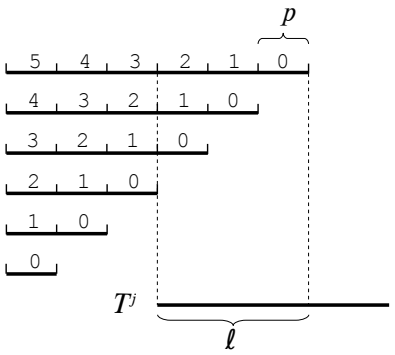

Fig. 2. String T^i has an overlap of length $\ell = 3p$ with T^j. One of the first three filters is bound to find the overlap during backward search.

pieces, denoted α_i, but now the number of pieces is determined by the threshold t and error-rate ϵ. Let $k = \lceil \epsilon t \rceil$ be the maximum number of errors allowed for the shortest overlap possible, and for simplicity, let us assume that all pieces are of even length p (to be defined later). Now the number of pieces is $h = \lceil |T^i|/p \rceil$.

Candidate overlaps are found by searching each prefix filter $S^i = \alpha_1 \alpha_2 \cdots \alpha_i$ for $1 \leq i \leq h$ separately: start the backward search from the end of the last piece α_i and match it exactly. Each time a boundary of two pieces is crossed, the number of allowed errors is increased by one. Now assume that pieces from i to jth piece have been processed, that is, the current range $[sp \ldots ep]$ corresponds to pieces $\alpha_j \alpha_{j+1} \cdots \alpha_i$. Before the backward search crosses the boundary from the piece α_j to α_{j-1}, we check the range $T^{bwt}[sp \ldots ep]$ and output those \$-terminators as candidate overlaps. These candidates are prefixes of strings in $\mathcal{T}$ that *may* be valid approximate overlaps of length $p \cdot (h - j + 1)$. Only overlaps whose lengths are multiples of the piece length p can be obtained.

We give two different strategies to find all approximate overlaps, not just those with length $p, 2p, 3p, \ldots$ But first, let us prove that the final set of candidates produced by the above method contains all valid overlaps of length pj for any $j \geq \lceil t/p \rceil$ (recall that valid overlaps must be longer than t).

Assume that there is a valid overlap of length $\ell = pj$ between T^i and some T^j, as displayed in Fig. 2. Prefix filters of T^i will locate this occurrence if we can guarantee that the suffix $T^i_{m-\ell,m}$ has been partitioned into $\lceil \epsilon \ell \rceil + 1$ pieces, where $\lceil \epsilon \ell \rceil$ gives the maximum edit distance for an overlap of length ℓ. Recall that in our partition the suffix $T^i_{m-\ell,m}$ was split into pieces of length p. We can define p as $\min_{\ell=t}^{|T^i|} \lceil \frac{\ell}{\lceil \epsilon \ell \rceil + 1} \rceil$. This guarantees that we have chosen short enough pieces for our partition, as at least one of the filters $S^h, S^{h-1}, \ldots, S^{h-j+1}$ will output the string T^j as a candidate overlap. Figure 2 illustrates this idea. In the

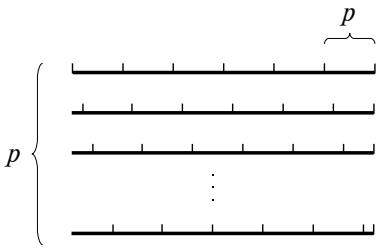

Fig. 3. Strategy I produces p different partitions of T^i.

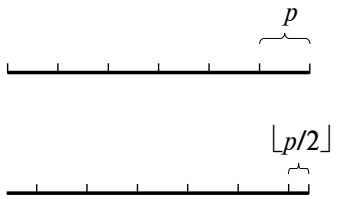

Fig. 4. Strategy II produces two different partitions of T^i.

end, all candidate overlaps must be validated since some of the candidates may not represent a valid approximate overlap.

Strategy I produces p different partitions for T^i so that the boundaries (start position of pieces) cover all indices of T^i. For simplicity, assume that $m = |T^i|$ is a multiple of p. The jth partition, $1 \leq j \leq p$, has boundaries $\{j, p + j, 2p + j, \ldots, m\}$. As a result, the very last piece "shrinks" as seen in Fig. 3. Each partition forms its own set of filters, which are then searched as described above. It is straightforward to see that filters of the jth partition find all overlaps of lengths $\ell \in \{p - j + 1, 2p - j + 1, 3p - j + 1, \ldots, m - j + 1\}$. Thus, all overlap lengths $\ell \geq t$ are covered by searching through all p partitions. Advantage of this strategy is that during the backward search, we can always match p symbols (with 0-errors) before we check for candidate matches. The "shrinking" last piece α_h can be shorter than p but it never produces candidates since $p \leq t$. Downside is that the number of different filter sets S^i to search for grows to p.

Strategy II produces only two partitions for T^i. Again, assume that $m = |T^i|$ is a multiple of p. Now the two partitions have the boundaries $\{1, p + 1, 2p + 1, \ldots, m\}$ and $\{\lceil p/2 \rceil, p + \lceil p/2 \rceil, 2p + \lceil p/2 \rceil, \ldots, m\}$, as visualized in Fig. 4. To acquire candidates for all overlap lengths $\ell \geq t$, we modify the backtracking search as follows: instead of outputting candidates only at the boundaries, we start to output candidates after $\lceil p/2 \rceil$ symbols of each piece has been matched. More precisely, assume we are matching symbol at position i' in some α_i. If $i' \leq p - \lceil p/2 \rceil$,we output all \$-terminators from range $T^{bwt}[sp \ldots ep]$ as candidate overlaps. Then the first partition outputs candidates for overlap lengths $\ell \in [\lceil p/2 \rceil, p] \bigcup [p + \lceil p/2 \rceil, 2p] \bigcup \cdots$ and the second partition for lengths $\ell \in [p + 1, p + \lceil p/2 \rceil] \bigcup [2p + 1, 2p + \lceil p/2 \rceil] \bigcup \cdots$ Since $\lceil p/2 \rceil \leq t$, these filters together cover all overlap lengths $\ell \geq t$. Obvious advantage of this strategy is that only two sets of filters must be searched. However, the number of candidates produced is generally higher than in strategy I. If p is really small, the number of candidates found after $\lceil p/2 \rceil$ symbols grows substantially.

Unfortunately, prefix filters cannot guarantee any worst-case time complexities. We conclude with the following theorem:

Table 1. Experiments with *k-mismatches*. Time is reported as average time (s) per read. Strategy II produces exactly the same overlaps as strategy I.

Method	t	k	ϵ	Time (s)	Max. ℓ	Avg. ℓ	Std.dev. ℓ
Backtracking	20	2	–	0.005	506	33.9	24.0
	20	4	–	0.277	506	27.4	16.4
	20	6	–	≈ 8	*full result not computed*		
Strategy I	20	–	5%	0.365	524	42.1	34.5
	20	–	10%	0.753	1040	46.5	38.1
	40	–	2.5%	0.212	506	74.8	45.6
	40	–	5%	0.213	524	76.7	45.7
	40	–	10%	0.553	1040	78.8	46.4
Strategy II	20	–	5%	0.140	524	42.1	34.5
	20	–	10%	0.990	1040	46.5	38.1
	40	–	2.5%	0.029	506	74.8	45.6
	40	–	5%	0.053	524	76.7	45.7
	40	–	10%	0.341	1040	78.8	46.4

Table 2. Experiments with *k-errors*. Time is reported as average time (s) per read. Strategy II produces exactly the same overlaps as strategy I.

Method	t	k	ϵ	Time (s)	Max. ℓ	Avg. ℓ	Std.dev. ℓ
Backtracking	40	2	–	0.031	535	77.2	49.4
	40	4	–	≈ 6	*full result not computed*		
Strategy I	40	–	2.5%	1.196	561	116.1	80.9
	40	–	5%	1.960	1010	121.4	82.2
	40	–	10%	≈ 6	1040	123.9	80.5
Strategy II	40	–	2.5%	0.072	561	116.1	80.9
	40	–	5%	0.179	1010	121.4	82.2
	40	–	10%	1.730	1040	123.9	80.5

Theorem 3. *Given a set $\mathcal{T}$ of r strings of total length n, a minimum overlap threshold $t \geq 1$ and an error-rate ϵ, all* approximate *overlaps within edit distance $\lceil \epsilon \ell \rceil$, where ℓ is the length of the overlap, can be found using prefix filters and in $nH_k + o(n \log \sigma) + r \log r$ bits of space.*

4 Experiments

We implemented the different techniques described in Sect. 3 on top of succinct data structures from the *libcds* library[4]. The implementation supports both the *k-mismatches* and *k-errors* (i.e. edit distance) models. Edit distance computation is done using bit-parallel dynamic programming [20]. Overlaps can be searched by using either the backtracking algorithm (for fixed k) or suffix filters (for error-rate ϵ). The experiments were run on Intel Xeon E5440 and 32 GB of memory.

[4] `http://code.google.com/p/libcds/`

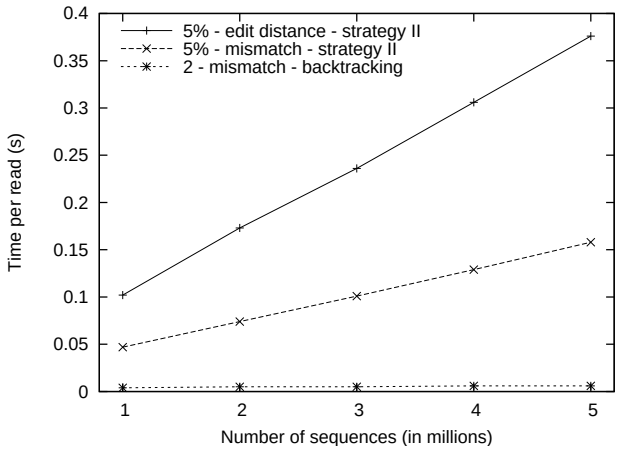

Fig. 5. Average time per read when the number of sequences increases from 1 to 5 million. The average times for $\epsilon = 5\%$ (both edit distance and mismatches) were measured using strategy II with minimum overlap length $t = 40$. All averages were measured by matching 10 000 reads against each set.

The algorithms were tested on sets of DNA sequences produced by a *454 Sequencing System* [2]. All of the DNA was sequenced from one individual *Melitaea cinxia* (a butterfly). Since the 454 system is known to produce sequencing errors in long *homopolymers* (runs of single nucleotide) [6], all homopolymers longer than 5 were truncated. The full set contained 5 million reads of total length 1.7 GB. The average read length was 355.1 with a standard deviation of 144.2. Smaller sets of 4, 3, 2, and 1 million reads were produced by cutting down the full set. Majority of these experiments were run using the smallest set of one million reads to allow extensive coverage of different parameters in feasible time.

Our implementation of the suffix filters uses extra $n \log \sigma + O(d \log \frac{u}{d})$ bits (plain sequences plus a delta-encoded bit-vector in main memory) to be able to check candidate matches more efficiently. In practice, the total size of the index for the sets of 5 and 1 million reads was 2.8 GB and 445 MB, respectively. A minimum overlap length $t \in \{20, 40\}$ was used to limit the output size. Furthermore, results were post-processed to contain only the longest overlaps for each ordered string pair.

Table 1 summarizes our results on k-mismatch overlaps for the set of one million reads. As expected, backtracking slows down exponentially and does not scale up to high values of k. The parameter $k = 4$ corresponds approximately to $0.7\% \leq \epsilon \leq 20\%$. Strategy I is faster than strategy II when the piece length gets small ($\epsilon = 10\%$ and $t = 20$). On all other parameters, however, it is more efficient to check the candidates produced by the two filters in strategy II, than to search through all partitions in strategy I. Notice that strategy II ($\epsilon = 5\%$ and $t = 40$) is only about 10 times slower than $k = 2$ but produces a significantly bigger quantity of long overlaps (cf. Fig. 6). Against $k = 4$, strategy II is on

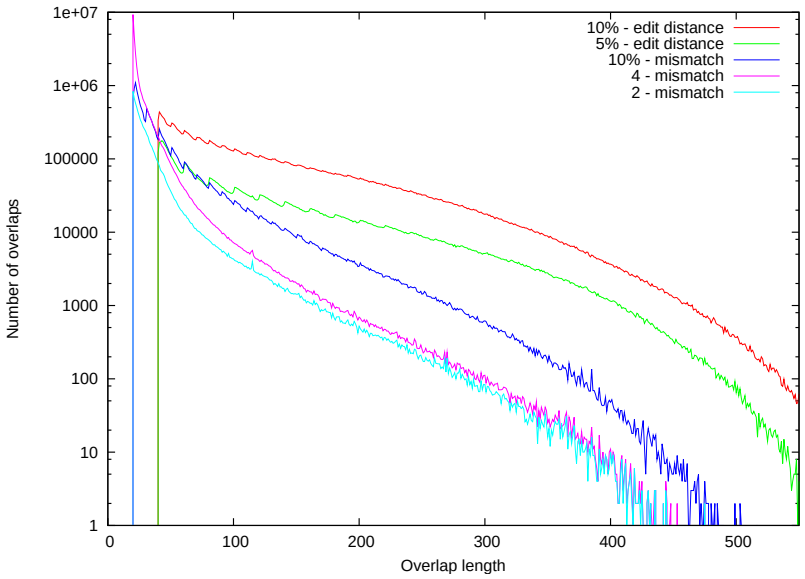

Fig. 6. Graph of overlap lengths for different error-rates ϵ and k-mismatches over a set of one million reads. The mismatch curves $\epsilon = 10\%$ and $k = 4$ cross each other at overlap lengths ℓ where $k = \lceil \epsilon \ell \rceil$. The y-axis is logarithmic.

par regarding time (when $t = 40$) and produces longer overlaps. Table 2 gives numbers for similar tests in the k-errors model.

In our third experiment, we measured the average time as a function of the number of sequences. Figure 5 gives the average times per read for backtracking with 2-mismatch and suffix filters with $\epsilon = 5\%$ and $t = 40$. The suffix filters, for both edit distance and mismatch, slow down by a factor of ≈ 3.5 between the smallest and largest set. The backtracking algorithm slows down only by a factor of ≈ 1.5.

The graph in Fig. 6 displays the frequencies of overlap lengths computed with the different k and ϵ parameters. Notice that increasing k from 2 to 4 mismatches mainly increases the number of short overlaps. Overlaps computed using error-rate give a much gentle distribution of overlaps, since they naturally allow less errors for shorter overlaps. Furthermore, at overlap lengths 100–400, the 10%-mismatch search finds about 5 times more overlaps than methods with fixed k. When searching with 10%-edit distance, there are more than a hundred times more overlaps of length 300 compared to the 2-mismatch search. This suggests that insertions and deletions (especially at homopolymers) are frequent in the dataset.

5 Discussion

Currently, many state-of-the-art sequence assemblers for short read sequences (e.g. [23, 29, 3]) use de Bruijn graph alike structures that are based on the q-grams shared by the reads. It will be interesting to see whether starting instead from the overlap graph (resulting from the approximate overlaps studied in this paper), and applying the novel techniques used in the de Bruijn appoaches, yields a competitive assembly result. Such pipeline is currently under implementation [25].

Acknowledgments

We wish to thank Richard Durbin, Jared T. Simpson, Esko Ukkonen and Leena Salmela for insightful discussions, and Jouni Sirén for implementing the bit-parallel techniques.

DNA sequences were provided by The Metapopulation Research Group/The Glanville Fritillary Butterfly Genome and Population Genomics Project: Rainer Lehtonen[5], Petri Auvinen[6], Liisa Holm[7], Mikko Frilander[8], Ilkka Hanski[5], funded by ERC (232826) and the Academy of Finland (133132).

References

1. M. Burrows and D. Wheeler. A block sorting lossless data compression algorithm. Technical Report Technical Report 124, Digital Equipment Corporation, 1994.
2. Roche Company. 454 life sciences. `http://www.454.com/`.
3. J. T. Simpson et al. Abyss: A parallel assembler for short read sequence data. *Genome Res.*, 19:1117–1123, 2009.
4. R. D. Morin et al. Profiling the hela s3 transcriptome using randomly primed cdna and massively parallel short-read sequencing. *BioTechniques*, 45(1):81–94, 2008.
5. R. Li et al. Soap2. *Bioinformatics*, 25(15):1966–1967, 2009.
6. T. Wicker et al. 454 sequencing put to the test using the complex genome of barley. *BMC Genomics*, 7(1):275, 2006.
7. P. Ferragina and G. Manzini. Indexing compressed texts. *Journal of the ACM*, 52(4):552–581, 2005.
8. P. Ferragina, G. Manzini, V. Mäkinen, and G. Navarro. Compressed representations of sequences and full-text indexes. *ACM Transactions on Algorithms (TALG)*, 3(2):article 20, 2007.

[5] Metapopulation Research Group, Department of Biological and Environmental Sciences, University of Helsinki
[6] DNA Sequencing and Genomics Laboratory, Institute of Biotechnology, University of Helsinki
[7] Institute of Biotechnology and Department of Biological and Environmental Sciences, University of Helsinki
[8] Institute of Biotechnology and Metapopulation Research Group, University of Helsinki

9. D. Gusfield. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997.
10. H. Hyyrö and G. Navarro. Bit-parallel witnesses and their applications to approximate string matching. *Algorithmica*, 41(3):203–231, 2005.
11. J. Kärkkäinen and J. C. Na. Faster filters for approximate string matching. In *Proc. ALENEX'07*, pages 84–90. SIAM, 2007.
12. J. D. Kececioglu and E. W. Myers. Combinatorial algorithms for dna sequence assembly. *Algorithmica*, 13:7–51, 1995.
13. T. W. Lam, W. K. Sung, S. L. Tam, C. K. Wong, and S. M. Yiu. Compressed indexing and local alignment of dna. *Bioinformatics*, 24(6):791–797, 2008.
14. B. Langmead, C. Trapnell, M. Pop, and S. L. Salzberg. Ultrafast and memory-efficient alignment of short dna sequences to the human genome. *Genome Biology*, 10(3):R25, 2009.
15. V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.
16. H. Li and R. Durbin. Fast and accurate short read alignment with burrows-wheeler transform. *Bioinformatics*, 2009. Advance access.
17. V. Mäkinen, N. Välimäki, A. Laaksonen, and R. Katainen. Unifying view of backward backtracking in short read mapping. In *Tapio Elomaa, Heikki Mannila, Pekka Orponen editors*, LNCS Festschrifts. Springer, 2010. To appear.
18. Veli Mäkinen and Gonzalo Navarro. Dynamic entropy-compressed sequences and full-text indexes. *ACM Transactions on Algorithms*, 4(3), 2008.
19. U. Manber and G. Myers. Suffix arrays: a new method for on-line string searches. *SIAM Journal on Computing*, 22(5):935–948, 1993.
20. G. Myers. A fast bit-vector algorithm for approximate string matching based on dynamic programming. *J. ACM*, 46(3):395–415, 1999.
21. G. Navarro. A guided tour to approximate string matching. *ACM Comput. Surveys*, 33(1):31–88, 2001.
22. G. Navarro and V. Mäkinen. Compressed full-text indexes. *ACM Computing Surveys*, 39(1):article 2, 2007.
23. P. Pevzner, H. Tang, and M. Waterman. An eulerian path approach to dna fragment assembly. *Proc. Natl. Acad. Sci.*, 98(17):9748–9753, 2001.
24. M. Pop and S. L. Salzberg. Bioinformatics challenges of new sequencing technology. *Trends Genet.*, 24:142–49, 2008.
25. L. Salmela. Personal communication, 2010.
26. P. Sellers. The theory and computation of evolutionary distances: Pattern recognition. *Journal of Algorithms*, 1(4):359–373, 1980.
27. Z. Wang, M. Gerstein, and M. Snyder. Rna-seq: a revolutionary tool for transcriptomics. *Nature Reviews Genetics*, 10(1):57–63, 2009.
28. P. Weiner. Linear pattern matching algorithm. In *Proc. 14th Annual IEEE Symposium on Switching and Automata Theory*, pages 1–11, 1973.
29. D. R. Zerbino and E. Birney. Velvet: algorithms for de novo short read assembly using de bruijn graphs. *Genome Research*, 18(5):821–829, May 2008.