Optimum Search Schemes for Approximate String Matching Using Bidirectional FM-Index

Kiavash Kianfar[1,*], Christopher Pockrandt[2,3], Bahman Torkamandi[1], Haochen Luo[1], and Knut Reinert[2,3,**]

[1] Department of Industrial and Systems Engineering, Texas A&M University
*kianfar@tamu.edu
[2] Department of Computer Science and Mathematics, Freie Universität Berlin, Germany
[3] Max Planck Institute for Molecular Genetics, Berlin, Germany
**knut.reinert@fu-berlin.de

Abstract. Finding approximate occurrences of a pattern in a text using a full-text index is a central problem in bioinformatics and has been extensively researched. *Bidirectional* indices have opened new possibilities in this regard allowing the search to start from anywhere within the pattern and extend in both directions. In particular, use of *search schemes* (partitioning the pattern and searching the pieces in certain orders with given bounds on errors) can yield significant speed-ups. However, finding *optimal search schemes* is a difficult combinatorial optimization problem.

Here for the first time, we propose a mixed integer program (MIP) capable to solve this optimization problem for Hamming distance with given number of pieces. Our experiments show that the optimal search schemes found by our MIP significantly improve the performance of search in bidirectional FM-index upon previous ad-hoc solutions. For example, approximate matching of 101-bp Illumina reads (with two errors) becomes 35 times faster than standard backtracking. Moreover, despite being performed purely in the index, the running time of search using our optimal schemes (for up to two errors) is comparable to the best state-of-the-art aligners, which benefit from combining search in index with in-text verification using dynamic programming. As a result, we anticipate a full-fledged aligner that employs an intelligent combination of search in the bidirectional FM-index using our optimal search schemes and in-text verification using dynamic programming that will outperform today's best aligners. The development of such an aligner, called FAMOUS (Fast Approximate string Matching using OptimUm search Schemes), is ongoing as our future work.

Keywords: FM index, bidirectional, read mapping, approximate matching, mixed integer programming, optimization

1 Introduction

Finding approximate occurrences of a string in a large text is a fundamental problem in computer science with numerous applications. The *approximate string matching (ASM) problem* for Hamming distance considered in this paper is defined as follows: Given a number of mismatches K, a string (here referred to as a *read*) of length R, and a text of length T, composed of characters from an alphabet of size σ, find a substring of the text whose Hamming distance to the read is at most K. A similar definition can be provided for ASM for edit distance where in addition to mismatches, insertions and deletions, are also considered.

Solving the ASM problem has become especially important in bioinformatics due to the advances in sequencing technology during the last years. The mainstream second generation sequencing techniques like Illumina produce reads of length 150-250 with an error rate of about 1%, mostly substitutions caused by the sequencing technology. Other sequencing technologies, e.g., Pacific Bioscience or Oxford Nanopore, produce much longer reads but with a higher error rate (in the range of 15%) containing both substitutions and insertions/deletions. A standard problem is to map the reads back to a reference genome while taking into account the errors introduced by the sequencing technology as well as those caused by biological variation, such as SNPs or small structural variations. Such a problem is almost always modeled as the ASM problem for Hamming or edit distance.

There are two main algorithmic strategies to address the ASM problem for large input sizes (in number of reads and size of the text): *filtering* and *indexing*. In this work, we focus on the indexing approach. Here, the main idea is to preprocess the reference sequence, the set of reads, or both, in a more intricate way. Such preprocessing into *full-text string indices* has the benefit that we usually do

not have to scan the whole reference, but can conduct queries much faster at the expense of larger memory consumption. String indices that are currently used are the suffix array [12] and enhanced suffix array [1], and affix arrays [11, 18], as well as the FM-index [3], a data structure based on the Burrows-Wheeler Transform (BWT) [2] and some auxiliary tables. For an in-depth discussion see [16]. Such indices are used to compute exact matches between a query and a text as a subroutine in backtracking approaches. For the ASM problem for Hamming or edit distance, the existing algorithms all have exponential complexity in K (e.g. [5, 19]), and are hence only suited for small K.

Lam et al. [7] introduced *bidirectional* FM indices to speed up ASM for Hamming distance. For the cases $K = 1$ and 2, they partitioned the read into $K + 1$ equal pieces, and argued that performing approximate matching on a certain combination of these pieces in a bidirectional index amounts to *faster* approximate matching of the whole read. This combination is such that all possible *mismatch patterns*, i.e., all possible distributions of K mismatches among the pieces, are covered. The main idea behind improved speed is that a bidirectional index not only can start the search from the beginning (or end) of the read, but also from the beginning (or end) of any of the pieces. Therefore, we can start the search from a middle piece and then expand it to the left or right into adjacent pieces in any order we like. By choosing multiple appropriate orderings of pieces for this purpose, we can perform a much faster ASM compared to a unidirectional search because we can enforce exact or near-exact searches on the first pieces in the partition, significantly reducing the number of backtrackings, while using different orderings of pieces to ensure all possible mismatch patterns are still covered.

Kucherov et al. [6] formalized and generalized this idea by defining the concept of *search schemes*. Assume a read can be partitioned into a given number of pieces, denoted by P (not necessarily equal to $K+1$). The pieces are indexed from left to right. A search scheme $\mathcal{S} = \{(\pi_s, L_s, U_s), s = 1, \ldots, S\}$ is a collection of S searches, where each search s is designated by a triplet (π_s, L_s, U_s). π_s is a permutation of $1, \ldots, P$ and denotes the order in which the pieces of the partition are searched in search s. If $\pi_{s,i} = j$, then piece j is searched at position i in the order (shortly referred to as *iteration* i in this paper). Due to the way a bidirectional index works, the permutation π_s must satisfy the so-called *connectivity* condition, i.e, a piece j can appear at iteration $i > 1$ in the permutation only if at least one of pieces $j - 1$ or $j + 1$ have appeared at an iteration before i. L_s and U_s each are strings of P numbers. $L_{s,i}$, is the lower bound on the cumulative number of mismatches allowed at iteration i of search s, and $U_{s,i}$ is the upper bound on this value.

Having this formal framework, the answer to the *Optimal Search Scheme* problem, defined as follows, can potentially have a great impact on improving the running time of ASM using a Bidirectional index (ASM-B).

Optimal Search Scheme Problem: *What is the search scheme that minimizes the number of steps in ASM-B while ensuring all possible mismatch patterns are covered?*

It turns out that this is a very difficult combinatorial optimization problem due to several reasons: There are a large number of attributes that define a solution (including S, P, size of each piece, and (π_s, L_s, U_s) for each search) with a large number of possibilities for each attribute; the solution must satisfy complex combinatorial constraints; and, calculating the objective function, i.e., number of steps in the ASM-B algorithm, for a given solution is complicated.

Kucherov et al. [6] presented some interesting results contributing initial insight into this key problem. More specifically, they assumed the number of steps in the ASM-B algorithm with a given search scheme, is a constant factor of the (weighted) total number of substrings enumerated by the algorithm in all searches. Assuming that a randomly generated read is to be matched to a randomly generated text, they presented a method to calculate this objective function for a given search scheme. They then showed that unequal pieces in the partition can potentially improve the objective function compared to equal pieces, and presented a dynamic programming (DP) algorithm that for a single

prespecified search, with given P and (π, L, U), finds the optimal sizes of pieces assuming that we only calculate the objective function as the total number of substrings up to a limited length (justified by total randomness of the read and the text); see [6] for more details. In fact, the superiority of this DP over explicit enumeration is only due to this assumption. Nevertheless, this DP is very inefficient, and most importantly, it only finds the optimal piece sizes for a *prespecified* search. In other words, it does not address the problem of finding an optimal search scheme which calls for determining S and all attributes of each search in the search scheme, and ensuring that they cover all mismatch patterns.

Kucherov et al. [6] also presented solutions for another limited problem, i.e., lexicographically minimizing the lexicographically maximal U string (critical U string) in a search scheme, only for $P = K+1$ or $K+2$ and assuming that the L strings for all searches contain only zeros. The usefulness of these solutions is justified by the high probability that the search with the critical U string has the largest share in the objective function; see [6] for details. Again from the perspective of finding an optimal search scheme, this result has similar limitations. Only one of the attributes (U) of one of the searches for two specific values of P are optimized by fixing all L strings, which is far from designing a globally optimal search scheme as defined above. Consequently, in their computational experiments, Kucherov et al. [6] use a greedy algorithm based on this limited result to construct search schemes with unknown quality and only optimize the piece sizes for these schemes using their DP.

In this paper, for the first time, we propose a method to solve the optimal search scheme problem for ASM-B with Hamming distance, for any given P and equal-size pieces. Our method is based on a novel and powerful mixed integer linear program (MIP) that gets K, R, P, and an upper bound on S, denoted by $\overline{S}$, as input, and provides, as its solution, all the attributes of the exact optimal search scheme (MIP methodology for optimization has been addressed in many references such as [13, 20]). We present our MIP and provide the results of our computational study on the characteristics of its optimal solution and its running time, for different values of its input parameters.

Next we conduct an experiment using a bidirectional search (for Hamming and edit distance) that performs the search based on the optimal search schemes we have obtained from our MIP. This bidirectional index search is implemented in SeqAn [15] and uses a recent fast implementation of bidirectional indices [14] based on EPR dictionaries. We show that, for practical ranges of various input parameters, the number of substrings for the optimal search schemes found by our MIP can reduce to as small as half the number of substrings in the unidirectional complete backtracking. In another experiment, we search for all occurrences of Illumina reads in the human genome completely in the bidirectional index using our optimal search schemes and show that our time is is up to 3, 14, and 35 times faster than the standard backtracking search in the index for $K = 1$, 2, and 3 Hamming distance errors, respectively. Although our MIP finds the optimal search schemes for Hamming distance, when we used its optimal schemes with the edit distance, we got similar improvements.

The drastic improvement over standard backtracking gained by using our optimal search schemes for bidirectional search in index suggests that the performance of read mappers that utilize an index can be significantly improved. To gauge this potential, we even challenged our optimal search schemes by performing a pure index-based search using them and comparing the performance with the full-fledged state-of-the-art aligners that benefit from using a combination of search in index and verification in text using dynamic programming. We noticed that our optimal schemes are so effective that although they are employed in the pure index-based search, the resulting times for $K = 1$, 2, and 3, are competitive with the state-of-the-art aligners which use a combination of index search and verification in text, in strata mode. In the all-mapping mode, the results are better for $K = 1$, and competitive for $K = 2$. These observations suggest that a full-fledged aligner that employs an intelligent combination of search in the bidirectional FM-index using our optimal search schemes and verification in text using dynamic programming can outperform today's best approximate read

mappers. The development of such an aligner, called FAMOUS (Fast Approximate string Matching using OptimUm search Schemes) is ongoing in our group as future work.

We will introduce our MIP for solving the optimal search scheme problem in Section 2 and discuss our computational studies on solving this MIP. Then in Section 3, we present the computational impact of using the optimal search schemes obtained from the MIP on solving ASM-B using various realistic scenarios. In Section 4, we will motivate the potential impact of our optimal search schemes in developing a full-fledged read mapper and mention FAMOUS as our future work. We will conclude in Section 5 by summarizing our contributions and raising several open problems and possible extensions.

2 Solving Optimal Search Scheme Problem using MIP

In this section, we provide our MIP-based methodology for finding optimal search schemes after presenting some preliminaries. We will then follow with a brief computational report on solving our MIP in order to find the optimal search schemes, including its optimal objective value as a function of its input parameters, its solution running time, and its convergence rate to optimal solution).

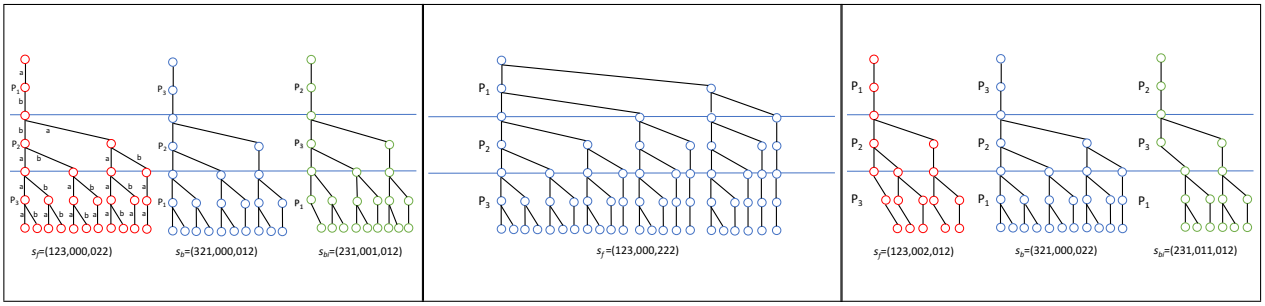
Our MIP is for Hamming distance, but as mentioned before, based on our computational experiments (Section 4), its optimal schemes for Hamming distance are very good (but not necessarily optimal) search schemes for the edit distance as well.

2.1 Preliminaries

Our MIP presented in Section 2.2 will solve the optimal search scheme problem assuming P is given as an input (is not a decision variable in optimization) and all P pieces of the partition are equal in length, i.e., $R = mP$, where m denotes the length of any piece. Note that these assumptions pose no practical restrictions. Solving problems which include P as a decision variable and allow unequal pieces is part of our future research plan. Given the upper bound on Hamming distance K (maximum number of mismatches) as an input, a *mismatch pattern* is a particular distribution of h mismatches among the P pieces, for any $h \leq K$. Specifically, the mismatch pattern q is a string of P integers $a_{q,1} \dots a_{q,P}$ such that $a_{q,j} \in \{0, \dots, \min\{m, K\}\}$ for $j = 1, \dots, P$, and $\sum_{j=1}^{P} a_{q,j} = h$. For given K and P, we denote the set of all possible mismatch patterns by $\mathcal{M}$. Note that if $K \leq m$ then $|\mathcal{M}| = \sum_{h=0}^{K} \binom{h+P-1}{h}$. Given a search $s = (\pi_s, L_s, U_s)$, a mismatch pattern q is said to be *covered* by s if at every iteration $i = 1, \dots, P$ of s, $L_{s,i} \leq \sum_{t=1}^{i} a_{q,\pi_{s,t}} \leq U_{s,i}$, i.e., the cumulative number of mismatches up to iteration i is between the allowed lower and upper bounds of search s. A search scheme $\mathcal{S}$ is feasible if and only if every mismatch pattern in $\mathcal{M}$ is covered by at least one search in $\mathcal{S}$.

A search scheme can be visualized by representing each of its searches as a trie that captures all substrings enumerated by the search. Each edge at a level of the trie corresponds to a character of the alphabet at that level of search. A vertical edge represents a match, and a diagonal edge represents a mismatch. Fig. 1(a) shows the tries associated with the search scheme presented by Lam et al. [7] for $K = 2$ and $P = 3$, $\mathcal{S}_{Lam}$, applied on the six-character read "abbaaa" from alphabet $\{a, b\}$ (note that the tries are slightly different from the ones given in [6], which contained a small error). Fig. 1(b) shows a search scheme with a single unidirectional search (complete backtracking), $\mathcal{S}_{Uni}$, for the same problem, and Fig. 1(c) shows the optimal search scheme, $\mathcal{S}_{Opt}$, found by our MIP, for the same problem. Each one of the three schemes in Fig. 1 covers all 10 mismatch patterns, namely $\{000, 001, 010, 100, 011, 101, 110, 002, 020, 200\}$. Interestingly, the three searches s_f, s_b, s_{bi} in $\mathcal{S}_{Opt}$ cover the mismatch patterns $\{002, 011\}$, $\{000, 010, 100, 110, 020, 200\}$, and $\{001, 101\}$, respectively, which is indeed a partition of all mismatch patterns (see open problems in Section 5), whereas in $\mathcal{S}_{Lam}$, the searches s_f and s_b both cover 000 and 010 redundantly.

Following the method of Kucherov et al. [6], we define the performance of a search scheme as the number of forward and backward steps taken by the ASM-B algorithm, which is equal to the total



(a) (b) (c)

Fig. 1. (a) The search of Lam et al. [7] as described by Kucherov [6] for $K = 2$ and $P = 3$, i.e., $\mathcal{S}_{Lam} = \{s_f = (123, 000, 022), s_b = (321, 000, 012), s_{bi} = (231, 001, 012)\}$, shown for the read "abbaaa" from the alphabet {a,b}, i.e., $R = 6$ and $\sigma = 2$. The read is partitioned into P_1 =ab, P_2 =ba, and P_3 =aa. Partition borders are shown by horizontal lines. A vertical and a diagonal edge represent a match and a mismatch, respectively. Edge labels are only shown for s_f for a cleaner picture. The search corresponding to each trie is designated underneath it by its (π, L, U). The number of edges in $\mathcal{S}_{Lam}$ tries is 71. **(b)** The unidirectional search scheme $\mathcal{S}_{Uni} = \{s_f = (123, 000, 222)\}$ for the same problem. The number of edges in $\mathcal{S}_{Uni}$ is 62, i.e., for this particular problem, in which R is very small, $\mathcal{S}_{Lam}$ enumerates even more substrings than $\mathcal{S}_{Uni}$ (if all possible substrings are present in the text). Of course, if R gets larger, the situation is reversed, making $\mathcal{S}_{Lam}$ more efficient than $\mathcal{S}_{Uni}$ as reported by Lam et al. [7]. **(c)** The optimal search scheme $\mathcal{S}_{Opt} = \{s_f = (123, 002, 012), s_b = (321, 000, 022), s_{bi} = (231, 011, 012)\}$ for the same problem, found by our MIP. The total number of edges in $\mathcal{S}_{Opt}$ (optimal number of edges) is 59, which is less than that of $\mathcal{S}_{Uni}$, and significantly less than that of $\mathcal{S}_{Lam}$. As shown in Section 4, for bigger problems, the reduction in the total number of edges of the optimal search scheme found by our MIP compared to the unidirectional search is much more significant (up to 50%).

number of substrings enumerated by all searches in the scheme. We assume a single step of forward or backward search in the bidirectional index takes the same amount of time. The tries of any search scheme in Fig. 1 contain all possible substrings of length R. The number of substrings in each trie is equal to the number of edges (or total number of non-root nodes). If the text contains all substrings of length R, the search enumerates all substrings in the tries; hence, the performance of the search scheme can be measured by the total number of edges in the search scheme. Otherwise, only a subset of the substrings in the tries will be enumerated depending on whether they occur in the text or not. To address the performance measure in this latter case, Kucherov et al. [6] assumed the read and the text are randomly and independently drawn from the alphabet according to a uniform distribution, and hence, calculated the expected number of substrings enumerated by the scheme as the sum, over all non-root nodes of the tries, of the probability that the corresponding substring appears in the text. As a result, they presented a *weighted* sum of number of edges as the measure of performance. Due to the assumption of complete randomness and independence of the read and the text, they show that the weights of the edges at levels lower than $\lceil log_\sigma T \rceil + c_\sigma$ of the tries, where c_σ is that $((\sigma - 1)/\sigma)^{c_\sigma}$ is sufficiently small, are almost zero meaning that they can be dropped from the weighted summation.

For the main application of our interest, i.e. ASM of DNA sequence reads to reference genomes, the assumption of randomness and independence of the read and the text is far from reality. Calculating the expected number of substrings enumerated by a scheme calls for significant more study on determining probabilities that DNA sequence reads of particular length from a sample occur in the reference genomes. As currently there is no trivial answer to this problem, in this paper, we use the same performance measure of total number of edges in the tries of the search scheme even for the case where not all substrings occur in the text. Of course, our MIP can be easily modified to incorporate any other weighting scenario which might be proposed in the future.

Adapting the method from [6], the total number of edges in the search scheme is calculated by

$$\sum_{s=1}^{S} \sum_{l=1}^{R} \sum_{d=0}^{K} n_{s,l,d}, \tag{1}$$

where $n_{s,l,d}$ is defined as the number of edges at level l of the trie of search s that end at nodes corresponding to substrings with d cumulative mismatches up to that level. The value of $n_{s,l,d}$ can be

calculated using the following recursive equation, which is an adaptation of the formula in [6]:

$$n_{s,l,d} = n_{s,l-1,d} + (\sigma-1)n_{s,l-1,d-1} \qquad \text{for } l \geq 1 \text{ and } \mathscr{L}_{s,l} \leq d \leq \mathscr{U}_{s,l}, \tag{2}$$

where, by definition, $n_{s,0,0} = 1$, $n_{s,0,-1} = 0$ and $n_{s,0,d} = 0$, for $d \geq 1$, $s = 1, \ldots, S$, and $\mathscr{L}_{s,l}$ and $\mathscr{U}_{s,l}$ denote the smallest and largest cumulative number of mismatches that can occur at level l of the trie of search s, respectively, calculated as $\mathscr{L}_{s,l} = \max\{L_{s,\lceil l/m\rceil-1}, L_{s,\lceil l/m\rceil} - m\lceil l/m\rceil + l\}$ and $\mathscr{U}_{s,l} = \min\{U_{s,\lceil l/m\rceil}, \mathscr{U}_{s,l-1} + 1\}$. Here $\lceil l/m\rceil$, the smallest integer greater than or equal to l/m, would be the index of the iteration in which level l falls, and by definition, $\mathscr{L}_{s,0} = \mathscr{U}_{s,0} = 0$, for $s = 1, \ldots, S$. For example, for search s_{bi} of $\mathcal{S}_{Opt}$, we have $\mathscr{L}_{s_{bi}} = (0,0,0,1,1,1)$ and $\mathscr{U}_{s_{bi}} = (0,0,1,1,2,2)$.

2.2 MIP Formulation of Optimal Search Scheme Problem

Our MIP formulation, presented below, solves the optimal search scheme problem assuming P is given as an input and pieces are all equal in length. More specifically, for given K, R, P, and $\overline{S}$, this MIP finds the search scheme with minimum total number of edges among all feasible search schemes that have at most $\overline{S}$ searches. The optimal solution to the MIP provides the (π, L, U) of all searches in the optimal search scheme. The objective value of this optimal solution provides the minimum total number of edges (substrings) achievable among all feasible search schemes.

$$\min \quad \sum_{s=1}^{\overline{S}} \sum_{l=1}^{R} \sum_{d=0}^{K} n_{s,l,d} \tag{3}$$

subject to

$$\sum_{i=1}^{P} x_{s,i,j} = 1 \qquad \text{for all } s \text{ and } j \tag{4a}$$

$$\sum_{j=1}^{P} x_{s,i,j} = 1 \qquad \text{for all } s \text{ and } i \tag{4b}$$

$$\sum_{h=1}^{i} x_{s,h,j} - \sum_{h=1}^{i} x_{s,h,j-1} = t^{+}_{s,i,j} - t^{-}_{s,i,j} \qquad \text{for all } s,\ i = 2, \ldots, P-1,\ j = 1, \ldots, P+1 \tag{5a}$$

$$\sum_{j=1}^{P+1} (t^{+}_{s,i,j} + t^{-}_{s,i,j}) = 2 \qquad \text{for all } s,\ i = 2, \ldots, P-1 \tag{5b}$$

$$d - (L_{s,\lceil l/m\rceil} - m\lceil l/m\rceil + l) + 1 \leq (R+1)\underline{z}_{s,l,d} \qquad \text{for all } s,\ l, \text{ and } d \tag{6a}$$

$$U_{s,\lceil l/m\rceil} + 1 - d \leq (K+1)\overline{z}_{s,l,d} \qquad \text{for all } s,\ l, \text{ and } d \tag{6b}$$

$$\binom{l}{d}(\sigma-1)^d(\overline{z}_{s,l,d} + \underline{z}_{s,l,d} - 2) \leq n_{s,l,d} - n_{s,l-1,d} - (\sigma-1)n_{s,l-1,d-1} \qquad \text{for all } s,\ l, \text{ and } d \tag{6c}$$

$$L_{s,i} \leq L_{s,i+1} \qquad \text{for all } s, \text{ and } i = 1, \ldots, P-1 \tag{7a}$$

$$U_{s,i} \leq U_{s,i+1} \qquad \text{for all } s, \text{ and } i = 1, \ldots, P-1 \tag{7b}$$

$$L_{s,i} + K(\lambda_{q,s} - 1) \leq \sum_{h=1}^{i} \sum_{j=1}^{P} a_{q,j} x_{s,h,j} \leq U_{s,i} + K(1 - \lambda_{q,s}) \qquad \text{for all } q,\ s, \text{ and } i \tag{8a}$$

$$\sum_{s=1}^{\overline{S}} \lambda_{q,s} \geq 1 \qquad \text{for all } q \tag{8b}$$

$$n_{s,l,d} \geq 0 \qquad \text{for all } q,\ s,\ i,\ j,\ l, \text{ and } d \tag{9a}$$

$$L_{s,i}, U_{s,i} \geq 0 \quad \text{Integer} \qquad \text{for all } s \text{ and } i \tag{9b}$$

$$x_{s,i,j}, \lambda_{q,s}, \overline{z}_{s,l,d}, \underline{z}_{s,l,d}, t^{+}_{s,i,j}, t^{-}_{s,i,j} \in \{0,1\} \qquad \text{for all } q,\ s,\ i,\ j,\ l, \text{ and } d \tag{9c}$$

The objective function (3) minimizes the total number of edges as calculated by (1) with $n_{s,l,d}$ as defined before. The binary variables $x_{s,i,j}$ capture the assignment of pieces to iterations, i.e., $x_{s,i,j} = 1$ if piece j is searched at iteration i of search s, and $x_{s,i,j} = 0$ otherwise. We define $x_{s,i,0} = x_{s,i,P+1} = 0$ to simplify presentation of constraints. At optimality, these variables determine the π_s values for the optimal search scheme. Constraints (4a) and (4b) make sure that for any search s, only one piece is assigned to an iteration and only one iteration is assigned to a piece.

Constraints (5a)-(5b) ensure the connectivity of the pieces and are in fact linearization of the following constraint using auxiliary binary variables $t^{+}_{s,i,j}$ and $t^{-}_{s,i,j}$:

$$\sum_{j=1}^{P} \left| \sum_{h=1}^{i} x_{s,h,j} - \sum_{h=1}^{i} x_{s,h,j-1} \right| = 2 \qquad \text{for all } s \text{ and } i = 2, \ldots, P-1, \tag{10}$$

which is one way to enforce connectivity of pieces. The term $\sum_{h=1}^{i} x_{s,h,j}$ will have a binary value which denotes whether or not piece j has been searched at any of iterations 1 to i of search s. The term $\sum_{h=1}^{i} x_{s,h,j-1}$ captures the same notion for piece $j-1$. If at any iteration all searched pieces form a connected block on the read, the value of $\sum_{h=1}^{i} x_{s,h,j} - \sum_{h=1}^{i} x_{s,h,j-1}$ will be equal to 1 only for one j, -1 for another j, and 0 for all other j's, which is ensured by (10), and hence its linearization.

Constraints (6a)-(6c) enforce calculation of $n_{s,l,d}$ based on the recursive equation (2) with the help of binary variables $\underline{z}_{s,l,d}$ and $\overline{z}_{s,l,d}$. Due to (6a), if $d \geq L_{s,\lceil l/m \rceil} - m\lceil l/m \rceil + l$, then $\underline{z}_{s,l,d} = 1$, and due to (6b), if $d \leq U_{s,\lceil l/m \rceil}$, then $\overline{z}_{s,l,d} = 1$. Calculation of equation (2) is then enforced by (6c). When $\underline{z}_{s,l,d} = \overline{z}_{s,l,d} = 1$, (6c) reduces to $n_{sld} - n_{s,l-1,d} - (\sigma - 1)n_{s,l-1,d-1} \geq 0$, which implies $n_{sld} - n_{s,l-1,d} - (\sigma - 1)n_{s,l-1,d-1} = 0$ since the objective function is to be minimized. If any of $\underline{z}_{s,l,d}$ or $\overline{z}_{s,l,d}$ is equal to 0, (6c) does not enforce anything as $-\binom{l}{d}(\sigma - 1)^d$ is a lower bound on the right-hand side of (6c). Constraints (7a)-(7b) ensure $L_{s,i}$ and $U_{s,i}$ are non-decreasing as they are cumulative values. Constraints (8a)-(8b) ensure feasibility of the search scheme. $\lambda_{q,s}$ is a binary variable designating whether or not mismatch pattern q is covered by search s. Constraint (8a) forces $\lambda_{q,s} = 0$ if search s does not cover mismatch pattern q and constraint (8b) ensures every mismatch pattern q is covered by at least one search, for $q = 1, \ldots, |\mathcal{M}|$.

Constraints (4a)-(9c) are enough to formulate the MIP; however, we have noticed that imposing the additional constraints

$$x_{1PP} = 1 \tag{11a}$$

$$\sum_{t=s}^{\overline{S}} \sum_{k=1}^{j-1} x_{t,1,k} \leq (\overline{S} - s + 1)(1 - x_{s,1,j}) \qquad \text{for all } s \text{ and } j = 2, \ldots, P \tag{11b}$$

$$\sum_{j=1}^{P-i+1} x_{sij} + \sum_{j=i}^{P} x_{sij} = 1 \qquad \text{for all } s \text{ and } i \geq \lceil P/2 \rceil + 1 \tag{12}$$

strengthens the formulation while preserving at least one optimal solution, resulting in faster solution time for the MIP. Constraints (11a) and (11b) eliminate some symmetry in the solution space. For every search scheme, there is an equivalent search scheme obtained by reversing all π_s, $s = 1, \ldots, \overline{S}$. Constraint (11a) eliminates one of these two equivalent solutions in each pair by forcing piece P to be assigned to iteration P in the first search, eliminating the solutions in which piece 1 is assigned to iteration P. For any search scheme, another equivalent search scheme can be obtained by permuting the indices of searches within the scheme. Existence of only one of the search schemes obtained by this index permutation in the feasible solution set is enough. This can be achieved by sorting (in ascending order) the searches based on the piece assigned to their first iteration. This is done by constraint (11b), which does not allow pieces $1, \ldots, j-1$ to be assigned to the first iteration of searches $s, \ldots, \overline{S}$ if piece j is assigned to the first iteration of search s. In addition to symmetry elimination, notice that the connectivity condition of pieces implies that the piece assigned to iteration P is either piece 1 or piece P, and in general, the piece assigned to iteration $i \geq \lceil P/2 \rceil + 1$ is one of pieces $1, \ldots, P-i+1, i, \ldots, P$. Constraint (12) enforces this property, which strengthens the formulation, and according to our computational tests, reduces the running time of the MIP.

Remark 1. A powerful feature of our MIP is that $\overline{S}$ is an upper bound on the number of searches, i.e., our MIP finds the optimal search scheme among all schemes with at most $\overline{S}$ searches. In our MIP, variables are defined for $\overline{S}$ searches, and if the optimal search scheme has $S^* < \overline{S}$ searches, our MIP generates $\overline{S} - S^*$ *empty* searches, i.e. searches in which $L_{s,i} > U_{s,i}$ for some i. □

2.3 Solving MIP

We used CPLEX 12.7.1 solver [4] to solve our MIP by implementing the code in C++ using CPLEX Callable Library. All instances were run over four 28-core nodes (2.4 GHz Intel Broadwell) with 64GB of memory per node. We ran our MIP solver for instances generated for a broad range of parameters K, R, $\overline{S}$, and P and gave each instance a 3-hour time limit. Fig. 2(a) in the Appendix is a small representative of our results. It shows the optimal objective value (total number of edges) for $R = 100$, $K = 1, \ldots, 4$, $P = 5, 6$, and $\overline{S} = 1, \ldots, 5$. If the problem is not solved to optimality in 3 hours, the best solution found within this time limit is shown. The optimal objective value does not show a consistent change pattern in terms of change in P; however, as expected, it increases as K increases, as R increases (not shown), and as $\overline{S}$ decreases. In all instances, the optimal objective value shows a sharp drop from $\overline{S} = 1$ to $\overline{S} = 2$, then a modest drop to $\overline{S} = 3$, and negligible change beyond $\overline{S} = 3$, generating empty searches in many cases. Therefore, as long as $\overline{S} = 5$, it is advisable to use $\overline{S} = 3$ instead if we would like to reduce the MIP running time and still find an optimal or near-optimal solution for $\overline{S} = 5$. We also noticed that the optimal search scheme obtained by our MIP is not sensitive to the value of R (see open problems in Section 5). Therefore, when R is large, it is advisable to solve the MIP for a much smaller reasonable value of R, e.g., $R = KP$, in order to get a solution that is most probably optimal for the large R in a much shorter amount of time.

Using the MIP formulation, we were able to solve considerable size problems to optimality. For instance, we were able to solve a problem with $K = 4$, $R = 100$, $P = 3$, and $\overline{S} = 3$ to optimality in 5802 seconds. However, more complicated cases reached the time limit of 3 hours without proving solution optimality. Consequently, it is important to investigate the rate of convergence of the solutions found during execution of MIP to the optimal solution. Fig. 2(b) illustrates the ratio of the best solutions found by MIP during its execution to the final optimal objective value plotted against running time for some instances which reached optimality. We observe that in all cases, within 0.1% to 1% of the total running time, the MIP finds a solution which is finally proved to be optimal or very close to the optimal after MIP execution is complete. In other words, the MIP solver finds optimal or near optimal solutions very early on and spends the rest of its time ensuring that no better solution exists. This can be partly due to the remaining symmetry in the solution space. Nevertheless, from practical perspective, this is an attractive property because, when the input parameters are much larger, we can run the MIP for a short time and find solutions which are most probably optimal or near-optimal.

3 Search-in-Index Computational Gains Achieved by Optimum Schemes

In this section, we present the computational advantages achieved by using our optimal search schemes in ASM-B (ASM performed completely in bidirectional FM-index).

While our optimal search schemes can be found for any alphabet size and read length, we chose to concentrate on parameter values relevant to standard sequencing reads, e.g., Illumina reads. In Table 1, for a number of relevant parameter values, we have shown how the total number of edges using the optimal search schemes found by our MIP is reduced compared to the unidirectional backtracking scheme. It can be seen that the reduction is between 42% and 49%. Also, for $K = 2$ and $K = 3$, the optimal search scheme with $P = K + 2$ has fewer edges than the one with $P = K + 1$.

Although the reduction factors in total number of edges obtained by our optimal search schemes in Table 1 are very significant in themselves, due to the stochastic nature of occurrence of errors in sequencing reads and occurrence of approximate matches in the reference genome, the real-case ASM speed-up factors achieved by these optimal search schemes compared to backtracking can be yet much more significant. To gain insight into this speed-up, we performed an experiment searching for *all* approximate matches (for $K = 1$, 2, and 3) of $100,000$ real Illumina reads of length $R = 101$ (SRA accession number ERX1959065) in the human genome hg38 and compared the running time of ASM-B performed with optimal search schemes obtained by our MIP for $P = K + 1$ and $P = K + 2$

Table 1. Total number of edges in the optimal search schemes found by our MIP for $K = 1, 2, 3$ and $P = K + 1$, $P = K + 2$ and $P = K + 3$ compared to full backtracking. The factor column shows the ratio of total number of edges in each scheme to that in backtracking. The optimal search schemes are listed in the Appendix in Table 3.

Distance	Search Scheme	$K = 1$		$K = 2$		$K = 3$		$K = 4$	
		Edges	Factor	Edges	Factor	Edges	Factor	Edges	Factor
Hamming	Backtracking	15,554	1.00	1,560,854	1.00	116,299,379	1.00	6,862,924,649	1.00
	Optimal ($P = K + 1$)	8,004	0.51	892,769	0.57	67,888,328	0.58	4,064,852,156	0.59
	Optimal ($P = K + 2$)	8,922	0.57	854,303	0.55	65,116,676	0.56	3,916,700,994	0.57
	Optimal ($P = K + 3$)	8,004	0.51	835,213	0.54	64,060,718	0.55	3,887,857,820	0.57
Edit	Backtracking	41,208	1.00	11,154,036	1.00	2,264,515,748	1.00	367,846,294,116	1.00
	Optimal ($P = K + 1$)	20,908	0.51	6,315,779	0.57	1,299,709,022	0.57	213,296,122,595	0.58
	Optimal ($P = K + 2$)	23,356	0.57	6,025,907	0.54	1,246,126,103	0.55	205,509,484,572	0.56
	Optimal ($P = K + 3$)	20,908	0.51	5,892,667	0.53	1,226,903,544	0.54	203,270,363,390	0.55

to that of unidirectional backtracking for Hamming and edit distance. All of our tests were conducted on Debian GNU/Linux 7.1 with Intel Xeon E5-2667V2 CPUs at fixed frequency of 3.3 GHz to prevent dynamic overclocking effects. All data was stored on tmpfs, a virtual file system in main memory to prevent loading data just on demand during the search and thus affecting the speed of the search by I/O operations. All tools were run with a single thread to make the results comparable. The results are shown in Table 2. We can see that for both Hamming and edit distance, employing our optimal search schemes, is much faster than backtracking, verifying our expectation. The respective speed-ups for $K = 1$, 2, and 3 are 3.1, 14.3, and 35.2 for Hamming distance, and 4.1, 11.1, and 21.3 for edit distance, much more significant than reduction in the total number of edges reported in Table 1.

Table 2. Running time comparison of searching all approximate matches of 100,000 Illumina reads ($R = 101$) using optimal bidirectional scheme with $P = K + 1$ and $P = K + 2$ versus backtracking for Hamming and edit distance. The factor column is the speed-up ratio versus backtracking in each category.

Dist.	Search Tool	$K = 1$		$K = 2$		$K = 3$	
		Time	Factor	Time	Factor	Time	Factor
Hamm.	Backtracking	22.80s	1.00	269.24s	1.00	2417.06s	1.00
	Optimal-scheme bidirect. ($P = K + 1$)	**7.73s**	**2.95**	**19.78s**	**13.61**	**74.62s**	**32.39**
	Optimal-scheme bidirect. ($P = K + 2$)	**7.39s**	**3.09**	**18.81s**	**14.31**	**68.69s**	**35.19**
Edit	Backtracking	43.59s	1.00	1245.70s	1.00	27889.40s	1.00
	Optimal-scheme bidirect. ($P = K + 1$)	**11.21s**	**3.89**	**120.70s**	**10.32**	**1338.61s**	**20.83**
	Optimal-scheme bidirect. ($P = K + 2$)	**10.66s**	**4.09**	**112.23s**	**11.10**	**1307.23s**	**21.33**

We also compared the optimal search schemes with Hamming distance against Bowtie1 [9] by searching for all alignments with at most K mismatches (`-v <K> -a`). It turns out that our search schemes are significantly faster (Bowtie1 takes 23, 68 and 180 seconds for 1, 2 and 3 errors).

4 Towards a full-fledged aligner

Due to the exponential complexity of ASM using FM-index in terms of K, the state-of-the-art aligners do not perform ASM completely in index but rather use a combination of search in the index and verification in text using dynamic programming (DP). Pure index-based search using standard backtracking is very slow for larger values of K. However, the drastic improvement, over standard backtracking, gained by using our optimal search schemes for search in an bidirectional index, as observed in Section 3, suggests a potential for significant improvement in the performance of read mappers that utilize index-based search. To acquire a sense of this potential, we decided to even challenge our optimal search schemes by using them in a pure index-based search and compare the results against the full-fledged state-of-the-art aligners that have the advantage of using a combination of index-based search and in-text verification using DP.

We performed our first set of comparisons with BWA [10], Yara [17], as well as an available implementation of the 01^*0-filter scheme combined with dynamic programming (named Bwolo) [19] in

the all-mapping mode. We did these comparisons for the edit distance only as all these tools work for edit distance. As before, 100,000 Illumina reads of length $R = 101$ were aligned. BWA was run with the options `-N -n` <K>, and Yara was run with the options `-e` <K> `-s` <K> `-y full -t 1`. We note that Bowtie2 [8] is not designed with all-mapping in mind (for our data set, it did not terminate in 3 hours with default configuration and `-a` option). Moreover, imposing an all-mapping with maximum K errors in Bowtie2 in a way that its results are comparable to other tools is difficult. Bowtie2 settings used in [19] do not enforce this, and nonetheless, led to a very long running time for our data set (did not terminate in 13 hours). Consequently, we did not use Bowtie2 in this study.

Remark 2. We would like to note that while BWA and Yara are standard read mapping tools for Illumina reads, Bwolo is interesting in the context of our approach. Vroland et al. [19] presented this fast method for searching in an index by partitioning the read into $K + 2$ pieces for K errors and then exploiting the fact that the $K + 2$ pieces must contain the mismatch pattern 01^*0. Their method searches for an occurrence of the 01^*0 mismatch pattern in the read and then verifies the remaining pieces of the read partially in the index (they use a unidirectional index) and partially in the text via dynamic programming. Interestingly, their 01^*0 method can be formulated as a search scheme. For example, for $K = 2$, and hence $P = 4$, their method can be expressed as the following search scheme: $\mathcal{S}_{01^*0} = \{(4321, 0000, 0122), (3214, 0000, 0122), (2134, 0000, 0022)\}$. The optimal search scheme found by our MIP for these parameters is $\mathcal{S}_{Opt} = \{(4321, 0002, 0122), (3214, 0000, 0112), (2134, 0011, 0022)\}$, which is quite similar, but has yet fewer edges. Indeed, we verified the practical superiority of $\mathcal{S}_{Opt}$ to $\mathcal{S}_{01^*0}$ by searching 100,000 real Illumina reads in the human genome with $K = 2$ Hamming and edit distance and noticed that $\mathcal{S}_{Opt}$ took less time than $\mathcal{S}_{01^*0}$ (18.8s vs. 22.0s for Hamming, and 112s vs. 160s for edit distance, respectively). □

Back to our first set of comparisons, for $K = 1$, BWA, Bwolo, and Yara, took 12.67s, 20.79s, and 11.53s, respectively. This is while, according to Table 2, the pure index-based ASM using our optimal schemes for $P = K + 1 = 2$ and $P = K + 2 = 3$, took 11.21s, and 10.66s, respectively, i.e., our optimal schemes are so effective that although the search is performed completely in index, the resulting times are better than these full-fledged state-of-the-art aligners, which use a combination of index search and in-text verification. Note that pure index-based search using standard backtracking takes 43.59s which is much worse than all aforementioned times. For $K = 2$, BWA, Bwolo, and Yara times took 143.32s, 64.09s, and 70.56s, while pure index-based ASM using our optimal schemes for $P = K + 1 = 3$ and $P = K + 2 = 4$, took 120.70s, and 112.23s, respectively, i.e., they performed better than BWA but worse than Bwolo and Yara. For $K = 3$, the benefit of using in-text verification in full-fledged aligners catches up and all of them work faster than the pure index-based search using our optimal schemes.

We find these results for our optimal search schemes very impressive. To our knowledge, this is the first time that, for $K = 1$ and $K = 2$, ASM of reads of this size ($R = 101$) performed completely in index has been reported to compete in running time with the best full-fledged aligners, which use combination of index search and in-text DP verification. This implies the power of our optimal search schemes. We note that the results are even better in strata mode. We performed a second set of comparisons with Yara, this time in strata mode. The 0-strata search means we first search the reads with 0 errors, then search all the reads with no exact match, with 1 error, and so on, until K is reached. This strategy can be generalized to s-strata, where $s \leq K$. This means that, for $b = 0$ to $K - s$, for all reads with a b-error best match, we compute all occurrences with up to $b + s$ errors. We ran Yara in 1-strata mode using `-e` <K> `-s 0 -y full -t 1` and compared the time with the pure index-based search using our optimal schemes in the same mode. For $K = 1$, 2, and 3, Yara took 3.57s, 8.00s, and 38.48s, respectively, and the pure index-based search took 4.36s, 8.57s, and 53.74s, respectively. That is for $K = 1$, 2, and 3, the pure index-based search using our optimal search scheme is about as fast as the full-fledged aligner Yara, which uses index search and in-text DP verification.

Of course, we did not expect to be able to outperform full-fledged aligners for larger values of K, because for larger K and this read length, verification in index is too costly, especially if the number of successful verifications is low (in our case almost all reads occur only once in the genome). Although Vroland et al. [19] raised the option of using a bidirectional index for verification, they only used in-text DP verification for the last pieces as they had only a unidirectional index at hand. Nevertheless, interestingly, for their data set (40bp, exactly 3 errors), pure index-based search using our optimal search schemes outperforms Bwolo by a factor of almost 1.5 (data not shown), so the read length matters. Although the optimal scheme found by our MIP is superior to the 01*0 scheme of Vroland et al. [19] for search in the index, for a larger K, one has to take into account the number of remaining verifications versus the number of edges in the trie for the remaining pieces. If that ratio is low, it does not pay off to verify in the index instead of verification in the text as our comparisons showed.

Our observations in this section about the power of our optimal search schemes suggest that a full-fledged aligner that employs an intelligent combination of search in the bidirectional FM-index using our optimal search schemes and in-text verification using DP can outperform today's best approximate read mappers. The development of such an aligner, called FAMOUS (Fast Approximate string Matching using OptimUm search Schemes) is ongoing in our group as future work.

5 Conclusions

In this paper, we contributed to the approximate string matching research as follows:

1. We proposed, for the first time, a method to solve the optimal search scheme problem for ASM-B for Hamming distance (using a MIP formulation).
2. We demonstrated that our MIP approach can solve the optimum search scheme problem to optimality in a reasonable amount of time for input parameters of considerable size, and enjoys very quick convergence to optimal or near-optimal solutions for input parameters of larger size.
3. We showed that approximate search in a bidirectional FM-index can be performed significantly faster if the optimal schemes obtained from our MIP are used. This was demonstrated based on number of edges in the search tries as well as actual running time of in-index search on real Illumina reads (up to 35 times faster than standard backtracking for 3 errors). We also showed that although our MIP solutions are for Hamming distance they perform equally well for edit distance.
4. We showcased the power of our optimal search schemes by demonstrating that for $K = 1$ and 2 errors, approximate string matching of reads of size $R = 101$ performed completely in index compete in running time with the best full-fledged aligners, which benefit from combining search in index with in-text dynamic programming verification. This suggests that a full-fledged aligner that intelligently combines search in bidirectional index using our optimal search schemes with in-text verification using DP can outperform today's best approximate aligners. The development of such an aligner, called FAMOUS is ongoing as our future work.

Moreover, our approach in this research has raised some interesting open problems:

1. Our computational experiments in Section 2.3 showed that our current MIP has two attractive properties: the early solutions it finds are optimal or near-optimal, and its optimal search scheme is insensitive to the value of R (we ask: *"is this insensitivity to R a theoretically provable fact?"*). This makes our current MIP quite powerful in practice because, even if all input parameters K, R, P, $\overline{S}$ are quite large, we can run the MIP for a short time with a much smaller R to get a solution that is most probably optimal or near-optimal for the original problem. Nevertheless, solving the MIP completely to ascertain optimality is of great interest and currently consumes considerable computational resources for large instances, especially when $\overline{S} > 5$, $K > 4$, $P > 6$, $R > 100$. We ask *"can the solution time be improved by introducing other MIP formulations, or strengthening the current formulation using strong cutting planes or further elimination of symmetric solutions?"*

2. Our current MIP is restricted to optimizing over equal-size pieces, with P and $\overline{S}$ given as part of the input. We ask *"are there (MIP) approaches to optimize over the number of pieces in the partition, and/or unequal piece lengths, and/or with no upper bound on the number of searches?"*
3. Our optimal search scheme, $\mathcal{S}_{Opt}$, for the example of Lam et al. [7] covers every possible mismatch pattern only once, ensuring that no duplicate computational effort is spent on the same mismatch pattern. We, however, believe that this is not always the case for every optimal search scheme. We ask *"how would enforcing this requirement on the solution of MIP affect its solution time and the performance of optimal search schemes it obtains?"*
4. We demonstrated that solutions found by our MIP, although for Hamming distance, perform very well for the edit distance as well. We ask *"are there (MIP) approaches to find the actual optimal search scheme for the edit distance?"*
5. We demonstrated that the verification of few occurrences with high errors in the index is worse than in-text DP verification. We ask *"what is the best point to stop verification in the index and start verifying in the text instead?."* This can be individually decided for each pattern. We are addressing this problem in the ongoing development of our upcoming full-fledged aligner FAMOUS.

Acknowledgments

We acknowledge Texas A&M University High Performance Research Computing (HPRC) for providing resources to perform parts of computational experiments. The second author also acknowledges the support of the International Max-Planck Research School for Computational Biology and Scientific Computing (IMPRS-CBSC).

References

1. Abouelhoda, M.I., Kurtz, S., Ohlebusch, E.: Replacing suffix trees with enhanced suffix arrays. Journal of Discrete Algorithms **2**(1) (2004) 53–86
2. Burrows, M., Wheeler, D.J.: A block-sorting lossless data compression algorithm. Technical Report 124, Digital SRC Research Report (1994)
3. Ferragina, P., Manzini, G.: Opportunistic data structures with applications. In: FOCS '00. (2000) 390–398
4. IBM-ILOG: Cplex 12.7.1, https://www.ibm.com/support/knowledgecenter/en/sssa5p_12.7.1/ilog.odms.studio.help/optimization_studio/topics/cos_home.html (Accessed on Nov. 2, 2017).
5. Karkkainen, J., Na, J.C.: Faster filters for approximate string matching. In: ALENEX '07. (2007) 84–90
6. Kucherov, G., Salikhov, K., Tsur, D.: Approximate string matching using a bidirectional index. Theoretical Computer Science **638** (2016) 145–158
7. Lam, T.W., Li, R., Tam, A., Wong, S., Wu, E., Yiu, S.M.: High throughput short read alignment via bi-directional bwt. In: IEEE BIBM '09. 31–36
8. Langmead, B., Salzberg, S.: Fast gapped-read alignment with Bowtie 2. Nature Methods **9**(4) (2012) 357–359
9. Langmead, B., Trapnell, C., Pop, M., Salzberg, S.L.: Ultrafast and memory-efficient alignment of short dna sequences to the human genome. Genome biology **10**(3) (2009) R25
10. Li, H., Durbin, R.: Fast and accurate short read alignment with Burrows-Wheeler transform. Bioinformatics **25**(14) (2009) 1754–1760
11. Maaß, M.G.: Linear bidirectional on-line construction of affix trees. Algorithmica **37**(1) (2003) 43–74
12. Manber, U., Myers, E.W.: Suffix arrays: a new method for on-line string searches. In: SODA '90. (1990) 319–327
13. Nemhauser, G.L., Wolsey, L.A.: Integer and combinatorial optimization. Wiley, New York (1988)
14. Pockrandt, C., Ehrhardt, M., Reinert, K.: EPR-Dictionaries: A Practical and Fast Data Structure for Constant Time Searches in Unidirectional and Bidirectional FM Indices. In: RECOMB '17. (2017) 190–206
15. Reinert, K., Dadi, T.H., Ehrhardt, M., Hauswedell, H., Mehringer, S., Rahn, R., Kim, J., Pockrandt, C., Winkler, J., Siragusa, E., Urgese, G., Weese, D.: The SeqAn C++ template library for efficient sequence analysis: A resource for programmers. Journal of Biotechnology **261** (2017) 157–168
16. Reinert, K., Langmead, B., Weese, D., Evers, D.J.: Alignment of Next-Generation Sequencing Reads. Annual review of genomics and human genetics **16** (2015) 133–151
17. Siragusa, E.: Approximate string matching for high-throughput sequencing. PhD thesis, Freie Universität Berlin (2015)
18. Strothmann, D.: The affix array data structure and its applications to rna secondary structure analysis. Theoretical Computer Science **389**(1-2) (2007) 278–294
19. Vroland, C., Salson, M., Bini, S., Touzet, H.: Approximate search of short patterns with high error rates using the 01*0 lossless seeds. Journal of Discrete Algorithms (2016) 3–16
20. Wolsey, L.A.: Integer programming. Wiley, New York (1998)

Appendix

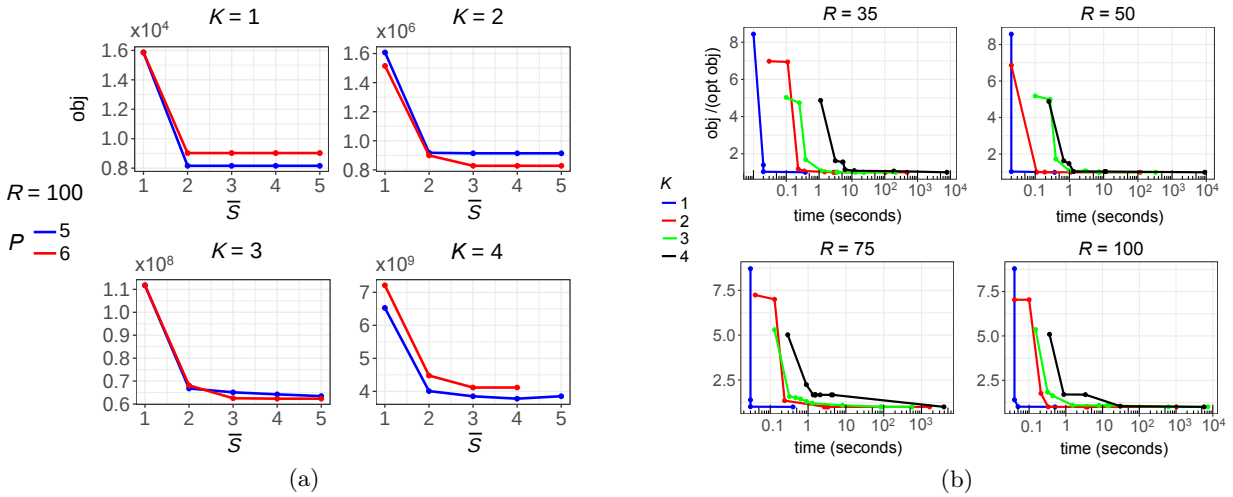


Fig. 2. (a). Sensitivity of optimal objective value to parameters R, K, $\overline{S}$, and P. For some cases due to memory overflow, there is no data point. **(b)** Rapid convergence of feasible solutions to the optimal solution.

Table 3. Search schemes found by our MIP for $K = 1, 2, 3$ and $P = K + 1$, $P = K + 2$ and $P = K + 3$ used for experiments in Tables 1 and 2. The schemes for $K = 1$ and 2 are optimal schemes with $\overline{S} = 5$. To control the running time of MIP, the schemes for $K = 3$ and 4 are best solutions found by running the MIP for 2 hours with $\overline{S} = 3$. These schemes are most probably optimal for $\overline{S} = 3$.

	$K = 1$	$K = 2$	$K = 3$	$K = 4$
Optimal ($P = K + 1$)	$(12, 00, 01)$ $(21, 01, 01)$	$(123, 002, 012)$ $(321, 000, 022)$ $(231, 011, 012)$	$(1234, 0003, 0233)$ $(2341, 0000, 1223)$ $(3421, 0022, 0033)$	$(12345, 00004, 03344)$ $(23451, 00000, 22334)$ $(54321, 00033, 00444)$
Optimal ($P = K + 2$)	$(123, 001, 001)$ $(321, 000, 011)$	$(2134, 0011, 0022)$ $(3214, 0000, 0112)$ $(4321, 0002, 0122)$	$(12345, 00022, 00333)$ $(43215, 00000, 11223)$ $(54321, 00003, 02233)$	$(123456, 000004, 033344)$ $(234561, 000000, 222334)$ $(654321, 000033, 004444)$
Optimal ($P = K + 3$)	$(1234, 0000, 0011)$ $(4321, 0001, 0011)$	$(21345, 00011, 00222)$ $(43215, 00000, 00112)$ $(54321, 00002, 01122)$	$(123456, 000003, 022233)$ $(234561, 000000, 111223)$ $(654321, 000022, 003333)$	$(1234567, 0111111, 3333334)$ $(1234567, 0000000, 0044444)$ $(7654321, 0000004, 0333344)$