Automated design of efficient search schemes for lossless approximate pattern matching

Luca Renders[0000−0002−2244−1427]1, Lore Depuydt[0000−0001−8517−0479]1, Sven Rahmann[0000−0002−8536−6065]2, and Jan Fostier[0000−0002−9994−8269]1

[1] Ghent University - imec, Technologiepark 126, 9052 Ghent, Belgium
{luca.renders,lore.depuydt,jan.fostier}@ugent.be
[2] Center for Bioinformatics Saar and Saarland University,
Saarland Informatics Campus, 66123 Saarbrücken, Germany
sven.rahmann@uni-saarland.de

Abstract. We present a novel method for the automated design of search schemes for lossless approximate pattern matching. Search schemes are combinatorial structures that define a series of searches, each of which specifies lower and upper bounds on the number of errors in each part of a partitioned search pattern, and the processing order of these parts. Collectively, these searches guarantee that all approximate occurrences of a search pattern up to a predefined number of k errors are identified. Because generating efficient search schemes is increasingly computationally expensive for larger k, search schemes have been proposed in literature for only up to $k = 4$ errors.

To design search schemes allowing more errors, we combine a greedy algorithm and a new Integer Linear Programming (ILP) formulation. Efficient, ILP-optimal search schemes for up to $k = 7$ errors are proposed and shown to outperform alternative strategies, both in theory and in practice. Additionally, we propose a technique to dynamically select an appropriate search scheme given a specific search pattern. These combined approaches result in reductions of up to 53% in runtime for higher values of k.

We introduce Hato, an open-source software tool (AGPL-3.0 license) to automatically generate search schemes. It implements the greedy algorithm and solves the ILP formulation using CPLEX. Furthermore, we present Columba 1.2, an open-source lossless read mapper (AGPL-3.0 license) implemented in C++. Columba can identify all approximate occurrences of 100 000 Illumina reads (150 bp) in the human reference genome in 24 s (maximum edit distance of 4) and in 75 s (edit distance of 6) using a single CPU core, thereby outperforming existing state-of-the-art tools for lossless approximate matching by a large margin.

Keywords: search scheme · integer linear program · approximate pattern matching · sequence alignment

1 Introduction

We consider the Approximate Pattern Matching (APM) problem: given a search pattern P (e.g., a read) and a text T (e.g., a reference (pan-)genome), find all approximate occurrences of P in T with at most k errors under the Hamming (only substitutions) or edit/Levenshtein distance (substitutions and/or indels).

We focus on *lossless* APM, where *all* such approximate occurrences are guaranteed to be identified. Many bioinformatics tools, such as BLAST [2], Bowtie [10] and BWA [11] are *lossy* in nature: they rely on fast heuristics to quickly detect some, but not necessarily all, approximate matches of P within T. Lossy algorithms are popular because they are fast. Nevertheless, lossless algorithms offer opportunities for applications in bioinformatics that benefit from a comprehensive overview of all candidate alignment positions, such as the Minimum Bait Cover Problem [1], HLA genotyping [3], or alignment to pan-genomes [4]. Because different lossless APM algorithms have identical output, the research task at hand is to improve computational performance. This work further closes the performance gap between lossy and lossless approaches.

The simplest approach to lossless APM is naive backtracking: using a full-text index, exhaustively generate, character by character, candidate occurrences of P in T, thus forming a search trie. Its runtime is governed by the size of the search space, i.e., the total number of sequences that are generated during the search process. However, naive backtracking leads to exploring an excessive number of (mostly unsuccessful) branches near the dense root of this trie. This renders backtracking computationally impractical, even for modest values of k [14].

Lam et al. [9] recognized the potential of bidirectional indexes in significantly accelerating lossless APM. This enhancement is rooted in the classical "pigeonhole principle": by dividing pattern P into $k + 1$ non-overlapping parts, with k the maximum number of allowed errors, at least one part must be error-free. By initially performing an *exact* search for each of these $k + 1$ parts and subsequently extending those matches using branching and backtracking, a significant reduction in search space size is achieved.

Kucherov et al. generalized these ideas by introducing the concept of *search schemes* [8]. Search schemes divide the pattern P into a certain number of parts and define how these parts are matched using a bidirectional full-text index, enabling the rapid elimination of unsuccessful branches in the search trie and therefore minimizing runtime. Kucherov et al. presented efficient search schemes for up to $k = 4$ errors. Kianfar et al. [7] proposed a mixed integer linear programming (MILP) formulation to generate search schemes for up to $k = 4$ errors tailored to the Hamming distance metric. The aforementioned APM based on the pigeonhole principle can also be expressed as search schemes [14], and the same is true for the work by Vroland et al. on 01*0 seeds [19]. Even though search schemes based on the pigeonhole principle and the 01*0 seeds can easily be defined for arbitrary k, they are not computationally optimal. Therefore, the design of efficient search schemes for $k > 4$ is still an open research question.

In earlier work, we introduced Columba [16, 15], a lossless read-mapper that supports arbitrary search schemes for both the edit and Hamming distance. In present work, we make following new contributions:

1. We propose a greedy algorithm to improve the computational performance of existing search schemes. Even though this algorithm cannot guarantee optimality, it is able to quickly generate lossless search schemes for up to $k = 13$ errors with good computational performance. We show that for $k = 5, 6$ and 7 errors, the search space (and thus: runtime) is reduced by respectively 55%, 63% and 71%, compared to the best previously known search schemes (i.e., either pigeonhole principle based or 01*0 seed based)
2. We present a novel ILP formulation to generate search schemes. In contrast to the greedy algorithm, these search schemes are guaranteed to be optimal under the set of used constraints. Using the ILP algorithm, we generated for the first time efficient search schemes for $k = 5, 6$ and 7 errors that demonstrate superior performance in a practical scenario. For $k = 7$ errors, the search space is further reduced with up to 16% as compared to the search scheme generated using our greedy algorithm.
3. For a given value of k, multiple, co-optimal ILP solutions can be generated. Even though co-optimal search schemes are equally fast *on average*, their individual performance may vary among different search patterns P. We propose a *dynamic selection* method where, for each pattern P, the expected best performing search scheme is chosen. We show that this heuristic can reduce the search space by up to 11% as compared to the static use of a search scheme.
4. We present Hato (Japanese for "pigeon") and Columba (Latin for "pigeon") 1.2, two open source C++11 tools[3] under AGPL-3.0 license. Hato implements the greedy algorithm and the ILP solver using CPLEX 22.1.1 [6]. Columba 1.2 is a lossless read-mapper that implements dynamic selection, on top of other improvements described in [16] and [15]. We demonstrate that Columba, using dynamic selection and the novel search schemes generated by Hato, significantly outperforms other state-of-the-art lossless alignment implementations, such as GEM [13], Yara [18], BWA aln (in lossless mode) [11] and Bwolo [19], in the task of identifying all occurrences of 150 bp Illumina reads in the human reference genome. Additionally, we demonstrate superior performance to BWA in lossy mem mode for $k \leq 3$ and a similar runtime for $k = 4$.

2 Preliminaries

We assume that up to k errors are allowed over pattern P and that P is divided into a fixed number of parts p. It is useful, but not strictly required, to choose $p \geq k+1$ to have at least one error-free part. We use square brackets for indexing

[3] available at https://github.com/biointec/hato and https://github.com/biointec/columba, respectively.

sequences, and indexing starts at 0, i.e. $s[0]$ is the first character of sequence s. Sequences of single-digit integers are written without interpunctuation, e.g. $s = (0, 0, 1)$ is written as $s = 001$.

Definition 1. *Given parameters p, k, a* ***search*** *is a triplet of sequences $\mathcal{S} = (\pi, L, U)$. The π-sequence is a permutation of $\{0, \ldots, p-1\}$ and indicates the order in which the parts of the pattern are processed. To ensure that a search can be performed with a bidirectional index, this permutation must satisfy the* ***connectivity property****: for every $i > 0$, $\pi[i]$ is either $(\min_{j<i} \pi[j]-1)$ or $(\max_{j<i} \pi[j]+1)$. The U- and L-sequences are sequences of length p over $\{0, \ldots, k\}$; they respectively provide upper and lower bounds on the* cumulative *number of errors in the parts in π-order. Because they are cumulative, both L and U must be non-decreasing, and they must satisfy $L[i] \leq U[i]$ for all i.*

Example 1. Consider the search $\mathcal{S} = (102, 001, 012)$ defined for $k = 2$ and $p = 3$. The search starts by handling part $\pi[0] = 1$. As $U[0] = 0$, an exact match of this part is searched. If such a match is found, the search continues with part $\pi[1] = 0$, where $U[1] = 1$ errors are allowed. Finally, the search continues with part $\pi[2] = 2$. Here, up to two cumulative errors are allowed ($U[2] = 2$), but at the end at least one error ($L[2] = 1$) should have been encountered.

Definition 2. *An* ***error distribution*** *e, defined for at most k errors over p parts, is an integer sequence over $\{0, \ldots, k\}$ of length p, for which $\sum_{i=0}^{p-1} e[i] \leq k$.*

Example 2. Consider error distribution 101 defined for 2 errors and 3 parts. This distribution has one error in the leftmost part and one error in the rightmost part.

Lemma 1. *There are $Q = \binom{p+k}{k}$ distinct error distributions for p parts with at most k errors [5].*

Definition 3. *A search $\mathcal{S} = (\pi, L, U)$* ***covers*** *an error distribution if and only if for all $0 \leq i < p$: $L[i] \leq \sum_{j=0}^{i} e[\pi[j]] \leq U[i]$.*

Example 3. The error distribution from example 2 is covered by the search from example 1. We have $L[0] \leq e[\pi[0]] \leq U[0]$ (spelling out $0 \leq e[1] = 0 \leq 0$); then $L[1] \leq e[\pi[0]] + e[\pi[1]] \leq U[1]$ (spelling out $0 \leq 1 \leq 1$); and finally $L[2] \leq \sum_j e[j] \leq U[2]$ (spelling out $1 \leq 1 \leq 2$).

Definition 4. *A* ***search scheme*** *$\mathbb{S}$ is a set of searches defined for fixed k and p. A search scheme is* ***valid*** *if each error distribution with p parts and at most k errors is covered by at least one search of $\mathbb{S}$.*

Example 4. The most intuitive search schemes are based on the **pigeonhole principle**. Given k, these schemes consist of $p = k + 1$ parts and $k + 1$ searches. Since there are only k errors to distribute, each possible error distribution must leave at least one of the parts error-free. Hence each search starts by matching a different part exactly. The other parts can then be processed allowing at most k errors. All searches have lower and upper bounds $L = 00 \ldots 0$ and $U = 0k \ldots k$. For $k = 2$, this results in $\mathbb{S} = \{\mathcal{S}_0, \mathcal{S}_1, \mathcal{S}_2\}$ with $\mathcal{S}_0 = (012, 000, 022)$, $\mathcal{S}_1 = (102, 000, 022)$, $\mathcal{S}_2 = (210, 000, 022)$.

Example 5. For the same $k = 2, p = 3$, Kucherov et al. [8] proposed the following scheme: $\mathcal{S}_0 = (012, 000, 022)$; $\mathcal{S}_1 = (102, 001, 012)$; $\mathcal{S}_2 = (210, 000, 012)$. While $\mathcal{S}_0$ is identical to the previous example, the lower and upper bounds are tighter for the subsequent searches, making them more efficient than the search scheme based on the pigeonhole principle. By checking all error distributions, it is shown that this search scheme is still valid.

These definitions were first formalized by Kucherov et al. [8] Search schemes are designed to systematically manage the number of allowed errors during the search process. The primary idea is to incrementally increase the number of permissible errors, a notion reflected in the U-sequence. By doing so, search schemes enable the efficient elimination of unsuccessful branches within the search trie. Additionally, the L-sequences serve a crucial role in preventing redundant coverage of error distributions. A well-constructed search scheme aims to include just one covering search for each error distribution, ensuring efficiency and effectiveness.

We aim to optimize search schemes for practical scenarios, specifically targeting repetitive and complex genomes like the human genome. In our earlier work [15, 16] we found that fast growth rates of the U-sequence lead to larger search spaces, resulting in increased computational requirements. On the other hand, fast-growing L-sequences reduce the search space. Hence, our goal is to minimize the U-sequences and maximize the L-sequences. We call the resulting schemes **minU search schemes**. In Section 3, we introduce a greedy heuristic for improving the U and L sequences, given an initial search scheme. In Section 4, we present an ILP formulation that incorporates these objectives into the objective function. Section 5 discusses the dynamic selection method, which identifies the expected best performing search scheme for a given pattern P from a set of co-optimal schemes. The practical performance of the newly designed search schemes and the dynamic selection method is evaluated and compared to the current state-of-the-art in Section 6. Finally, Section 7 provides a summary of the overall findings.

3 A Greedy Heuristic for Improving Search Schemes

We propose a greedy heuristic for designing search schemes. This algorithm takes as input any valid search scheme, whose U- and L-values are then decreased resp. increased by greedy local changes. We may start with the search scheme based on the pigeonhole principle or from the 01*0 seeds strategy [19]. The π-sequences of the searches are not modified by this heuristic.

The algorithm tracks which searches cover which error distributions. If a distribution is covered by only one search, it is labeled as *critical* for that search. First, we aim to minimize the U-sequences by iterating through positions $i = 0, \ldots, p-1$. Before each iteration, searches are sorted lexicographically by decreasing U-sequences. In case of a tie, they are sorted by increasing L-sequences. For each search $\mathcal{S}_x$ in this list, we assess the possibility of reducing

$U_x[i]$ without violating the monotonicity of the bounds or losing coverage of critical error distributions. If $U_x[i]$ is decreased, we update the list of covering searches for each error distribution previously covered by $\mathcal{S}_x$. In the event that an error distribution e is now exclusively covered by another search $\mathcal{S}_y$, it will be added to $\mathcal{S}_y$'s list of critical error distributions.

Following this, we seek to maximize the L-sequences. This process is analogous but involves iterating over decreasing positions $i = p-1, \ldots, 1, 0$ (to guarantee maintaining monotonicity of L), and we check by how much we can increase $L_x[i]$ without violating any properties.

We have generated search schemes for up to $k = 13$ using this technique. Pseudo-code for the algorithm as well as the greedily adapted search schemes are available in the appendix. The complexity of this algorithm is dominated by the number of error distributions $\binom{p+k}{k}$.

4 Integer Linear Program Formulation

Kianfar et al. [7] introduced a Mixed Integer Linear Programming (MILP) model to find optimal search schemes in a specific sense: Their model minimized the (recursively defined) expected number of enumerated substrings for a given pattern length. This expected number of explored branches is computed based on the assumption that both T and P are random sequences. However, in the context of sequence alignment, both P and T are not random; and moreover, we expect P to have an approximate occurrence in T, thus the assumption of randomness typically does not hold. As estimating the number of expected branches is computationally very expensive, Kianfar et al. only generated search schemes for up to $k = 4$ and $S = |\mathbb{S}| = 3$. For $k = 3$ and $k = 4$, this resulted in search schemes for which at least one of the searches did not start with an exact match. We proved in earlier work [16] that such search schemes result in non-competitive runtimes on repetitive and complex genomes such as the human genome.

Therefore, we propose a novel Integer Linear Programming (ILP) formulation to design practical and effective search schemes with a different objective. As stated earlier, our goal is to minimize the U-sequences and maximize the L-sequences. As an additional objective, we want to minimize the number of error distributions that are redundantly covered. Our Integer Linear Programming formulation (Table 1) aims to identify a search scheme that efficiently identifies all approximate matches with up to k errors, considering S searches and p parts. The remainder of this section provides a technical explanation of the ILP model. For brevity, we abbreviate the set $\{0, 1, \ldots, n-1\}$ by $[n]$.

ILP details. The total number of possible error distributions is denoted as $Q = \binom{p+k}{k}$ (Lemma 1). For $q \in [Q]$, e_q denotes the q-th error distribution (in some fixed enumeration), and $E_{q,j}$ denotes the cumulative number of errors encountered in e_q up to and including index j. These are precomputed constants.

For the *variables*, we partially adopt the modeling by Kianfar et al. for U, L and λ, where $U_{s,i}$ and $L_{s,i}$ respectively denote the upper and lower bound of

Table 1. ILP formulation to design search schemes with minimized U-sequences, maximized L-sequences and a small number of redundantly covered error distributions.

Variables:

U- and L-sequences	$U_{s,i}, L_{s,i} \in [k+1]$	$s \in [S], i \in [p]$
encoded π-sequences	$y_{s,i} \in \{0,1\}$	$s \in [S], i \in [p-1]$
covering indicators	$\lambda_{q,s} \in \{0,1\}$	$q \in [Q], s \in [S]$
cumulative number of errors in e_q after stage i of $\mathcal{S}_s$	$\epsilon_{q,s,i} \in [k+1]$	$q \in [Q], s \in [S], i \in [p]$
indicators of $L_{s,i} \leq \epsilon_{q,s,i}$ and $\epsilon_{q,s,i} \leq U_{s,i}$	$\mu^L_{q,s,i},\ \mu^U_{q,s,i} \in \{0,1\}$	$q \in [Q], s \in [S], i \in [p]$
indicator for $(\max_{i' \leq i} \pi_s[i']) = j$	$r'_{s,i,j} \in \{0,1\}$	$s \in [S], i \in [p], j \in [p]$

Objective:

$$\text{Minimize} \quad \sum_{s=0}^{S-1} \sum_{i=0}^{p-1} (k+1)^{(p-i-1)} \cdot U_{s,i} - \sum_{s=0}^{S-1} \sum_{i=0}^{p-1} (p-i) \cdot L_{s,i} + \sum_{q=0}^{Q-1} \sum_{s=0}^{S-1} \lambda_{q,s}\,,$$

subject to

$$L_{s,i} \leq U_{s,i} \qquad s \in [S], i \in [p] \qquad (1)$$

$$L_{s,i} \leq L_{s,i+1} \qquad s \in [S], i \in [p-1] \qquad (2)$$

$$U_{s,i} \leq U_{s,i+1} \qquad s \in [S], i \in [p-1] \qquad (3)$$

$$\sum_{s=0}^{S-1} \lambda_{q,s} \geq 1 \qquad q \in [Q] \qquad (4)$$

$$\lambda_{q,s} \leq \mu^L_{q,s,i} \qquad q \in [Q], s \in [S], i \in [p] \qquad (5)$$

$$\lambda_{q,s} \leq \mu^U_{q,s,i} \qquad q \in [Q], s \in [S], i \in [p] \qquad (6)$$

$$\lambda_{q,s} \geq \sum_{i=0}^{p-1} (\mu^L_{q,s,i} + \mu^U_{q,s,i}) - 2p + 1 \qquad q \in [Q], s \in [S] \qquad (7)$$

$$(k+1) \cdot \mu^L_{q,s,i} \geq 1 + \epsilon_{q,s,i} - L_{s,i} \qquad q \in [Q], s \in [S], i \in [p] \qquad (8)$$

$$k \cdot \mu^L_{q,s,i} \leq k + \epsilon_{q,s,i} - L_{s,i} \qquad q \in [Q], s \in [S], i \in [p] \qquad (9)$$

$$(k+1) \cdot \mu^U_{q,s,i} \geq 1 - \epsilon_{q,s,i} + U_{s,i} \qquad q \in [Q], s \in [S], i \in [p] \qquad (10)$$

$$k \cdot \mu^U_{q,s,i} \geq k - \epsilon_{q,s,i} + U_{s,i} \qquad q \in [Q], s \in [S], i \in [p] \qquad (11)$$

$$(p-1) - \sum_{h=0}^{p-2} y_{s,h} = \sum_{j=0}^{p-1} j \cdot r'_{s,i,j} \qquad s \in [S], i \in [p] \qquad (12)$$

$$\sum_{j=0}^{p-1} r'_{s,i,j} = 1 \qquad s \in [S], i \in [p] \qquad (13)$$

$$r'_{s,i,j} = 0 \qquad s \in [S], i \in [p], j \in [i] \qquad (14)$$

$$\epsilon_{q,s,i} = \sum_{j=i}^{p-1} r'_{s,i,j} \cdot E_{q,j} - \sum_{j=0}^{p-i-2} r'_{s,i,j+i+1} \cdot E_{q,j} \qquad q \in [Q], s \in [S], i \in [p] \qquad (15)$$

$$\sum_{i=0}^{p-2} y_{s+1,i} \cdot 2^i \geq 1 + \sum_{i=0}^{p-2} y_{s,i} \cdot 2^i \qquad s \in [S-1] \qquad (16)$$

$\mathcal{S}_s$ for the i-th part (in π_s-order), and $\lambda_{q,s}$ is a boolean variable that indicates whether e_q is covered by $\mathcal{S}_s$.

To model the permutations π_s with the additional connectivity restriction, we use a novel direct encoding and introduce $p-1$ binary variables $y_{s,i}$ per search $\mathcal{S}_s$. The total number of zeros in the sequence y_s corresponds to $\pi_s[0]$.

A 1-bit in the sequence corresponds to an extension to the right, whereas a 0-bit in the sequence is an extension to the left. For example, for $p = 5$, the sequence $y = 0110$ corresponds to $\pi = 21340$: We have $\pi[0] = 2$, as there are two zeros in y; then the interval $[2, 2]$ is extended to the left, right, right, and left, which gives $1, 3, 4, 0$. In this way, connected permutations on $[p]$ are in a 1-to-1 correspondence with bit-vectors of length $p - 1$.

Our new *objective function* consists of three components. The first component aims to minimize the upper bounds across all searches, with higher importance given to earlier upper bounds within a search. The weighting factor $(k+1)^{(p-i-1)}$ ensures that lexicographically smaller U-sequences are preferred over lexicographically larger ones, and give a high overall weight to the U-sequences. The second component maximizes the lower bounds. Here, a weighting factor of $(p-i)$ is used to give (slightly) higher importance to bounds earlier in the search. The third component aims to minimize the coverage of error distributions by multiple searches. By minimizing the coverage, the objective function encourages the utilization of fewer searches to handle each error distribution, thereby enhancing efficiency and reducing redundancy.

The *constraints* (and auxiliary variables to express the constraints) together ensure that the search scheme is valid. Constraints (1), (2) and (3) state the obvious properties of the L- and U-sequences.

The next constraints (4)–(7) ensure that the search scheme is valid, i.e., that each error distribution is covered by at least one search. Here, (4) directly models this coverage constraint using the λ-variables, and the other constraints express that $\lambda_{q,s} = 1$ if and only if $\mu^L_{q,s,i} = \mu^U_{q,s,i} = 1$ for all $i \in [p]$, which we then constrain to be indicators of $L_{s,i} \leq \sum_{j \leq i} e_q[\pi_s[j]]$ and $\sum_{j \leq i} e_q[\pi_s[j]] \leq U_{s,i}$, respectively.

Indeed, constraints (8)–(11) do exactly this using auxiliary variables $\epsilon_{q,s,i} := \sum_{j \leq i} e_q[\pi_s[j]]$ (which remain to be constrained to satisfy this definition). In particular, constraints (8) and (9) ensure $\mu^L_{q,s,i} = 1$ if and only if $L_{s,i} \leq \epsilon_{q,s,i}$, whereas constraints (10) and (11) ensure $\mu^U_{q,s,i} = 1$ if and only if $\epsilon_{q,s,i} \leq U_{s,i}$.

Now, for the hardest part constraints (12)–(15): As mentioned above, we need variables $\epsilon_{q,s,i} := \sum_{j \leq i} e_q[\pi_s[j]]$ that contain the cumulative number of errors of the q-th error distribution in the i-th part of search s in π_s-order, but π_s is encoded in the y_s-variables. Fortunately, we can express the right border (call it $r_{s,i}$) of the interval $\pi_s[0, \ldots, i]$ using the y-variables. As $\pi_s[0]$ corresponds to the number of zeros in y_s, and each 1-bit extends the interval to the right, we have $r_{s,0} = (p-1) - \sum_{h=0}^{p-2} y_{s,h}$ and $r_{s,i+1} = r_{s,i} + y_{s,i}$ for $i \in [p-1]$. Combining these two gives $r_{s,i} = (p-1) - \sum_{h=i}^{p-2} y_{s,h}$ for $i \in [p]$ (with the empty sum taking a value of 0 as usual). For the left interval borders we have $\ell_{s,i} = r_{s,i} - i$. We can now write $\epsilon_{q,s,i}$ as the difference between the number of errors encountered up until the right border minus the number of errors encountered before the left border: $\epsilon_{q,s,i} = E_{q,r_{s,i}} - E_{q,(\ell_{s,i}-1)}$, using the precomputed constants $E_{q,j}$. To select the correct $E_{q,j}$ values to subtract from each other using linear relations, we convert the interval endpoint $r_{s,i}$ into a sequence of binary indicator variables $r'_{s,i,j}$ with $r'_{s,i,j} = 1$ if and only if $r_{s,i} = j$. This relation is modeled by (12) and (13). As we

must have $r_{s,i} \geq i$, we have $r'_{s,i,j} = 0$ for $j < i$, which is (14). Similarly, the left boundary $\ell_{s,i}$ can be converted into a binary sequence $\ell'_{s,i,j}$. However, it is not necessary to do this explicitly because $\ell_{s,i} = r_{s,i} - i$ and hence $\ell'_{s,i,j} = r'_{s,i,j+i}$ for $j \in [p-i]$. Together, we obtain constraint (15): $\epsilon_{q,s,i} = E_{q,r_{s,i}} - E_{q,l_{s,i}-1} = \sum_{j=0}^{p-1} (r'_{s,i,j} - \ell'_{s,i-1,j})\, E_{q,j} = \sum_{j=i}^{p-1} r'_{s,i,j}\, E_{q,j} - \sum_{j=0}^{p-i-2} r'_{s,i,j+i+1}\, E_{q,j}$.

Finally, constraint (16) is a symmetry-breaking constraint enforcing a monotone order of the permutations π_s across the S searches by enforcing that the y_s-sequences (interpreted as binary numbers) are strictly increasing. (Recall that a search scheme is a set of searches in which the order of searches is unimportant; so we prescribe a canonical ordering to avoid symmetric solutions.)

Limiting the scope of the ILP-model. The ILP in Table 1 is general and works in principle for any parameter combination of k, p and S. In practice, we are mostly interested in the special case with $p = k + 1$ parts (one guaranteed error-free part) and $S = p$ searches, each of which starts with a distinct error-free part. Under these assumptions, we can introduce extra constraints that reduce the time needed to solve the ILP to optimality. We remove the symmetry-breaking constraints (16) and add the following explicit ones:

$$U_{s,0} = 0, \quad s \in [S]\,, \tag{17}$$

$$U_{s,k} = k, \quad s \in [S]\,, \tag{18}$$

$$(p-1) - \sum_{i=0}^{p-2} y_{s,i} = s, \quad s \in [S]\,, \tag{19}$$

$$y_{0,i} = 1, \quad i \in [p-1]\,, \tag{20}$$

$$y_{S-1,i} = 0, \quad i \in [p-1]\,. \tag{21}$$

Constraint (17) ensures that each search starts with an exact match. Constraint (18) ensures that each search allows up to k errors in the last part; this is necessary for $S = p = k+1$ to cover all error distributions with k 1s and a single 0. Constraint (19) enforces that each search starts with a different part by setting $\pi_s[0] = s$. Constraints (20) and (21) then respectively model the explicit π-sequence of the first search (all extensions to the right) and of the last search (all extensions to the left), as these are the only possibilities when we start on the left resp. right end.

Kucherov et al. defined the *critical sequence* of a search scheme $\mathbb{S}$ as its lexicographically maximal U-sequence. They showed that the minimal critical sequence in a valid search scheme for $p = k + 1$ is $013355\ldots kk$ for odd k and $02244\ldots kk$ for even k [8], and in an optimal search scheme, some search, called the *critical search*, has this U-sequence. Thus, we add the following constraints if k is odd, as the critical sequence starts with $01\ldots$:

$$U_{s,1} \leq 1, \qquad s \in [S]\,; \tag{22}$$

If k is even, we distinguish between the critical search s' with U-sequence $U_{s'} = 02\ldots$ and the other (non-critical) searches s, starting with $U_s = 01\ldots$. By

looping over the different non-redundant possibilities for s' (see below), we define a sequence of ILPs (one for each s') with constraints

$$U_{s,1} \leq 1, \qquad s \in [S],\, s \neq s'\,, \tag{23}$$

$$U_{s',1} = 2\,. \tag{24}$$

It is sufficient to find those schemes where the critical search $\mathcal{S}_{s'}$ starts with a part in the first half (including part $k/2$) of the pattern, as the other schemes can be constructed by mirroring the permutations, i.e., using $\bar{\pi}[i] := (p-1) - \pi[i]$, but keeping the same U- and L-sequences.

Implementation. We use the CPLEX 22.1.1 solver, implementing the ILP model in C++ with the Concert Technology API [6]. The solver uses up to 32 threads (the maximum for CPLEX) on a 64-core Intel® Xeon® E5-2698 v3 CPU, running at a base clock frequency of 2.30 GHz CPU.

To speed up solving the model to optimality, we provide a warm start. This start can be any valid search scheme. The given scheme is then first greedily improved (using only the upper bounds loop) before starting the ILP. As a first step, the ILP is solved with all lower bounds fixed to zero. Afterwards, the found search scheme is again greedily adapted. This solution with improved lower bounds is then given as initial solution (warm start) to the main ILP, which then computes the overall optimum. This last step, solving the full ILP, may be omitted for large values of k to get good (but sub-optimal) solutions fast, as solving the full ILP takes the majority of the running time. Resulting search schemes are presented in Section 6.2. For $k = 6$, the program finished within 1 hour, for $k = 7$ the optimal solution was found within 1.5 hours.

5 Dynamic Selection

In scenarios where there are multiple co-optimal search schemes that each have a notable load imbalance among their searches, dynamic selection of search schemes proves to be a valuable strategy. These co-optimal solutions are equivalent (from the ILP's point of view), but differ in which search is critical, which may lead to different performance on specific patterns and texts. This variability arises from the fact that (for practically usable search schemes) each search starts with an exact search of one of the parts (each U_s starts with 0), and the number of exact matches for each part can be very different depending on the contents of P and T, while the ILP does not consider these contents. Hence, the size of the search space for the corresponding search also fluctuates.

The crucial insight here is that when the critical search, determined by the lexicographically highest U-sequence, notably differs from the other searches, it typically dominates the overall search time, as it results in an expected broader search trie structure and consequently a larger search space (i.e. more nodes visited). By preemptively identifying the part of P with the fewest exact matches, we can select the co-optimal search scheme for which the critical search starts

with this part, to reduce the expected search space size, as only very few exact matches have to be extended by a large number of errors in the next part.

One notable instance is when k is even and $k + 1 = p = S$. In this specific scenario, co-optimal solutions arise, for which the critical search's U-string starts with 02, while all other U-strings start with 01.

Example 6. Consider a DNA read that starts with a low-complexity part (say, a repeat of `CA`) with *many* exact matches in the reference genome, but all other parts are highly specific and contain differences with respect to the reference genome, so there are *no* exact matches. Assume that we want to find all occurrences of the read within edit distance 4, using 5 parts and 5 searches, where each search starts with a different part. Consider the following two minU search schemes (co-optimal with equal ILP objective value), in which the critical search is marked with an asterisk:

	Scheme A			Scheme B		
s	π_s	L_s	U_s	π_s	L_s	U_s
0	01234	01114	01444	01234	00222	02244*
1	10234	00003	01444	12034	00000	01244
2	23410	01111	02244*	21034	01111	01244
3	32410	00000	01244	34210	00003	01444
4	43210	00222	01244	43210	01114	01444

In Scheme A, the critical search (which covers error distribution 02020) has index $s = 2$. It first searches part 2 exactly and finds zero exact matches (according to our scenario) and immediately stops. Search $s = 0$ that first searches part 0 exactly and finds many exact matches, extends all of these to part 1, allowing only 1 error, which creates much less work than extending them allowing 2 errors. In contrast, in Scheme B, the critical search has index $s = 0$. It first searches part 0 exactly and finds many exact matches, all of which have to be extended allowing up to 2 errors in part 1. With dynamic selection scheme A will be chosen. We observed search space reductions of up to 88% in practice in such scenarios using dynamic selection.

6 Experiments and Results

6.1 Dataset and Computational Environment

All benchmarks were done using a dataset of 100 000 Illumina NovaSeq 6000 reads (151 bp) randomly sampled from a larger whole genome sequencing dataset (SRR9091899). We exhaustively identified approximate read occurrences up to an edit distance of $k = 2, 3, \ldots, 7$ on both strands of the human reference genome (GRCh38) [17] where non-ACGT characters (e.g., Ns) were replaced with a randomly selected nucleotide. The chromosomes were concatenated into a single sequence. Spurious occurrences that span the borders of adjacent chromosomes can be easily filtered during post-processing.

Table 2. Average number of visited nodes (proportional to search time) for different search schemes. Search schemes above the middle line have been proposed in the literature (-: results not available), while those below the line are proposed in this work. Approximately co-optimal values are highlighted in boldface; values improved from standard minU by dynamic selection (only for even k) are highlighted in bold italics (N/A: not applicable for odd k).

Search scheme	Source	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$
Kucherov $k+1$	[8]	689	1472	4500	-	-	-
Kucherov $k+2$	[8]	858	1627	5020	-	-	-
Pigeonhole principle	[9]	847	3073	10467	31200	81908	191668
01*0	[19]	808	2183	5729	13861	31104	67347
Man_{Best}	[14]	-	-	5030	-	-	-
Kianfar	[7]	**637**	9359	36967	-	-	-
Greedy Kucherov $k+1$	here	**640**	**1287**	3643	-	-	-
Greedy Pigeonhole	here	**640**	1433	3493	6572	13592	21723
Greedy 01*0	here	711	1528	3246	6281	11426	19622
minU (ILP)	here	**638**	**1287**	**2943**	**6041**	**9978**	**16542**
minU + dynamic	here	***567***	N/A	***2859***	N/A	***9618***	N/A

Performance benchmarks were obtained using a single core of a 64-core Intel® Xeon® E5-2698 v3 CPU, operating at a base clock frequency of 2.30 GHz. Each benchmark run was repeated 20 times. We report the average wall clock time and the standard deviation.

6.2 Better Search Schemes

In our comparative analysis of search schemes we explore an array of approaches. From literature, we included Kucherov et al.'s schemes with $k+1$ and $k+2$ parts [8], the pigeonhole principle-based schemes [9], the 01*0-seeds-based schemes [19], Kianfar et al.'s scheme for $k+1$ parts, and Man_{Best}, a manual adaptation of the scheme by Kianfar et al. for $k=4$ [14]. Additionally, we include the search schemes created by our greedy algorithm, namely the greedy adaptions of the search schemes by Kucherov et al. with $k+1$ parts, the search schemes based on the pigeonhole principle ($p=k+1$) and the search schemes based on 01*0-seeds ($p=k+2$). Finally, we include the minU scheme created by the ILP (only one of the co-optimal solutions is shown). All search schemes, including the co-optimal minU schemes, are listed in the appendix in tables 4 to 11.

To assess the different search schemes, Table 2 lists, for different values of k, the average number of nodes visited in the search trie across the approximate matching procedure of all sequences and their reverse complements. This metric directly reflects the size of the search space and serves as an objective measure of efficiency, regardless of implementation quality. Smaller search spaces indicate more efficient search schemes. Each visited node represents a single character extension, which, using a bidirectional FM-index, is executed in constant time. Consequently, reducing the search space results in shorter runtimes. Note that

for even values of k, the number of nodes are averaged out over the globally co-optimal minU schemes.

Overall, the minU schemes consistently yield the smallest search space across all values of k, confirming that minimizing the U-sequence leads to better performing search schemes in practice. Notably, for $k > 2$, the simple greedy adaptations of the pigeonhole principle and the 01*0 seeds already outperform all other search schemes previously proposed in literature, demonstrating the power of this fast algorithm.

For $k = 2$, the scheme by Kianfar et al., the greedy adaptations with $k + 1$ parts, and the minU schemes are equivalent. Small performance differences that can be observed are due to minor differences (e.g. mirroring) and the specific dataset that was used. For $k = 3$ and $k = 4$, the schemes by Kucherov et al. were, prior to this work, the fastest search schemes available. Our ILP-optimal minU schemes reduce the search space by respectively 12.6% and 34.6%. For $k = 3$, the greedy adaptation of Kucherov et al.'s search scheme leads to the same search scheme as the one discovered by the ILP, and hence to the same search space size. For $k \geq 5$, no prior efficient search schemes had been proposed in the literature. As a baseline, we make use of the pigeonhole principle and the 01*0-strategy, where the 01*0-strategy can be considered the prior state-of-the-art. The greedy adaptations of the 01*0 schemes reduce the search space by 54.7%, 63.3% and 70.9% for $k = 5$, 6 and 7, respectively. The minU schemes reduce the search space by 56.4%, 67.9% and 75.4% for $k = 5$, 6 and 7, respectively.

Dynamic selection of minU schemes for even k reduces the search space even more (Table 2, last row), by an additional 11.1%, 2.9% and 3.6% for $k = 2$, 4 and 6, respectively.

In conclusion, our results demonstrate that the minU schemes, along with the greedy adaptations, outperform prior state-of-the-art approaches in terms of reducing the search space. This reduction in the number of visited nodes within the search trie translates to significantly improved runtimes.

6.3 Application to Lossless Read Mapping

We benchmark the runtime of these search schemes using our read aligner Columba 1.2. The previous version Columba 1.1 [15] implements a cache-friendly BWT representation and a dynamic partitioning strategy [16], and it relies on bit-parallel edit distance computations and in-text verification. Columba 1.2 can handle arbitrary (valid) search schemes, including the minU schemes with dynamic selection. It features a 64-bit mode to handle larger (pan-)genomes and supports outputs in SAM format [12].

We compare Columba to Bwolo [19], GEM [13], Yara [18] and BWA [11] in all-mapping mode. Note that Bwolo does not report the CIGAR string of the alignments whereas all other tools do. For the GEM aligner, not all occurrences could be reported as the tool failed when using the `all` parameter. Therefore, GEM was configured to report at most 1000 occurrences per read. For $k \geq 5$, GEM failed even with this restriction. The results for Columba 1.1 use the best

Table 3. Runtime comparison of state-of-the-art lossless alignment tools, with the exception of BWA in 'mem' mode, which is a lossy alignment algorithm. For the lossless alignment tools, the footnotes mention with which parameters the tools were ran. The Mem. column indicates the peak-RAM usage. The smaller value between brackets indicates the percentage of mapped reads with the lossless aligners.

Tool	Lang.	Mem.	$k = 2$ (90.5%)	$k = 3$ (93.1%)	$k = 4$ (94.8%)	$k = 5$ (95.9%)	$k = 6$ (96.7%)
Columba 1.2[1]	C++	22.9 GB	6.5 ± 0.1 s	12.6 ± 0.1 s	24.0 ± 0.2 s	46.4 ± 4.2 s	74.6 ± 3.7 s
Columba 1.1[2]	C++	21.5 GB	7.5 ± 1.2 s	12.9 ± 0.1 s	27.9 ± 0.2 s	72.9 ± 0.7 s	160.1 ± 6.1 s
BWA[3]	C	3.0 GB	135.9 ± 3.0 s	1473.1 ± 29.5 s	DNC (> 3h)	DNC (> 3h)	DNC (> 3h)
Bwolo	C++	6.7 GB	25.0 ± 1.0 s	63.9 ± 2.0 s	189.2 ± 4.9 s	631.2 ± 14.2 s	3336.2 ± 85.1 s
GEMv3[4]	C	13.3 GB	19.8 ± 2.2 s	37.0 ± 2.7 s	83.4 ± 5.1 s	crash	crash
Yara v0.9.11[5]	C++	4.8 GB	21.6 ± 1.6 s	84.2 ± 5.0 s	542.3 ± 22.6 s	774.8 ± 34.5 s	4666.4 ± 181.0 s
BWA mem (lossy)	C	5.1 GB	31.4 ± 0.5 s (independent of k)				

known search schemes prior to this work. These are manual adaptations based on the schemes by Kucherov et al. for $k \leq 4$ and the 01*0 seeds for $k \geq 5$.

Table 3 shows the runtimes and peak RAM-usage for different tools and different values of k. Note that Columba is currently not optimized for memory. Both versions of Columba show superior performance in terms of runtime over other lossless aligners. Columba 1.2 outperforms its predecessor for $k = 2, 4, 5$ and 6 and matches it for $k = 3$. For $k = 2$, the reduced runtime can be mainly attributed to the dynamic selection technique as the used search schemes are highly similar. For $k = 3$ the search schemes used by Columba 1.1 and 1.2 are similar, resulting in comparable runtimes. Starting from $k = 4$, we start to see the effects of the newly designed search schemes. Columba 1.2 is 14.0% faster than its predecessor for $k = 4$. For $k = 5$ and $k = 6$ the improvements are even higher, respectively 36.4% and 53.4%, as previously no optimized search schemes were available for these values.

Columba outperforms existing lossless aligners for all values of $k \geq 2$. Among the competing tools, GEM is the fastest, but it cannot handle the large number of occurrences for some reads. Bwolo and Yara require respectively 55 and 78 minutes to align reads with up to $k = 6$ errors; Columba 1.2 performs this task in only 75 seconds and is hence approximately 45× faster. BWA in all-mapping mode is outperformed by the tools specifically designed for lossless APM, and it does not finish the alignment of the 100 000 reads for $k \geq 4$ errors in 3 hours. The analysis for $k = 1$ is not included as its optimal search scheme is trivial. In earlier work, we found that Yara outperformed Columba for this case, likely due to the overhead of a bidirectional index used in Columba [15].

[1] `-e` [k] `-i 5 -ss multiple /path/to/minU/schemes` for even k
`-e` [k] `-i 5 -ss custom /path/to/minU/scheme` for odd k
[2] `-e` [k] `-i 5 -ss custom /path/to/kuch_k+1_adapted/` for $k \leq 4$
`-e` k `-i 5 -ss custom /path/to/01star0/` for $k \geq 5$
[3] `aln -N -n` [k] `-i 0 -l 150 -k` [k]
[4] `-t 1 -e` [k] `-s` [k] `--alignment-model edit --mapping-mode complete -M 1000`
[5] `-e` [k] `-s` [k] `-y full -t 1`

BWA-mem (which is lossy) does not require specifying a maximum error threshold k and typically reports only a single candidate alignment position per read. The runtime reported for BWA-mem includes the time it takes to read the index structure from disk, which is not the case for Columba 1.2. BWA-mem can handle paired-end reads, a feature not yet present in Columba 1.2. Columba 1.1 outperforms BWA-mem in terms of speed for $k = 2$ and $k = 3$. Moreover, When k is set to 4, Columba 1.2 is over 20% faster than BWA-mem, while the runtime for Columba 1.1 is similar to that of BWA-mem. Even for higher values of k, the runtime of Columba 1.2 is less than three times the runtime of BWA mem. Strikingly, BWA-mem, as a lossy tool, outputs only 100 086 alignment positions, whereas Columba 1.2 outputs respectively 6 to 60 times as many alignment positions, depending on the specific choice of k.

7 Conclusion

We have introduced novel methods for designing efficient search schemes for lossless approximate pattern matching, allowing a larger number of errors than previously possible, up to $k = 13$, filling a critical gap in this field. We presented two novel approaches: a greedy algorithm and an ILP formulation, implemented in Hato, a versatile open source tool capable of crafting search schemes using a combination of these approaches.

Furthermore, we introduced the concept of dynamic selection, tailoring the choice of search scheme to the characteristics of each input read. The reduction in search space achieved through our newly designed search schemes and dynamic selection collectively amounts to 69% for $k = 6$ errors. These ideas are implemented in Columba 1.2, which significantly outperforms existing lossless read aligners.

Our results suggest a narrowing performance gap between lossless and lossy alignment tools. The inclusion of a strata-based search strategy, where k is dynamically selected per read, could result in a read alignment tool with stronger guarantees for optimality. Given the performance advances in lossless approximate pattern matching, and given the fact that most Illumina reads can be aligned with few errors, next-generation lossless mappers need not be slower than state-of-the-art lossy mappers such as Bowtie or BWA.

Acknowledgments

Luca Renders and Lore Depuydth are funded by the Research Foundation – Flanders (FWO), through a PhD Fellowship SB (1SE7822N) and a PhD Fellowship FR (1117322N), respectively.

Appendix

Algorithm 1 Algorithm to greedily adapt a search scheme.

```
Input: A valid search scheme 𝕊 for p parts and upto k errors. This scheme is updated during the procedure.
Input: The adapted search scheme.
Generate all error distributions E.
for each error distribution e in E do
    Create an empty list e.covering
for each search S_x in 𝕊 do
    Create empty lists x.covering and x.critical
for each error distribution e in E do
    for each search S_x in 𝕊 do
        if S_x covers e then
            add x to e.covering.
            add e to x.covering.
    if |e.covering| = 1 then
        add e to e[0].critical.
for i ← 0 to p − 1 do
    Sort the searches in 𝕊 on decreasing U-string and increasing L-string.
    for each search S_x in 𝕊 do
        while true do
            Decrease the value at U_x[i] by 1.
            if U_x[i] < 0 or invalid bounds or not (all e in x.critical are covered by S_x) then
                Increase U_x[i] by 1
                Break the while loop
        for all error distributions e in x.covering do
            if S_x does not cover e then
                Remove x from e.covering.
                Remove e from x.covering
                if |e.covering| = 1 then
                    add e to e[0].critical.
for i ← p − 1 to 0 do
    Sort the searches in 𝕊 on decreasing U-string and increasing L-string.
    for each search S_x in 𝕊 do
        while true do
            Increase the value at L_x[i] by 1.
            if L_x[i] > k or invalid bounds or not (all e in x.critical are covered by S_x) then
                Decrease L_x[i] by 1
                Break the while loop
        for all error distributions e in x.covering do
            if S_x does not cover e then
                Remove x from e.covering.
                Remove e from x.covering
                if |e.covering| = 1 then
                    add e to e[0].critical.
```

Table 4. The search schemes by Kucherov et al. [8]

	$k = 2$	$k = 3$	$k = 4$
$p = k + 1$	(012, 000, 022) (210, 000, 012) (102, 001, 012)	(0123, 0000, 0133) (1023, 0011, 0133) (2310, 0000, 0133) (3210, 0011, 0133)	(01234, 00000, 02244) (43210, 00000, 01344) (10234, 00133, 01334) (01234, 00133, 01334) (32410, 00011, 01244) (21034, 00013, 01244) (10234, 00124, 01244) (01234, 00034, 00444)
$p = k + 2$	(0123, 0000, 0112) (3210, 0000, 0122) (1230, 0001, 0012) (0123, 0002, 0022)	(01234, 00000, 01233) (12340, 00000, 01223) (23410, 00001, 01133) (34210, 00012, 00333)	(012345, 000000, 012344) (123450, 000000, 012344) (543210, 000001, 012244) (345210, 000012, 011344) (234510, 000023, 011244) (453210, 000133, 003344) (012345, 000333, 003344) (012345, 000044, 002444) (231045, 000124, 002244) (453210, 000044, 001444)

Table 5. The used search schemes by Kianfar et al. [7] and the manual adaptation ("Man_{Best}") of the search scheme for $k = 4$ [14].

$k = 2$	$k = 3$	$k = 4$	Man_{Best} $k = 4$
(012, 002, 012) (210, 000, 022) (120, 011, 012)	(0123, 0003, 0233) (1230, 0000, 1233) (2310, 0022, 0033)	(01234, 00004, 03344) (12340, 00000, 22334) (43210, 00033, 00444)	(012345, 000004, 033344) (123450, 000000, 022334) (213450, 011111, 022334) (321450, 012222, 012334) (543210, 000033, 004444)

Table 6. The search schemes based on the pigeonhole principle and the greedy adaptions of these schemes found by using Hato for up to $k = 7$. The left column is the original scheme and the right column is the greedily adpated search scheme.

	Original	Greedily adapted
$k = 2$	(012, 000, 022) (120, 000, 022) (210, 000, 022)	(012, 012, 022) (102, 000, 012) (210, 001, 012)
$k = 3$	(0123, 0000, 0333) (1023, 0000, 0333) (2310, 0000, 0333) (3210, 0000, 0333)	(0123, 0000, 0133) (1230, 0023, 0223) (2310, 0001, 0133) (3210, 0112, 0133)
$k = 4$	(01234, 00000, 04444) (12340, 00000, 04444) (23410, 00000, 04444) (34210, 00000, 04444) (43210, 00000, 04444)	(01234, 01234, 02344) (10234, 00123, 01444) (21034, 00000, 01244) (32104, 00011, 01334) (43210, 00002, 01344)
$k = 5$	(012345, 000000, 055555) (234510, 000000, 055555) (123450, 000000, 055555) (453210, 000000, 055555) (345210, 000000, 055555) (543210, 000000, 055555)	(012345, 000000, 012555) (123450, 000034, 014445) (234510, 000001, 013355) (345210, 002345, 022555) (453210, 000112, 013555) (543210, 011223, 013555)
$k = 6$	(0123456, 0000000, 0666666) (1234560, 0000000, 0666666) (2345610, 0000000, 0666666) (3456210, 0000000, 0666666) (4563210, 0000000, 0666666) (5643210, 0000000, 0666666) (6543210, 0000000, 0666666)	(0123456, 0123456, 0235666) (1023456, 0012345, 0145666) (2103456, 0000012, 0126666) (3210456, 0001123, 0133666) (4321056, 0000234, 0134466) (5432106, 0000000, 0125556) (6543210, 0000001, 0125666)
$k = 7$	(01234567, 00000000, 07777777) (12345670, 00000000, 07777777) (23456710, 00000000, 07777777) (34567210, 00000000, 07777777) (45673210, 00000000, 07777777) (56743210, 00000000, 07777777) (67543210, 00000000, 07777777) (76543210, 00000000, 07777777)	(01234567, 00000000, 01237777) (12345670, 00000012, 01266667) (23456710, 00000001, 01255577) (34567210, 00003456, 01444777) (45673210, 00000123, 01337777) (56743210, 00234567, 02257777) (67543210, 00011234, 01357777) (76543210, 01122345, 01357777)

Table 7. The search schemes based on 01*0 seeds [19, 14] and the greedy adaptions of these schemes found by using Hato for up to $k = 7$. The left column is the original scheme and the right column is the greedily adapted search scheme.

	Original	Greedily adapted
$k = 2$	(0123, 0000, 0122) (1230, 0000, 0122) (2310, 0000, 0022)	(0123, 0002, 0122) (1230, 0011, 0112) (2310, 0000, 0022)
$k = 3$	(01234, 00000, 01333) (12340, 00000, 01333) (23410, 00000, 01333) (34210, 00000, 00333)	(01234, 00003, 01233) (12340, 00022, 01223) (23410, 00111, 01133) (34210, 00000, 00333)
$k = 4$	(012345, 000000, 014444) (123450, 000000, 014444) (234510, 000000, 014444) (345210, 000000, 014444) (453210, 000000, 004444)	(012345, 000004, 012444) (123450, 000033, 012334) (234510, 000222, 012244) (345210, 001111, 011444) (453210, 000000, 003444)
$k = 5$	(0123456, 0000000, 0155555) (1234560, 0000000, 0155555) (2345610, 0000000, 0155555) (3456210, 0000000, 0155555) (4563210, 0000000, 0155555) (5643210, 0000000, 0055555)	(0123456, 0000004, 0123555) (1234560, 0000045, 0124445) (2345610, 0000333, 0123355) (3456210, 0002222, 0122555) (4563210, 0011111, 0115555) (5643210, 0000000, 0035555)
$k = 6$	(01234567, 00000000, 01666666) (12345670, 00000000, 01666666) (23456710, 00000000, 01666666) (34567210, 00000000, 01666666) (45673210, 00000000, 01666666) (56743210, 00000000, 01666666) (67543210, 00000000, 00666666)	(01234567, 00000005, 01236666) (12345670, 00000044, 01235556) (23456710, 00000456, 01244466) (34567210, 00003333, 01233666) (45673210, 00022222, 01226666) (56743210, 00111111, 01156666) (67543210, 00000000, 00356666)
$k = 7$	(012345678, 000000000, 017777777) (123456780, 000000000, 017777777) (234567810, 000000000, 017777777) (345678210, 000000000, 017777777) (456783210, 000000000, 017777777) (567843210, 000000000, 017777777) (678543210, 000000000, 017777777) (786543210, 000000000, 007777777)	(012345678, 000000004, 012347777) (123456780, 000000056, 012366667) (234567810, 000000445, 012355577) (345678210, 000004567, 012444777) (456783210, 000033333, 012337777) (567843210, 000222222, 012277777) (678543210, 001111111, 011577777) (786543210, 000000000, 003577777)

Table 8. The greedy adaptations of the search schemes by Kucherov et al. with $p = k + 1$.

$k = 2$	$k = 3$	$k = 4$
(012, 012, 022) (102, 001, 012) (210, 000, 012)	(0123, 0000, 0133) (1023, 0111, 0133) (2310, 0002, 0133) (3210, 0113, 0133)	(01234, 00002, 02244) (01234, 00334, 01334) (01234, 00344, 00444) (10234, 01334, 01334) (10234, 01224, 01244) (21034, 00113, 01244) (32410, 00111, 01244) (43210, 00000, 01344)

Table 9. The minU schemes for $k + 1$ parts found by Hato. For $k = 4$ and $k = 6$, two co-optimal variants are given. For all k, symmetric variants can be creating by reverse-mapping the π-strings.

$k = 2$	(012, 011, 022) (102, 000, 012) (210, 002, 012)	
$k = 3$	(0123, 0000, 0133) (1023, 0111, 0133) (2310, 0002, 0133) (3210, 0113, 0133)	
$k = 4$	(01234, 00222, 02244) (12034, 00000, 01244) (21034, 01111, 01244) (34210, 00003, 01444) (43210, 01114, 01444)	(01234, 01114, 01444) (10234, 00003, 01444) (23410, 01111, 02244) (32410, 00000, 01244) (43210, 00222, 01244)
$k = 5$	(012345, 000222, 013555) (102345, 011333, 013555) (231045, 000000, 013355) (321045, 011111, 013355) (453210, 000004, 013555) (543210, 011115, 013555)	
$k = 6$	(0123456, 0022226, 0226666) (1203456, 0111115, 0126666) (2103456, 0000004, 0126666) (3456210, 0000000, 0133666) (4356210, 0111111, 0133666) (5643210, 0002222, 0133666) (6543210, 0113333, 0133666)	(0123456, 0111115, 0126666) (1023456, 0000004, 0126666) (2103456, 0022226, 0226666) (3456210, 0002222, 0133666) (4356210, 0113333, 0133666) (5643210, 0000000, 0133666) (6543210, 0111111, 0133666)
$k = 7$	(01234567, 00000000, 01337777) (10234567, 01111111, 01337777) (23104567, 00022222, 01337777) (32104567, 01133333, 01337777) (45673210, 00000004, 01337777) (54673210, 01111115, 01337777) (67543210, 00022226, 01337777) (76543210, 01133337, 01337777)	

Table 10. The greedy adaptations of the search schemes based on the pigeonhole principle, for $8 \leq k \leq 9$, found by using Hato.

$k = 8$	$k = 9$
(012345678, 012345678, 023578888) (102345678, 001234567, 014578888) (210345678, 000001234, 012678888) (321045678, 000112345, 013378888) (432105678, 000023456, 013448888) (543210678, 000000012, 012555888) (654321078, 000000123, 012566688) (765432108, 000000000, 012377778) (876543210, 000000001, 012388888)	(0123456789, 0000000000, 0123499999) (1234567890, 0000000012, 0123888889) (2345678910, 0000000001, 0123777799) (3456789210, 0000001234, 0126666999) (4567893210, 0000000123, 0125559999) (5678943210, 0000345678, 0144499999) (6789543210, 0000012345, 0133799999) (7896543210, 0023456789, 0225799999) (8976543210, 0001123456, 0135799999) (9876543210, 0112234567, 0135799999)

Table 11. The greedy adaptations of the search schemes based on the pigeonhole principle, for $10 \leq k \leq 13$, found by using Hato.

	Greedily adapted
$k = 10$	(0 1 2 3 4 5 6 7 8 9 10 , 0 1 2 3 4 5 6 7 8 9 10 , 0 2 3 5 7 9 10 10 10 10 10) (1 0 2 3 4 5 6 7 8 9 10 , 0 0 1 2 3 4 5 6 7 8 9 , 0 1 4 5 7 9 10 10 10 10 10) (2 1 0 3 4 5 6 7 8 9 10 , 0 0 0 0 0 1 2 3 4 5 6 , 0 1 2 6 7 9 10 10 10 10 10) (3 2 1 0 4 5 6 7 8 9 10 , 0 0 0 1 1 2 3 4 5 6 7 , 0 1 3 3 7 9 10 10 10 10 10) (4 3 2 1 0 5 6 7 8 9 10 , 0 0 0 0 2 3 4 5 6 7 8 , 0 1 3 4 4 9 10 10 10 10 10) (5 4 3 2 1 0 6 7 8 9 10 , 0 0 0 0 0 0 0 1 2 3 4 , 0 1 2 5 5 5 10 10 10 10 10) (6 5 4 3 2 1 0 7 8 9 10 , 0 0 0 0 0 0 1 2 3 4 5 , 0 1 2 5 6 6 6 10 10 10 10) (7 6 5 4 3 2 1 0 8 9 10 , 0 0 0 0 0 0 0 0 0 1 2 , 0 1 2 3 7 7 7 7 10 10 10) (8 7 6 5 4 3 2 1 0 9 10 , 0 0 0 0 0 0 0 0 1 2 3 , 0 1 2 3 8 8 8 8 8 10 10) (9 8 7 6 5 4 3 2 1 0 10 , 0 0 0 0 0 0 0 0 0 0 0 , 0 1 2 3 4 9 9 9 9 9 10) (10 9 8 7 6 5 4 3 2 1 0 , 0 0 0 0 0 0 0 0 0 0 1 , 0 1 2 3 4 10 10 10 10 10 10)
$k = 11$	(0 1 2 3 4 5 6 7 8 9 10 11 , 0 0 0 0 0 0 0 0 0 0 0 0 , 0 0 , 0 1 2 3 4 5 11 11 11 11 11 11) (1 2 3 4 5 6 7 8 9 10 11 0 , 0 0 0 0 0 0 0 0 0 0 0 1 , 2 , 0 1 2 3 4 10 10 10 10 10 10 11) (2 3 4 5 6 7 8 9 10 11 1 0 , 0 0 0 0 0 0 0 0 0 0 0 0 , 1 , 0 1 2 3 4 9 9 9 9 9 11 11) (3 4 5 6 7 8 9 10 11 2 1 0 , 0 0 0 0 0 0 0 0 0 1 2 , 3 , 4 , 0 1 2 3 8 8 8 8 8 11 11 11) (4 5 6 7 8 9 10 11 3 2 1 0 , 0 0 0 0 0 0 0 0 0 0 1 , 2 , 3 , 0 1 2 3 7 7 7 7 11 11 11 11) (5 6 7 8 9 10 11 4 3 2 1 0 , 0 0 0 0 0 0 0 1 2 3 4 , 5 , 6 , 0 1 2 6 6 6 6 11 11 11 11 11) (6 7 8 9 10 11 5 4 3 2 1 0 , 0 0 0 0 0 0 0 0 1 2 3 , 4 , 5 , 0 1 2 5 5 5 11 11 11 11 11 11) (7 8 9 10 11 6 5 4 3 2 1 0 , 0 0 0 0 3 4 5 6 7 8 , 9 10 , 0 1 4 4 4 9 11 11 11 11 11 11) (8 9 10 11 7 6 5 4 3 2 1 0 , 0 0 0 0 0 1 2 3 4 5 , 6 , 7 , 0 1 3 3 7 9 11 11 11 11 11 11) (9 10 11 8 7 6 5 4 3 2 1 0 , 0 0 2 3 4 5 6 7 8 9 10 11 , 0 2 2 5 7 9 11 11 11 11 11 11) (10 11 9 8 7 6 5 4 3 2 1 0 , 0 0 0 1 1 2 3 4 5 6 7 8 , 0 1 3 5 7 9 11 11 11 11 11 11) (11 10 9 8 7 6 5 4 3 2 1 0 , 0 1 1 2 2 3 4 5 6 7 8 9 , 0 1 3 5 7 9 11 11 11 11 11 11)
$k = 12$	(0 1 2 3 4 5 6 7 8 9 10 11 12 , 0 1 2 3 4 5 6 7 8 9 10 11 12 , 0 2 3 5 7 9 11 12 12 12 12 12 12) (1 0 2 3 4 5 6 7 8 9 10 11 12 , 0 0 1 2 3 4 5 6 7 8 9 10 11 , 0 1 4 5 7 9 11 12 12 12 12 12 12) (2 1 0 3 4 5 6 7 8 9 10 11 12 , 0 0 0 0 0 1 2 3 4 5 6 7 8 , 0 1 2 6 7 9 11 12 12 12 12 12 12) (3 2 1 0 4 5 6 7 8 9 10 11 12 , 0 0 0 1 1 2 3 4 5 6 7 8 9 , 0 1 3 3 7 9 11 12 12 12 12 12 12) (4 3 2 1 0 5 6 7 8 9 10 11 12 , 0 0 0 0 2 3 4 5 6 7 8 9 10 , 0 1 3 4 4 9 11 12 12 12 12 12 12) (5 4 3 2 1 0 6 7 8 9 10 11 12 , 0 0 0 0 0 0 1 2 3 4 5 6 7 , 0 1 2 5 5 5 11 12 12 12 12 12 12) (6 5 4 3 2 1 0 7 8 9 10 11 12 , 0 0 0 0 0 0 0 1 2 3 4 5 6 , 0 1 2 5 6 6 6 12 12 12 12 12 12) (7 6 5 4 3 2 1 0 8 9 10 11 12 , 0 0 0 0 0 0 0 0 1 2 3 4 , 0 1 2 3 7 7 7 12 12 12 12 12 12) (8 7 6 5 4 3 2 1 0 9 10 11 12 , 0 0 0 0 0 0 0 0 0 1 2 3 4 5 , 0 1 2 3 8 8 8 8 12 12 12 12 12) (9 8 7 6 5 4 3 2 1 0 10 11 12 , 0 0 0 0 0 0 0 0 0 0 0 1 2 3 4 , 0 1 2 3 4 9 9 9 9 9 12 12 12) (10 9 8 7 6 5 4 3 2 1 0 11 12 , 0 0 0 0 0 0 0 0 0 0 0 0 1 2 3 , 0 1 2 3 4 10 10 10 10 10 10 12 12) (11 10 9 8 7 6 5 4 3 2 1 0 12 , 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 , 0 1 2 3 4 5 11 11 11 11 11 11 12) (12 11 10 9 8 7 6 5 4 3 2 1 0 , 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 , 0 1 2 3 4 5 12 12 12 12 12 12 12)
$k = 13$	(0 1 2 3 4 5 6 7 8 9 10 11 12 13 , 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 , 0 0 0 0 , 0 1 2 3 4 5 6 13 13 13 13 13 13 13) (1 2 3 4 5 6 7 8 9 10 11 12 13 0 , 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 , 0 0 1 2 , 0 1 2 3 4 5 12 12 12 12 12 12 12 13) (2 3 4 5 6 7 8 9 10 11 12 13 1 0 , 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 , 0 0 0 1 , 0 1 2 3 4 5 11 11 11 11 11 11 13 13) (3 4 5 6 7 8 9 10 11 12 13 2 1 0 , 0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 , 3 4 , 0 1 2 3 4 10 10 10 10 10 10 13 13 13) (4 5 6 7 8 9 10 11 12 13 3 2 1 0 , 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 , 2 3 , 0 1 2 3 4 9 9 9 9 9 13 13 13 13) (5 6 7 8 9 10 11 12 13 4 3 2 1 0 , 0 0 0 0 0 0 0 0 0 0 0 1 2 , 3 4 5 , 6 , 0 1 2 3 8 8 8 8 8 13 13 13 13 13) (6 7 8 9 10 11 12 13 5 4 3 2 1 0 , 0 0 0 0 0 0 0 0 0 0 1 , 2 3 4 , 5 , 0 1 2 3 7 7 7 7 13 13 13 13 13 13) (7 8 9 10 11 12 13 6 5 4 3 2 1 0 , 0 0 0 0 0 0 0 0 1 2 3 4 , 5 6 7 8 , 0 1 2 6 6 6 6 13 13 13 13 13 13 13) (8 9 10 11 12 13 7 6 5 4 3 2 1 0 , 0 0 0 0 0 0 0 0 1 2 3 , 4 5 6 7 , 0 1 2 5 5 5 11 13 13 13 13 13 13 13) (9 10 11 12 13 8 7 6 5 4 3 2 1 0 , 0 0 0 0 3 4 5 6 7 8 , 9 10 11 12 , 0 1 4 4 4 9 11 13 13 13 13 13 13 13) (10 11 12 13 9 8 7 6 5 4 3 2 1 0 , 0 0 0 0 0 1 2 3 4 5 , 6 7 8 9 , 0 1 3 3 7 9 11 13 13 13 13 13 13 13) (11 12 13 10 9 8 7 6 5 4 3 2 1 0 , 0 0 2 3 4 5 6 7 8 9 10 11 12 13 , 0 2 2 5 7 9 11 13 13 13 13 13 13 13) (12 13 11 10 9 8 7 6 5 4 3 2 1 0 , 0 0 0 1 1 2 3 4 5 6 7 8 9 10 , 0 1 3 5 7 9 11 13 13 13 13 13 13 13) (13 12 11 10 9 8 7 6 5 4 3 2 1 0 , 0 1 1 2 2 3 4 5 6 7 8 9 10 11 , 0 1 3 5 7 9 11 13 13 13 13 13 13 13)

References

1. Alanko, J.N., Slizovskiy, I.B., Lokshtanov, D., Gagie, T., Noyes, N.R., Boucher, C.: Syotti: scalable bait design for DNA enrichment. Bioinformatics **38**(Supplement_1), i177–i184 (06 2022). https://doi.org/10.1093/bioinformatics/btac226, https://doi.org/10.1093/bioinformatics/btac226
2. Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J.: Basic local alignment search tool. Journal of Molecular Biology **215**(3), 403–10 (1990)
3. Claeys, A., Merseburger, P., Staut, J., Marchal, K., den Eynden, J.V.: Benchmark of tools for in silico prediction of mhc class i and class ii genotypes from ngs data. BMC Genomics **24**, 247 (2023). https://doi.org/10.1186/s12864-023-09351-z, https://doi.org/10.1186/s12864-023-09351-z
4. Depuydt, L., Renders, L., Abeel, T., Fostier, J.: Pan-genome de bruijn graph using the bidirectional fm-index. BMC Bioinformatics **24**(1), 400 (Oct 2023). https://doi.org/10.1186/s12859-023-05531-6, https://doi.org/10.1186/s12859-023-05531-6
5. Feller, W.: An Introduction to Probability Theory and Its Applications, Vol. 1. Wiley, New York, 3rd edn. (1968)
6. IBM-ILOG: Cplex (2022), https://www.ibm.com/docs/en/icos/22.1.1?topic=documentation-introducing-ilog-cplex-optimization-studio-2211, accessed on Jul. 2, 2023

7. Kianfar, K., Pockrandt, C., Torkamandi, B., Luo, H., Reinert, K.: FAMOUS: fast approximate string matching using optimum search schemes. CoRR (2017), http://arxiv.org/abs/1711.02035
8. Kucherov, G., Salikhov, K., Tsur, D.: Approximate string matching using a bidirectional index. In: Kulikov, A.S., Kuznetsov, S.O., Pevzner, P. (eds.) Combinatorial Pattern Matching. pp. 222–231. Springer International Publishing, Cham (2014)
9. Lam, T., Li, R., Tam, A., Wong, S., Wu, E., Yiu, S.: High throughput short read alignment via bi-directional BWT. In: IEEE International Conference on Bioinformatics and Biomedicine. pp. 31 – 36 (Dec 2009). https://doi.org/10.1109/BIBM.2009.42
10. Langmead, B.: Aligning short sequencing reads with bowtie. Current protocols in bioinformatics **32**(1), 11–7 (2010)
11. Li, H., Durbin, R.: Fast and accurate short read alignment with Burrows–Wheeler transform. Bioinformatics **25**(14), 1754–1760 (2009). https://doi.org/10.1093/bioinformatics/btp324
12. Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., Durbin, R., 1000 Genome Project Data Processing Subgroup: The sequence Alignment/Map format and SAMtools. Bioinformatics **25**(16), 2078–2079 (Jun 2009)
13. Marco-Sola, S., Sammeth, M., R, G., Ribeca, P.: The gem mapper: fast, accurate and versatile alignment by filtration. Nature Methods **9**(12), 1185–1188 (2012). https://doi.org/10.1028/nmeth.2221
14. Pockrandt, C.M.: Approximate String Matching: Improving Data Structures and Algorithms. Ph.D. thesis, Freien Universität Berlin (2019), http://dx.doi.org/10.17169/refubium-2185
15. Renders, L., Depuydt, L., Fostier, J.: Approximate pattern matching using search schemes and in-text verification. In: Rojas, I., Valenzuela, O., Rojas, F., Herrera, L.J., Ortuño, F. (eds.) Bioinformatics and Biomedical Engineering. pp. 419–435. Springer International Publishing, Cham (2022)
16. Renders, L., Marchal, K., Fostier, J.: Dynamic partitioning of search patterns for approximate pattern matching using search schemes. iScience **24**(7), 102687 (2021). https://doi.org/10.1016/j.isci.2021.102687
17. Schneider, V., Graves-Lindsay, T., Howe, K., Bouk, N., Chen, H.C., Kitts, P., Murphy, T., Pruitt, K., Thibaud-Nissen, F., Albracht, D., Fulton, R., Kremitzki, M., Magrini, V., Markovic, C., McGrath, S., Steinberg, K., Auger, K., Chow, W., Collins, J., Church, D.: Evaluation of GRCh38 and de novo haploid genome assemblies demonstrates the enduring quality of the reference assembly. Genome Research **27** (2017). https://doi.org/10.1101/gr.213611.116
18. Siragusa, E.: Approximate string matching for high-throughput sequencing. Ph.D. thesis (2015)
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 **37**, 3–16 (2016). https://doi.org/10.1016/j.jda.2016.03.002