Fast and optimal sequence-to-graph alignment guided by seeds

Pesho Ivanov, Benjamin Bichsel, and Martin Vechev

Department of Computer Science,
ETH Zurich, Switzerland
{firstname.lastname}@inf.ethz.ch

Abstract. We present a novel A* *seed heuristic* that enables fast and optimal sequence-to-graph alignment, guaranteed to minimize the edit distance of the alignment assuming non-negative edit costs.
We phrase optimal alignment as a shortest path problem and solve it by instantiating the A* algorithm with our seed heuristic. The seed heuristic first extracts non-overlapping substrings (*seeds*) from the read, finds exact seed *matches* in the reference, marks preceding reference positions by *crumbs*, and uses the crumbs to direct the A* search. The key idea is to punish paths for the absence of foreseeable seed matches. We prove admissibility of the seed heuristic, thus guaranteeing alignment optimality.

Our implementation extends the free and open source aligner and demonstrates that the seed heuristic outperforms all state-of-the-art optimal aligners including GraphAligner, Vargas, PaSGAL, and the prefix heuristic previously employed by AStarix. Specifically, we achieve a consistent speedup of >60× on both short Illumina reads and long HiFi reads (up to 25kbp), on both the *E. coli* linear reference genome (1Mbp) and the MHC variant graph (5Mbp). Our speedup is enabled by the seed heuristic consistently skipping >99.99% of the table cells that optimal aligners based on dynamic programming compute.

AStarix aligner and evaluations: https://github.com/eth-sri/astarix
Full paper: https://www.biorxiv.org/content/10.1101/2021.11.05.467453

Keywords: Genome graph · Optimal alignment · Semi-global alignment · Edit distance · Shortest path · Long reads · A* algorithm · Seed heuristic

1 Introduction

Alignment of reads to a reference genome is an essential and early step in most bioinformatics pipelines. While linear references have been used traditionally, an increasing interest is directed towards graph references capable of representing biological variation [1]. Specifically, a *sequence-to-graph* alignment is a base-to-base correspondence between a given read and a walk in the graph. As sequencing errors and biological variation result in inexact read alignments, edit distance

is the most common metric that alignment algorithms optimize in order to find the most probable read origin in the reference.

Suboptimal alignment. In the last decades, approximate and alignment-free methods satisfied the demand for faster algorithms which process huge volumes of genetic data [2]. *Seed-and-extend* is arguably the most popular paradigm in read alignment [3, 4, 5]. First, substrings (called *seeds* or *kmers*) of the read are extracted, then aligned to the reference, and finally prospective matching locations are *extended* on both sides to align the full read.

While such a heuristic may produce acceptable alignments in many cases, it fundamentally does not provide quality guarantees, resulting in suboptimal alignment accuracy. In contrast, we demonstrate in this work that seeds can benefit optimal alignment as well.

Key challenges in optimal alignment. Finding optimal alignments is desirable but expensive in the worst case, requiring $\mathcal{O}(Nm)$ time [6], for graph size N and read length m. Unfortunately, most optimal sequence-to-graph aligners rely on dynamic programming (DP) and always reach this worst-case asymptotic runtime. Such aligners include Vargas [7], PaSGAL [8], GraphAligner [9], HGA [10], and VG [1], which use bit-level optimizations and parallelization to increase their throughput.

In contrast, AStarix [11] follows the promising direction of using a heuristic to avoid worst-case runtime on realistic data. To this end, AStarix rephrases the task of alignment as a shortest-path problem in an *alignment graph* extended by a *trie index*, and solves it using the A* algorithm instantiated with a problem-specific prefix heuristic. Importantly, its choice of heuristic only affects performance, not optimality. Unlike DP-based algorithms, this prefix heuristic allows scaling sublinearly with the reference size, substantially increasing performance on large genomes. However, it can only efficiently align reads of limited length.

This work: Optimal alignment for short and long reads. In this work, we address the key challenge of scaling to long HiFi reads, while retaining the superior scaling of AStarix in the size of the reference graph. To this end, we instantiate the A* algorithm with a novel seed heuristic, which outperforms existing optimal aligners on both short and long HiFi reads. Specifically, the seed heuristic utilizes information from the whole read to narrowly direct the A* search by placing *crumbs* on graph nodes which lead up to a *seed match*, i.e., an exact match of a substring of the read.

Overall, the contributions presented in this work are:

1. A novel A* seed heuristic that exploits information from the whole read to quickly align it to a general graphs reference.
2. An optimality proof showing that the seed heuristic always finds an alignment with minimal edit distance.
3. An implementation of the seed heuristic as part of the AStarix aligner.

$$
\begin{array}{lll}
(\langle u,i\rangle,\langle v,i+1\rangle,q[i],\Delta_{\text{match}}) \in E_{\mathsf{a}}^q & \text{if } (u,v,\ell) \in E_{\mathsf{r}}, \ell = q[i] & \text{(match)} \\
(\langle u,i\rangle,\langle v,i+1\rangle,q[i],\Delta_{\text{subst}}) \in E_{\mathsf{a}}^q & \text{if } (u,v,\ell) \in E_{\mathsf{r}}, \ell \neq q[i] & \text{(substitution)} \\
(\langle u,i\rangle,\langle v,i\rangle,\varepsilon,\Delta_{\text{del}}) \in E_{\mathsf{a}}^q & \text{if } (u,v,\ell) \in E_{\mathsf{r}} & \text{(deletion)} \\
(\langle u,i\rangle,\langle u,i+1\rangle,q[i],\Delta_{\text{ins}}) \in E_{\mathsf{a}}^q & & \text{(insertion)},
\end{array}
$$

Fig. 1: Formal definition of alignment graph edges $E_{\mathsf{a}}^q \subseteq V_{\mathsf{a}}^q \times V_{\mathsf{a}}^q \times \Sigma_\varepsilon \times \mathbb{R}_{\geq 0}$. Here, $u, v \in V_{\mathsf{r}}$, $0 \leq i < |q|$, $\ell \in \Sigma$, and ε represents the empty string, indicating that letter ℓ was deleted.

4. An extensive evaluation of our approach, showing that we align both short Illumina reads and long HiFi reads to both linear and graph references $\geq 60\times$ faster than existing optimal aligners.
5. A demonstration of superior empirical runtime scaling in the reference size N: $N^{0.46}$ on Illumina reads and $N^{0.11}$ on HiFi reads.

2 Prerequisites

We define the task of alignment as a shortest path problem (§2.1) to be solved using the A* algorithm (§2.2).

2.1 Problem statement: Alignment as shortest path

In the following, we formalize the task of optimally aligning a read to a reference graph in terms of finding a shortest path in an *alignment graph*. Our discussion closely follows [11, §2] and is in line with [12].

Reference graph. A reference graph $G_{\mathsf{r}} = (V_{\mathsf{r}}, E_{\mathsf{r}})$ encodes a collection of references to be considered when aligning a read. Its directed edges $E_{\mathsf{r}} \subseteq V_{\mathsf{r}} \times V_{\mathsf{r}} \times \Sigma$ are labeled by nucleotide letters from $\Sigma = \{\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}\}$, hence any walk π_{r} in G_{r} spells a string $\sigma(\pi_{\mathsf{r}}) \in \Sigma^*$.

An alignment of a read $q \in \Sigma^*$ to a reference graph G_{r} consists of (i) a walk π_{r} in G_{r} and (ii) a sequence of edits (matches, substitutions, deletions, and insertions) that transform $\sigma(\pi_{\mathsf{r}})$ to q. An alignment is *optimal* if it minimizes the sum of edit costs for a given real-valued cost model $\Delta = (\Delta_{\text{match}}, \Delta_{\text{subst}}, \Delta_{\text{del}}, \Delta_{\text{ins}})$. Throughout this work, we assume that edit costs are non-negative—a pre-requisite for the correctness of A*. Further, we assume that $\Delta_{\text{match}} \leq \Delta_{\text{subst}}, \Delta_{\text{ins}}, \Delta_{\text{del}}$—a prerequisite for the correctness of our heuristic.

We note that our approach naturally works for cyclic reference graphs.

Alignment graph. In order to formalize optimal alignment as a shortest path finding problem, we rely on an *alignment graph* $G_{\mathsf{a}}^q = (V_{\mathsf{a}}^q, E_{\mathsf{a}}^q)$. Its nodes V_{a}^q are *states* of the form $\langle v, i\rangle$, where $v \in V_{\mathsf{r}}$ is a node in the reference graph and $i \in \{0, \ldots, |q|\}$ corresponds to a position in the read q. Its edges E_{a}^q are selected

such that any path π_{a} in G^q_{a} from $\langle u, 0\rangle$ to $\langle v, i\rangle$ corresponds to an alignment of the first i letters of q to G_{r}. Further, the edges are weighted, which allows us to define an *optimal alignment* of a read $q \in \Sigma^*$ as a shortest path π_{a} in G^q_{a} from $\langle u, 0\rangle$ to $\langle v, |q|\rangle$, for any $u, v \in V_{\mathsf{r}}$. Fig. 1 formally defines the edges E^q_{a}.

2.2 A* algorithm for finding a shortest path

The A* algorithm is a shortest path algorithm that generalizes Dijkstra's algorithm by directing the search towards the target. Given a weighted graph $G = (V, E)$, the A* algorithm finds a shortest path from sources $S \subseteq V$ to targets $T \subseteq V$. To prioritize paths that lead to a target, it relies on an admissible heuristic function $h\colon V \to \mathbb{R}_{\geq 0}$, where $h(v)$ estimates the remaining length of the shortest path from a given node $v \in V$ to a target $t \in T$.

Algorithm. In a nutshell, the A* algorithm maintains a set of *explored* nodes, initialized by all possible starting nodes S. It then iteratively *expands* the explored state v with lowest estimated total cost $f(v)$ by exploring all its neighbors. Here, $f(v) := g(v) + h(v)$, where $g(v)$ is the distance from $s \in S$ to v, and $h(v)$ is the estimated distance from v to $t \in T$. When the A* algorithm expands a target node $t \in T$, it reconstructs the path leading to t and returns it. For completeness, App. A.1 provides an implementation of A*.

Admissibility. The A* algorithm is guaranteed to find a shortest path if its heuristic h provides a lower bound on the distance to the closest target, often referred to as h being *admissible* or optimistic.

Further, the performance of the A* algorithm relies critically on the choice of h. Specifically, it is crucial to have low estimates for the optimal paths but also to have high estimates for suboptimal paths.

Discussion. To summarize, we use the A* algorithm to find a shortest path from $\langle u, 0\rangle$ to $\langle v, |q|\rangle$ in G^q_{a}. To guarantee optimality, its heuristic function $h\langle v, i\rangle$ must provide a lower bound on the shortest distance from state $\langle v, i\rangle$ to a terminal state of the form $\langle w, |q|\rangle$. Equivalently, $h\langle v, i\rangle$ should lower bound the minimal cost of aligning $q[i{:}]$ to G_{r} starting from v, where $q[i{:}]$ denotes the suffix of q starting at position i (0-indexed). The key challenge is thus finding a heuristic that is not only admissible but also yields favorable performance.

3 Seed heuristic

We instantiate the A* algorithm with a novel, domain-specific *seed heuristic* which allows to quickly align reads to a general reference graph. We first intuitively explain the seed heuristic and showcase it on a simple example (§3.1). Then, we formally define the heuristic and prove its admissibility (§3.2). Finally, we adapt our approach to rely on a trie, which leads to a critical speedup (§3.3).

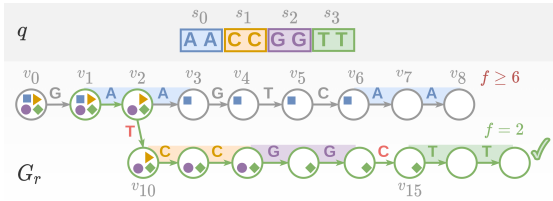

Fig. 2: A toy overview example using the seed heuristic to align a read q to a reference graph G_r. The read is split into four colored seeds, where their corresponding crumbs are shown inside reference graph nodes as symbols with matching color. The optimal alignment is highlighted as a green path ending with a tick (✓) and includes one substitution (T→A) and one deletion (C).

3.1 Overview

Fig. 2 showcases the seed heuristic on an overview example. It shows a read q to be aligned to a reference graph G_r. Our goal is to find an optimal alignment starting from an arbitrary node $v \in G_\mathsf{r}$. For simplicity of the exposition, we assume unit edit costs $\Delta = (0, 1, 1, 1)$, which we generalize in §3.2.

Intuition. The A* search requires us to provide a lower bound of the remaining path cost from a state $\langle v, i\rangle$ to a target state. Clearly, to align the whole query, each of the remaining seeds (i.e. at or after position i in q) has to be eventually aligned. The intuition underlying the seed heuristic is to punish the state for the absence of any foreseeable match of each remaining seed. Notice that the order of the seeds is not directly taken into account.

In order to quickly check if a seed s can lead to a match, we follow a procedure similar to the one used by Hansel and Gretel who were placing breadcrumbs to find their trail back home. Before aligning a query, we will precompute all *crumbs* from all seeds so that not finding a crumb for a seed s on node v indicates that seed s could not be matched exactly before the query is fully aligned continuing from v. This way, assuming that a shortest path includes many seed matches, the crumbs will direct the A* search along with it.

If a crumb from an expected seed is missing in node v, its corresponding seed s could not possibly be aligned exactly and this will incur the cost of at least one substitution, insertion, or deletion. Assuming unit edit costs, $h\langle v, i\rangle$ yields a lower bound on the cost for aligning $q[i{:}]$ starting from v by simply returning the number of missing expected crumbs in v.

Crumbs precomputation example. Fig. 2 shows four seeds as colored sections of length 2 each, and represents their corresponding crumbs as ■, ▶, ● and ◆, respectively. Four crumbs are expected if we start at v_2, but ■ is missing, so $h\langle v_2, 0\rangle = 1$. Analogously, if we reach v_2 after aligning one letter from the read, we expect 3 crumbs (except ■), and we find them all in v_2, so $h\langle v_2, 1\rangle = 0$. To precompute the crumbs for each seed, we first find all positions in G_r from which the seed aligns exactly. Fig. 2 shows these exact matches as colored sec-

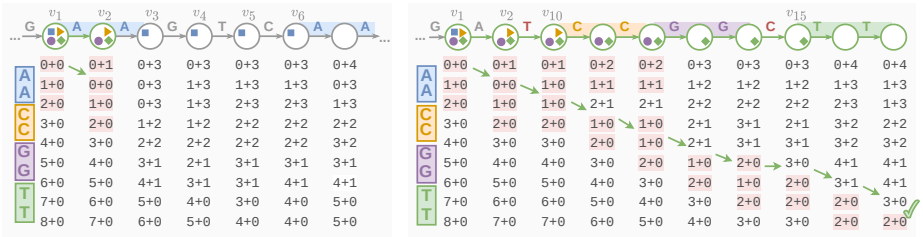

Fig. 3: Exploration of G^q_{a}, searching for a shortest path from the first to the last row using the seed heuristic. The table entry in the ith row (zero indexed) below node v shows $g\langle v,i\rangle + h\langle v,i\rangle$, where $g\langle v,i\rangle$ is the shortest distance from any starting state $\langle u,0\rangle$ to $\langle v,i\rangle$. States that may[1] be expanded by the A* algorithm are highlighted in pink, and the rest of the states are shown for completeness even though they are never expanded. The shortest path corresponding to the best alignment is shown with green arrows (→).

tions of G_{r}. Then, from each match we traverse G_{r} backwards and add crumbs to nodes that could lead to these matches. For example, because seed `CC` can be matched starting in node v_{10}, crumbs ▶ are placed on all nodes leading up to v_{10}. Similarly, seed `AA` has two exact matches, one starting in node v_0 and one starting in node v_6. However, we only add crumbs ■ to nodes v_0, v_1, and v_3–v_6, but not to node v_2. This is because v_2 is (i) strictly after the beginning of the match of `AA` at v_1 and (ii) too far before the match of `AA` at v_6. Specifically, any alignment starting from node v_2 and still matching `AA` at v_6 would induce an overall cost of 4 (it would require deleting the 4 letters A, G, T, and C). Even without a crumb ■ on v_2, our heuristic provides a lower bound on the cost of such an alignment: it never estimates a cost of more than 4, the number of seeds.

Guiding the search example. Fig. 3 demonstrates how $h\langle v,i\rangle$ guides the A* algorithm towards the shortest path by showing which states may be expanded when using the seed heuristic. Specifically, the unique optimal alignment in Fig. 2 starts from node v_1, continues to v_2, and then proceeds through node v_{10} (instead of v_3).

While the seed heuristic initially explores all states of the form $\langle v,0\rangle$ (we discuss in §3.3 how to avoid this by using a trie), it skips expanding any state that involves nodes v_3–v_8. This improvement is possible because all these explored states are penalized by the seed heuristic by at least 3, while the shortest path of cost 2 will be found before considering states on nodes v_3–v_8. Here, the heuristic function accurately predicts that expanding v_{10} may eventually lead to an exact alignment of seeds `CC`, `GG` and `TT`, while expanding v_3 may not lead to an alignment of either seed. In particular, the seed heuristic is not misled by the

[1] Depending on how the A* algorithm handles tie-braking, different sets of states could be explored. For simplicity, we show all states that *could potentially* be explored.

short-term benefit of correctly matching A in v_2, and instead provides a long-term recommendation based on the whole read. Thus, even though the walk to v_3 aligns exactly the first two letters of q, A* does not expand v_3 because the seed heuristic guarantees that the future cost will be at least 3.

3.2 Formal definition

Next, we formally define the seed heuristic function $h\langle v, i\rangle$. Overall, we want to ensure that $h\langle v, i\rangle$ is admissible, i.e., that it is a lower bound on the cost of a shortest path from $\langle v, i\rangle$ to some $\langle w, |q|\rangle$ in G^q_{a}.

Seeds. We split read $q \in \Sigma^*$ into a set *Seeds* of non-overlapping seeds $s_0, \ldots, s_{|Seeds|-1} \in \Sigma^*$. For simplicity, in this work we ensure that all seeds have the same length and are consecutive, i.e., we split q into substrings $s_0 \cdot s_1 \cdots s_{|Seeds|-1} \cdot t$, where all s_j are seeds of length k and we ignore the suffix t of q, which is shorter than k. We note that our approach could be trivially generalized to seeds of different lengths or non-consecutive seeds as long as they do not overlap. An interesting future work item is investigating how different choices of seeds affect the performance of our approach, and selecting seeds accordingly.

Matches. For each seed $s \in Seeds$, we locate all nodes $u \in M(s)$ in the reference graph that can be the start of an exact match of s:

$$M(s) := \{u \in V_{\mathsf{r}} \mid \exists \text{walk } \pi \text{ starting from } u \in G_{\mathsf{r}} \text{ and spelling } \sigma(\pi) = s\}.$$

To compute $M(s)$ efficiently, we leverage the trie introduced in §3.3.

Crumbs. For seed s_j starting at position i in q, we place crumbs on all nodes $u \in V_{\mathsf{r}}$ which can reach a node $v \in M(s_j)$ using less than $i + n_{\text{del}}$ edges:

$$C(s) := \{u \in V_{\mathsf{r}} \mid \exists v \in M(s) \colon dist(u, v) < i + n_{\text{del}}\},$$
$$\text{where } dist(u, v) \text{ is the length of a shortest walk from } u \text{ to } v.$$

Later in this section, we will select n_{del} to ensure that if an alignment uses more than n_{del} deletions, its cost must be so high that the heuristic function is trivially admissible.

To compute $C(s)$ efficiently, we can traverse the reference graph backwards from each $v \in M(s)$ by a backward breadth-first-search (BFS).

Heuristic. Let $Seeds_{\geq i}$ be the set of seeds that start at or after position i of the read, formally defined by $Seeds_{\geq i} := \{s_j \mid \lceil i/k \rceil \leq j < |Seeds|\}$. This allows us to define the number of expected but missing crumbs in state $\langle v, i\rangle$ as $misses\langle v, i\rangle := \left|\{v \notin C(s) \mid s \in Seeds_{\geq i}\}\right|$. Finally, we define the seed heuristic as

$$h\langle v, i\rangle = (|q| - i) \cdot \Delta_{\text{match}} + misses\langle v, i\rangle \cdot \delta_{min}, \tag{1}$$
$$\text{for } \delta_{min} = \min(\Delta_{\text{subst}} - \Delta_{\text{match}}, \Delta_{\text{del}}, \Delta_{\text{ins}} - \Delta_{\text{match}}), \tag{2}$$

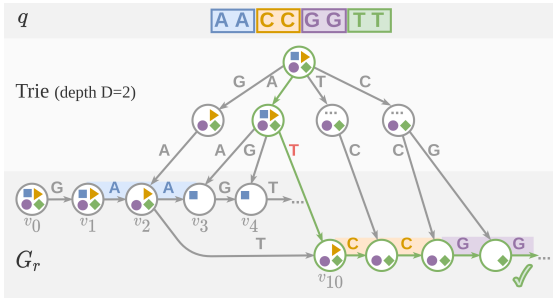

Fig. 4: Reference graph from Fig. 2, extended by a trie of depth $D = 2$. For simplicity, the reverse-complement reference graph and parts marked by "..." are omitted.

Intuitively, Eq. (1) reflects that the cost of aligning each remaining letter from $q[i{:}]$ is at least Δ_{match}. In addition, every inexact alignment of a seed induces an additional cost of at least δ_{min}. Specifically, every substitution costs Δ_{subst} but requires one less match; every deletion costs Δ_{del}; and every insertion costs Δ_{ins} but also requires one less match.

We note that $h\langle v, i\rangle$ implicitly also depends on the reference graph G_{r}, the read q, the set of seeds, and the edit costs Δ.

In order for an alignment with at least n_{del} deletions to have a cost so high that the heuristic function is trivially admissible, we ensure $n_{\text{del}} \cdot \Delta_{\text{del}} \geq h\langle v, i\rangle$ by defining

$$n_{\text{del}} := \left\lceil \frac{|q| \cdot \Delta_{\text{match}} + |Seeds| \cdot \delta_{min}}{\Delta_{\text{del}}} \right\rceil . \tag{3}$$

In Thm. 1, we show that $h\langle v, i\rangle$ is admissible, ensuring that our heuristic yields optimal alignments.

Theorem 1 (Admissibility). *The seed heuristic $h\langle v, i\rangle$ is admissible.*

Proof. We provide a proof for Thm. 1 in App. A.2. □

3.3 Trie index

Considering all nodes $v \in V_{\mathsf{r}}$ as possible starting points for the alignment means that the A* algorithm would explore all states of the form $\langle v, 0\rangle$, which immediately induces a high overhead of $|V_{\mathsf{r}}|$. In line with previous works [11, 13], we avoid this overhead by complementing the reference graph with a trie index to produce a new graph $G_{\mathsf{r}}^{+} = (V_{\mathsf{r}}^{+}, E_{\mathsf{r}}^{+})$, where V_{r}^{+} is the union of the reference graph nodes V_{r} and the new trie vertices, and E_{r}^{+} is the union of E_{r}, the trie edges, and edges connecting the trie leafs with reference nodes. Note that constructing this trie index is a one-time pre-processing step that can be reused for multiple queries.

Since we want to also support aligning reverse-complement reads by starting from the trie *root*, we build the trie not only from the original reference graph and also from its reverse-complement.

Intuition. Fig. 4 extends the reference graph G_{r} from Fig. 2 with a trie. Here, any path in the reference graph uniquely corresponds to a path starting from the trie *root* (the top-most node in Fig. 4). Thus, in order to find an optimal alignment, it suffices to consider paths starting from the trie *root*, by using state $\langle root, 0\rangle$ as the only source for the A* algorithm. Note that if the reference graph branches frequently, the number of paths with length D may rise exponentially, leading to an exponential number of trie leaves. To counteract this exponential growth, we can select D logarithmically small, as $\log_4 N$.

For a more thorough introduction to the trie and its construction, see [11]. Importantly, our placement of crumbs (§3.2) generalizes directly to reference graphs extended with a trie (see also Fig. 4).

Reusing the trie to find seed matches. As a second usage of the trie, we can also exploit it to efficiently locate all matches $M(s)$ of a given seed s. In order to find all nodes where a seed match begins, we align (without errors) $\bar{s}$, the reverse-complement of s. To this end, we follow all paths spelling $\bar{s}$ starting from the *root*—the final nodes of these paths then correspond to nodes in $M(s)$. We ensure that the seed length $|s|$ is not shorter than the trie depth D, so that matching all letters in $\bar{s}$ ensures that we eventually transition from a trie leaf to the reference graph.

Optimization: skip crumbs on the trie. Generally, we aim to place as few crumbs as possible, in order to both reduce precomputation time and avoid misleading the A* algorithm by unnecessary crumbs. In the following, we introduce an optimization to avoid placing crumbs on trie nodes that are "too close" to the match of their corresponding seed so they cannot lead to an optimal alignment.

Specifically, when traversing the reference graph backwards to place crumbs for a match of seed s starting at node w, we may "climb" from a reference graph node u to a trie node u' backwards through an edge that otherwise leads from the trie to the reference. Assuming s starts at position i in the read, we have already established that we can only consider nodes u that can reach w with less than $i + n_{\text{del}}$ edges (see §3.2). Here, we observe that it is sufficient to only climb into the trie from nodes u that can reach w using more than $i - n_{\text{ins}} - D$ edges, for

$$n_{\text{ins}} := \left\lceil \frac{|q| \cdot \Delta_{\text{match}} + |Seeds| \cdot \delta_{min}}{\Delta_{\text{ins}}} \right\rceil . \tag{4}$$

We define n_{ins} analogously to n_{del} to ensure that n_{ins} insertions will induce a cost that is always higher than $h\langle u, i\rangle$. We note that we can only avoid climbing into the trie if all paths from u to w are too short, in particular the longest one.

The following Lemma 1 shows that this optimization preserves optimality.

Lemma 1 (Admissibility when skipping crumbs) *The seed heuristic remains admissible when crumbs are skipped in the trie.*

Proof. We provide a proof for Lemma 1 in App. A.2. □

In order to efficiently identify all nodes u that can reach w by using more than $i - D - n_{\text{ins}}$ edges (among all nodes at a backward-distance at most $i + n_{\text{del}}$ from w), we use topological sorting: considering only nodes at a backward-distance at most $i + n_{\text{del}}$ from w, the length of a longest path from a node v to w is (i) ∞ if v lies on a cycle and (ii) computable from nodes closer to w otherwise.

4 Evaluations

In the following, we demonstrate that our approach aligns faster than existing optimal aligners due to its superior scaling. Specifically, we address the following research questions:

Q1 What speedup can the seed heuristic achieve?
Q2 How does the seed heuristic scale with reference size?
Q3 How does the seed heuristic scale with read length?

The modes of operation which we analyze include both short (Illumina) and long (HiFi) reads to be aligned on on both linear and graph references.

4.1 Seed heuristic implementation

Both the seed heuristic and the prefix heuristic reuse the same free and open source C++ codebase of the ASTARIX aligner [11]. It includes a simple implementation of a graph and trie data structure which is not optimized for memory usage. In order to easily align reverse complement reads, the reverse complement of the graph is stored alongside its straight version. The shortest path algorithm only constructs explored states explicitly, so most states remain implicitly defined and do not cause computational burden.

Both heuristics benefit from a default optimization in ASTARIX called *greedy matching* [11, Section 4.2] which skips adding a state to the A* queue when only one edge is outgoing from a state and the upcoming read and reference letters match.

4.2 Setting

All experiments were executed on a commodity machine with an Intel Core i7-6700 CPU @ 3.40GHz processor, using a memory limit of 20 GB and a single thread. We note that while multiple tools support parallelization when aligning a single read, all tools can be trivially parallelized to align multiple reads in parallel.

Compared aligners. We compare the novel seed heuristic to prefix heuristic (both heuristics are implemented in ASTARIX), GRAPHALIGNER, PASGAL, and

Vargas. We provide the versions and commands used to run all aligners and read simulators in App. A.3. We note that we do not compare to VG [1] and HGA [10] since the optimal alignment functionality of VG is meant for debugging purposes and has been shown to be inferior to other aligners [10, Tab. 4], and HGA makes use of parallelization and a GPU but has been shown to be superseded in the single CPU regime [10, Fig. 9]. PaSGAL and Vargas are compiled with AVX2 support. We execute the prefix heuristic with the default lookup depth $d = 5$.

Seed heuristic parameters. The choice of parameters for the seed heuristic influences its performance. Increasing the trie depth increases its number of nodes, but decreases the average out-degree of its leaves. We set the trie depth for all experiments to $D = 14 \approx \log_4 N$.

Shorter seeds are more likely to be matched perfectly by an optimal alignment, as they contain less letters that could be subject to edits. Thus, shorter seeds can tolerate higher error rate, but at the cost of slower precomputation due to a higher total number of matches, and slower alignment due to more off-track matches. In our experiments, we use seed lengths of $k = 25$ for Illumina reads and $k = 150$ for HiFi reads.

Data. We aligned reads to two different reference graphs: a linear *E. coli* genome (strain: K-12 substr. MG1655, ASM584v2) [14], with length of 4 641 652bp (approx. 4.7Mbp), and a variant graph with the Major Histocompatibility Complex (MHC), of size 5 318 019bp (approx. 5Mbp), taken from the evaluations of PaSGAL [8]. Additionally, we extracted a path from MHC in order to create a linear reference MHC-linear of length 4 956 648bp which covers approx. 93% of the original graph. Because of input graph format restrictions, we execute GraphAligner, Vargas and PaSGAL only on linear references in FASTA format (*E. coli* and the MHC-linear), while we execute the seed heuristic and the prefix heuristic on the original references (*E. coli* and MHC). This yields an underestimation of the speedup of the seed heuristic, as we expect the performance on MHC-linear to be strictly better than on the whole MHC graph.

To generate both short Illumina and long HiFi reads, we relied on two tools. We generated short single-end 200bp Illumina MSv3 reads using ART simulator [15]. We generated long HiFi reads using the script randomreads.sh[2] with sequencing lengths 5–25kbp and error rates 0.3%, which are typical for HiFi reads.

Edit costs. We execute AStarix with edit costs typical to the corresponding sequencing technology: $\Delta = (0, 1, 5, 5)$ for Illumina reads and $\Delta = (0, 1, 1, 1)$ for HiFi reads. As the performance of DP-based tools is independent of edit costs, we are using the respective default edits costs when executing GraphAligner, PaSGAL and Vargas.

Metrics. We report the performance of aligners in terms of runtime per one thousand aligned base pairs [s/kbp]. Since we measured runtime end-to-end (in-

[2] https://github.com/BioInfoTools/BBMap/blob/master/sh/randomreads.sh

Table 1: Runtime and memory comparison of optimal aligners. Simulated Illumina and HiFi reads are aligned to linear *E. coli* and graph MHC references. The runtime of the seed heuristic is expressed as absolute time per aligned kbp, while the other aligners are compared to the seed heuristic at a fold change. Additionally, the fraction of explored states is shown for the seed heuristic and the prefix heuristic.

	Illumina		**HiFi**		
Tool	*E. coli*	MHC	*E. coli*	MHC	
Seeds heuristic	0.019	0.041	0.001	0.002	s/kbp
(this work)	2.4	2.6	2.4	1.7	GB (max used)
	99.9996	99.9981	99.9989	99.9984	% skipped states
Prefix heuristic	269x	180x	n/a	n/a	x slowdown
	7.7	9.6	>20	>20	
	99.9501	99.9501	n/a	n/a	
GraphAligner	424x	212x	118x	64x	
	0.2	0.2	3.6	3.4	
Vargas	133x	67x	1 413x	705x	
	<0.1	<0.1	7.3	7.3	
PaSGAL	263x	130x	1 367x	736x	
	0.6	0.6	0.6	0.6	

cluding loading reference graphs and reads from disk, and building the trie index for AStarix), we ensured that alignment time dominates the total runtime by providing sufficiently many reads to each aligner. In order to prevent excessive runtimes for slower tools, we used a different number of reads for each tool and explicitly report them for each experiment.

Since shortest path approaches skip considerable parts of the computation performed by aligners based on dynamic programming, the commonly used Giga Cell Updates Per Second (GCUPS) metric is not adequate for measuring performance in our context.

We measured used memory by `max_rss` (Maximum Resident Set Size) from Snakemake[3].

We do not report accuracy or number of unaligned reads, as all evaluated tools align all reads with guaranteed optimality according to edit distance. We note that Vargas reports a warning that some of its alignments are not optimal—we ignore this warning and focus on its performance.

[3] https://snakemake.readthedocs.io/en/stable/

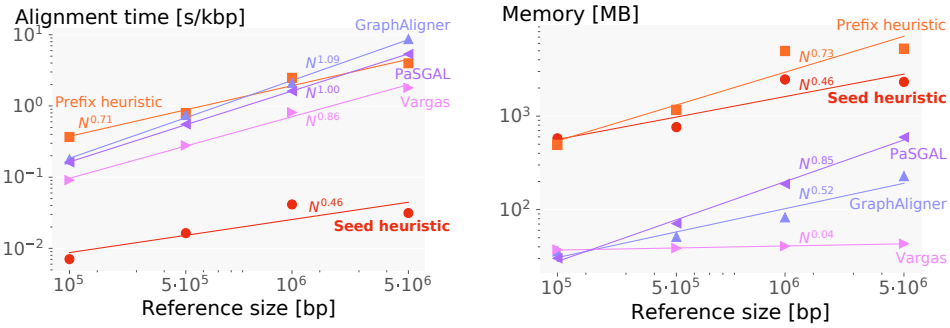

Fig. 5: Performance degradation with reference size for **Illumina reads**. Log-log plots of total alignment time (left) and memory usage (right) show the scaling difference between aligners.

4.3 Q1: Speedup of the seed heuristic

Table 1 shows that the seed heuristic achieves a speedup of at least 60 times compared to all considered aligners, across all regimes of operation: both Illumina and HiFi reads aligned on *E. coli* and MHC references.

In the Illumina experiments, the seed heuristic is given 100k reads, while the other tools are given 1000 reads. In the HiFi experiments, the seed heuristic is given reads that cover the reference 10 times, and the other tools are given reads of coverage 0.1.

The key reason for the speedup of the seed heuristic is that on all four experiments, it skips $\geq 99.99\%$ of the Nm states computed by the DP approaches of GraphAligner, PaSGAL, and Vargas. This fraction accounts for both the explored states during the A* algorithm, and the number of crumbs added to nodes during precomputation for each read.

The prefix heuristic exceeded the available memory on HiFi reads, as it is not designed for long reads.

4.4 Q2: Scaling with reference size

In order to study the scaling of the aligners in terms of the reference size, we extracted prefixes of increasing length from MHC-linear. We then generated reads from each prefix, and ran all tools on all prefixes with the corresponding reads.

Illumina reads. Fig. 5 shows the runtime scaling and memory usage for Illumina reads. The seed heuristic was provided with 10k reads, while other tools were provided with 1k reads. The runtime of GraphAligner, PaSGAL and Vargas grow approximately linearly with the reference length, whereas the runtime of the seed heuristic grows roughly with $\sqrt[2]{N}$, where N is the reference size. Even on relatively small graphs like MHC, the speedup of the seed heuristic

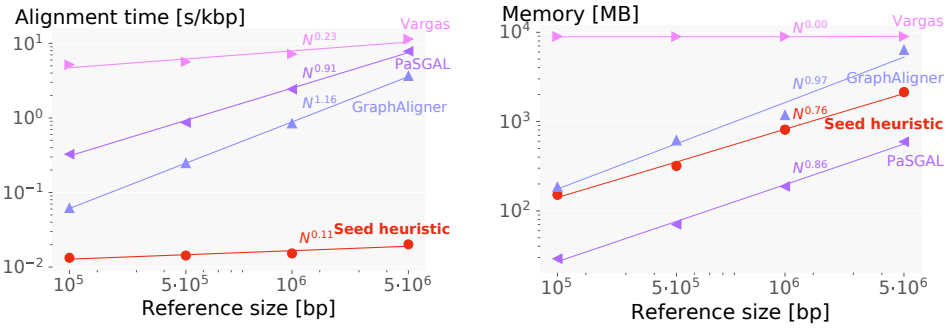

Fig. 6: Performance degradation with reference size for **HiFi reads**. Log-log plots of total alignment time (left) and memory usage (right) show the scaling difference between aligners. Linear best fits correspond to polynomials of varying degree.

reaches 200 times. Note that the scaling of the prefix heuristic is substantially worse than the seed heuristic since the 200bp reads are outside of its operational capabilities.

HiFi reads. Fig. 6 shows the runtime scaling and memory usage for HiFi reads. The respective total lengths of all aligned reads are 5Mbp for the seed heuristic, 500kbp for GraphAligner, and 100kbp for Vargas and PaSGAL. We do not show the prefix heuristic, since it explores too many states and runs out of memory. Crucially, we observe that the runtime of the seed heuristic is almost independent of the reference size, growing as $N^{0.11}$. We believe this improved trend compared to short reads is because the seed heuristic obtains better guidance on long reads, as it can leverage information from the whole read.

For both Illumina and HiFi reads, we observe near-linear scaling for PaSGAL and GraphAligner as expected from the theoretical $\mathcal{O}(Nm)$ runtime of the DP approaches. We conjecture that the runtime of Vargas for long reads is dominated by the dependence from the read length, which is why on HiFi reads we observe better than linear runtime dependency on N but very large runtime. The current alignment bottleneck of AStarix-seeds is its memory usage, which is distributed between remembering crumbs and holding a queue of explored states.

4.5 Q3: Scaling with read length

Fig. 7 shows the runtime and memory scaling with increasing length of aligned HiFi reads on MHC reference. Here we used reads with a total length of 100Mbp for the seed heuristic and 2Mbp for all other aligners.

The scaling of the seed heuristic in terms of read length is slightly worse than that of other aligners. However, this is compensated by its superior scaling in terms of reference size (see §4.4), leading to an overall better absolute runtime.

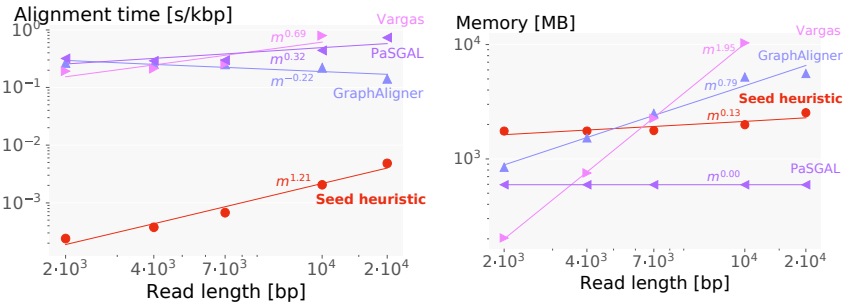

Fig. 7: Performance degradation with HiFi read length. Log-log plots of total alignment time (left) and memory usage (right) show the scaling difference between aligners.

We note that the memory usage of the seed heuristic does not heavily depend on the read length and for reads longer than 10kbp, it is superior to GraphAligner and Vargas.

5 Acknowledgements

We want to thank the anonymous reviewers for the valuable feedback. The first author is grateful for the Vipassana meditation which inspired the idea to punish paths for the absence of foreseeable seed matches.

6 Conclusion

We have presented an optimal read aligner based on the A* algorithm instantiated with a novel seed heuristic which guides the search by preferring crumbs on nodes that lead towards optimal alignments even for long reads.

The memory usage is currently limiting the application of AStarix for bigger references due to the size of the trie index. A remaining challenge is designing a heuristic function able to handle not only long but also noisier reads, such as the uncorrected PacBio reads that may reach 20% of mistakes. Possible improvements of the seed heuristic may include inexact matching of seeds, careful choice of seed positions, and accounting for the seeds order.

Bibliography

[1] E. Garrison, J. Sirén, A. M. Novak, G. Hickey, J. M. Eizenga, E. T. Dawson, W. Jones, S. Garg, C. Markello, M. F. Lin, B. Paten, and R. Durbin, "Variation graph toolkit improves read mapping by representing genetic variation in the reference," *Nature Biotechnology*, 2018.

[2] G. Kucherov, "Evolution of biosequence search algorithms: a brief survey," *Bioinformatics*, vol. 35, no. 19, pp. 3547–3552, 2019.

[3] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of Molecular Biology*, 1990.

[4] B. Langmead and S. L. Salzberg, "Fast gapped-read alignment with Bowtie 2," *Nature Methods*, 2012.

[5] H. Li and R. Durbin, "Fast and accurate short read alignment with Burrows–Wheeler transform," *Bioinformatics (Oxford, England)*, 2009.

[6] M. Equi, R. Grossi, V. Mäkinen, A. Tomescu, *et al.*, "On the complexity of string matching for graphs," in *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2019.

[7] C. A. Darby, R. Gaddipati, M. C. Schatz, and B. Langmead, "Vargas: heuristic-free alignment for assessing linear and graph read aligners," *Bioinformatics*, vol. 36, no. 12, pp. 3712–3718, 2020.

[8] C. Jain, S. Misra, H. Zhang, A. Dilthey, and S. Aluru, "Accelerating Sequence Alignment to Graphs," in *International Parallel and Distributed Processing Symposium (IPDPS)*, 2019. ISSN: 1530-2075.

[9] M. Rautiainen, V. Mäkinen, and T. Marschall, "Bit-parallel sequence-to-graph alignment," *Bioinformatics*, 2019.

[10] Z. Feng and Q. Luo, "Accelerating sequence-to-graph alignment on heterogeneous processors," in *50th International Conference on Parallel Processing*, pp. 1–10, 2021.

[11] P. Ivanov, B. Bichsel, H. Mustafa, A. Kahles, G. Rätsch, and M. T. Vechev, "AStarix: Fast and Optimal Sequence-to-Graph Alignment," in *RECOMB 2020*, 2020.

[12] M. Rautiainen and T. Marschall, "Aligning sequences to general graphs in $O(V+mE)$ time," preprint, Bioinformatics, Nov. 2017.

[13] G. Dox and J. Fostier, "Efficient algorithms for pairwise sequence alignment on graphs," Master's thesis, Ghent university, 2018.

[14] K. L. Howe, B. Contreras-Moreira, N. De Silva, G. Maslen, W. Akanni, J. Allen, J. Alvarez-Jarreta, M. Barba, D. M. Bolser, L. Cambell, *et al.*, "Ensembl Genomes 2020–enabling non-vertebrate genomic research," *Nucleic Acids Research*, 2020.

[15] W. Huang, L. Li, J. R. Myers, and G. T. Marth, "ART: a next-generation sequencing read simulator," *Bioinformatics*, vol. 28, no. 4, pp. 593–594, 2011.

A Appendix

Algorithm 1 A* algorithm

```
function A*(G: Graph, S: Sources, T: Targets, h: Heuristic function)
    g ← Map: (Nodes → ℝ≥0)                    ▷ Shortest paths lengths to explored nodes
    f ← Map: (Nodes → ℝ≥0)                    ▷ f(u) = g(u) + h(u)
    Q ← MinPriorityQueue(priority = f)        ▷ Priorities according to f
    for all s ∈ S do
        g[s] ← 0.0, f[s] ← 0.0
        Q.push(s)                             ▷ Initially, explore all s ∈ S
    while Q ≠ ∅ do
        curr ← Q.pop()                        ▷ Get state with minimal f to be expanded
        if curr ∈ T then
            return BacktrackPath(curr)        ▷ Reconstruct a walk to curr
        for all (curr, next, cost) ∈ G.outgoingEdges(curr) do
            g_next ← g[curr] + cost
            f̂_next ← g_next + h(next)         ▷ Candidate value for f[next]
            if f̂_next < f[next] then
                g[next] ← g_next
                f[next] ← f̂_next
                Q.push(next)                  ▷ Explore state next
    assert False                              ▷ Cannot happen if T is reachable from S
```

A.1 A* Algorithm

Algorithm 1 shows an implementation of the A* algorithm, taken from [11, §A.1]. We omit the implementation of BacktrackPath for simplicity.

A.2 Proofs

In the following, we provide proofs for Thm. 1 and Lemma 1, restated here for convenience.

Theorem 1 (Admissibility). *The seed heuristic $h\langle v, i\rangle$ is admissible.*

Proof. Let A be an optimal alignment of $q[i{:}]$ starting from $v \in G_{\mathsf{r}}$. We will prove that the cost of A is at least $h\langle v, i\rangle$.

If A contains at least n_{del} deletions, its cost is at least $n_{\text{del}} \cdot \Delta_{\text{del}}$, which is at least $|q| \cdot \Delta_{\text{match}} + |\mathit{Seeds}| \cdot \delta_{min}$ by plugging in n_{del} from Eq. (3). This is an upper bound for $h\langle v, i\rangle$, which we observe after maximizing $h\langle v, i\rangle$ by substituting $i = 1$ and $\mathit{misses} = |\mathit{Seeds}|$ into Eq. (1), which concludes the proof in this case.

Otherwise, A contains less than n_{del} deletions. If we interpret A as a path in G^q_{a}, we first observe that A must spell $q[i{:}]$. Thus, A must in particular also contain all seeds $s_j \in \mathit{Seeds}_{\geq i}$ as substrings. We then split A into *subalignments*

$A_{-1}, A_0, \ldots, A_p$, selected such that $A_0, \ldots, A_{p-1}$ spell the seeds $s_j \in \mathit{Seeds}_{\geq i}$, and A_{-1} and A_p spell the prefix and suffix of $q[i{:}]$ which do not cover any full seed.

This ensures that we can compute a lower bound on the cost of A as follows:

$$\begin{aligned}
\mathrm{cost}(A) &= \sum_{j=-1}^{p} \mathrm{cost}(A_k) && (5)\\
&\geq \sum_{j=-1}^{p} |\sigma(A_k)| \cdot \Delta_{\mathrm{match}} + \sum_{j=0}^{p-1} \left\{ \begin{matrix} 0 & \text{if } v \in C\big(s_{\lceil i/k \rceil + j}\big) \\ \delta_{min} & \text{if } v \notin C\big(s_{\lceil i/k \rceil + j}\big) \end{matrix} \right\} && (6)\\
&= (|q| - i) \cdot \Delta_{\mathrm{match}} + \big|\{v \notin C(s) \mid s \in \mathit{Seeds}_{\geq i}\}\big| \cdot \delta_{min} && (7)\\
&= h\langle v, i\rangle && (8)
\end{aligned}$$

Here, Eq. (5) follows from our decomposition of A. If we ignore the right-hand side in Eq. (6) (right of "+"), the inequality follows because matching all letters is the cheapest method to align any string. The right-hand side follows from a more precise analysis for subalignments A_k that spell a seed $s_{\lceil i/k \rceil + j}$ without a corresponding crumb in v. The absence of such a crumb indicates that no exact match of $s_{\lceil i/k \rceil + j}$ in G_{r} can be reached within less than $i + n_{\mathrm{del}}$ steps from v. However, because A contains less than n_{del} deletions, A_k must start within less than $i + n_{\mathrm{del}}$ steps from v. Thus, A_k does not align $s_{\lceil i/k \rceil + j}$ exactly, meaning that it introduces a cost of at least δ_{min}.

Eq. (7) follows from observing that $A_{-1}, \ldots, A_p$ have a total length of $|q| - i$, and observing that the right-hand sum adds up δ_{min} for every expected but missing crumb. Finally, Eq. (8) follows from our definition of $h\langle v, i\rangle$, concluding the proof. □

Lemma 1 (Admissibility when skipping crumbs) *The seed heuristic remains admissible when crumbs are skipped in the trie.*

Proof. Consider a reference graph with a match of seed s starting in node w. Now, consider a node v that cannot reach w using more than $i - D - n_{\mathrm{ins}}$ edges. We can then show that a trie node v' with a path to v does not require a crumb for the match of s in node w.

Specifically, any path from *root* through nodes v' and v to node w has total length greater or equal $i - n_{\mathrm{ins}}$. Thus, matching s at w requires at least n_{ins} insertions. Hence, the cost of such a path is at least $n_{\mathrm{ins}} \cdot \Delta_{\mathrm{ins}} = |q| \cdot \Delta_{\mathrm{match}} + n \cdot \delta_{min}$. Observing that this is an upper bound for $h\langle v, i\rangle$ concludes the proof. □

A.3 Versions, commands, parameters for running all evaluated approaches

In the following, we provide details on how we executed the newest versions of the tools discussed in §4:

Executing AStarix.
Obtained from https://github.com/eth-sri/astarix

Seed heuristic

Command `astarix align-optimal -D 14 -a astar-seeds -seeds_len l -f reads.fq -g graph.gfa >output`

Prefix heuristic

Command `astarix align-optimal -D 14 -a astar-prefix -d 5 -f reads.fq -g graph.gfa >output`

For aligning Illumina reads, `astarix` is used with additional `-M 0 -S 1 -G 5` and for HiFi reads with `-M 0 -S 1 -G 1` which better match the error rate profiles for these technologies.

Executing other tools.

Vargas

Obtained from https://github.com/langmead-lab/vargas (v0.2, commit `b1ad5d9`)

Command `vargas align -g graph.gdef -U reads.fq -ete`

Comment `-ete` stands for end to end alignment; default is 1 thread

PaSGAL

Obtained from https://github.com/ParBLiSS/PaSGAL (commit `9948629`)

Command `PaSGAL -q reads.fq -r graph.vg -m vg -o output -t 1`

Comment Compiled with AVX2.

GraphAligner

Obtained from https://github.com/maickrau/GraphAligner (v1.0.13, commit `02c8e26`)

Command `GraphAligner -seeds-first-full-rows 64 -b 10000 -t 1 -f reads.fq -g graph.gfa -a alignments.gaf >output` (commit `9948629`)

Comment `-seeds-first-full-rows` forces the search from all possible reference positions instead of using seeds; `-b 10000` sets a high alignment bandwidth; these two parameters are necessary for an optimal alignment according to the author and developer of the tool.

Simulating reads.

Illumina

```
art_illumina -ss MSv3 -sam -i graph.fasta -c N -l 200 -o
dir -rnd_seed 42
```

HiFi

```
randomreads.sh -Xmx1g build=1 ow=t seed=1 ref=graph.fa
illuminanames=t addslash=t pacbio=t pbmin=0.003
pbmax=0.003 paired=f gaussianlength=t minlength=5000
midlength=13000 maxlen=25000 out=reads.fq
```

Comment `BBMapcoverage`, https://github.com/BioInfoTools/BBMap/blob/master/sh/randomreads.sh (commit: a9ceda0)

A.4 Notations

Table 2 summarizes the notational conventions used in this work.

Table 2: Notational conventions.

Object	**Notation**
Queries	$Q = \{q_i \mid q_i \in \Sigma^m\}$
Read	$q \in Q$
Length	$m := \lvert q \rvert \in \mathbb{N}$
Position in read	$q[i] \in \Sigma$, $i \in \{0, \ldots, m-1\}$
Reference graph	$G_\mathsf{r} = (V_\mathsf{r}, E_\mathsf{r})$
Size	$\lvert G_\mathsf{r} \rvert := \lvert V_\mathsf{r} \rvert + \lvert E_\mathsf{r} \rvert \in \mathbb{N}$
Nodes	$u, v \in V_\mathsf{r}$
Number of nodes	$N := \lvert V_\mathsf{r} \rvert \in \mathbb{N}$
Edges	$e \in E_\mathsf{r} := V_\mathsf{r} \times V_\mathsf{r} \times \Sigma$
Edge letter	$\ell \in \Sigma$
Reference graph with a trie	$G_\mathsf{r}^+ = (V_\mathsf{r}^+, E_\mathsf{r}^+)$
Trie depth	$D \in \mathbb{N}_{>0}$
Alignment graph	$G_\mathsf{a}^q = (V_\mathsf{a}^q, E_\mathsf{a}^q)$
State	$\langle u, i \rangle \in V_\mathsf{a}^q := V \times \{0, \ldots, m\}$
Edges	$(\langle u, i \rangle, \langle v, j \rangle, \ell, w) \in E_\mathsf{a}^q \subseteq V_\mathsf{a}^q \times V_\mathsf{a}^q \times \Sigma_\varepsilon \times \mathbb{R}_{\geq 0}$, $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$
Edge cost	$w \in \mathbb{R}_{\geq 0}$
Alignment	$\pi \in E_\mathsf{e}^*$ and $\sigma(\pi) = q$
Alignment cost	$\mathrm{cost}(\pi) \in \mathbb{R}_{\geq 0}$
Seed heuristic	$h\langle u, i \rangle$
State	$\langle u, i \rangle$
Seed length	k
Maximum number of deletions	n_{del}
Maximum number of insertions	n_{ins}
In all graphs	$G = (V, E) \in \{G_\mathsf{r}, G_\mathsf{e}, G_\mathsf{a}^q\}$
Walk	$\pi \in G : \pi \in E^*$
Walk spelling	$\sigma(\pi) \in \Sigma^*$
Path	A walk without repeating nodes
A*	$A^\star(G, S, T, h)$
Graph	$G = (V, E)$
Nodes	$u, v \in V$
Edges	$e \in E \subseteq V \times V \times \mathbb{R}_{\geq 0}$
Source states	$S \subseteq V$
Target states	$T \subseteq V$
Heuristic function	$h \colon V \to \mathbb{R}_{\geq 0}$
Minimum cost to a target	$h^*(u)$
Explored state	A state pushed to the queue of Algorithm 1
Expanded state	A state popped from the queue of Algorithm 1