Schulz and Medvedev
Algorithms for Molecular Biology (2024) 19:19
https://doi.org/10.1186/s13015-024-00261-7

RESEARCH **Open Access**

ESKEMAP: exact sketch-based read mapping

Tizian Schulz[1,2,3*] and Paul Medvedev[4,5,6*]

Abstract

Background Given a sequencing read, the broad goal of read mapping is to find the location(s) in the reference genome that have a "similar sequence". Traditionally, "similar sequence" was defined as having a high alignment score and read mappers were viewed as heuristic solutions to this well-defined problem. For sketch-based mappers, however, there has not been a problem formulation to capture what problem an exact sketch-based mapping algorithm should solve. Moreover, there is no sketch-based method that can find all possible mapping positions for a read above a certain score threshold.

Results In this paper, we formulate the problem of read mapping at the level of sequence sketches. We give an exact dynamic programming algorithm that finds all hits above a given similarity threshold. It runs in $\mathcal{O}(|t| + |p| + \ell^2)$ time and $\mathcal{O}(\ell \log \ell)$ space, where $|t|$ is the number of k-mers inside the sketch of the reference, $|p|$ is the number of k-mers inside the read's sketch and ℓ is the number of times that k-mers from the pattern sketch occur in the sketch of the text. We evaluate our algorithm's performance in mapping long reads to the T2T assembly of human chromosome Y, where ampliconic regions make it desirable to find all good mapping positions. For an equivalent level of precision as minimap2, the recall of our algorithm is 0.88, compared to only 0.76 of minimap2.

Keywords Sequence sketching, Long-read mapping, Exact algorithm, Dynamic programming

Introduction

Read mapping continues to be one of the most fundamental problems in bioinformatics. Given a read, the broad goal is to find the location(s) in the reference genome that have a "similar sequence". Traditionally, "similar sequence" was defined as having a high alignment score and read mappers were viewed as heuristic solutions to this well-defined problem. However, the last few years has seen the community embrace sketch-based mapping methods, best exemplified by minimap2 [1] (see [2] for a survey). These read mappers work not on the original sequences themselves but on their sketches, e.g. the minimizer sketch. As a result, it is no longer clear which exact problem they are trying to solve, as the definition using an alignment score is no longer directly relevant. To the best of our knowledge, there has not been a problem formulation to capture what problem an exact sketch-based mapping algorithm should solve.

In this work, we provide a problem formulation (Section "Problem definition") and an exact algorithm to

*Correspondence:
Tizian Schulz
tizian.schulz@uni-bielefeld.de
Paul Medvedev
pzm11@psu.edu
[1] Faculty of Technology and Center for Biotechnology (CeBiTec), Bielefeld University, Bielefeld, Germany
[2] Bielefeld Institute for Bioinformatics Infrastructure (BIBI), Bielefeld University, Bielefeld, Germany
[3] Graduate School "Digital Infrastructure for the Life Sciences" (DILS), Bielefeld University, Bielefeld, Germany
[4] Department of Computer Science and Engineering, The Pennsylvania State University, University Park, USA
[5] Department of Biochemistry and Molecular Biology, The Pennsylvania State University, University Park, USA
[6] Huck Institutes of the Life Sciences, The Pennsylvania State University, University Park, USA

find all hits above a given score (Section "Algorithm for the Sketch Read Mapping Problem"). More formally, we consider the problem of taking a sketch t of a text T and a sketch p of a query P and identifying all sub-sequences of t that match p with a score above some threshold. A score function could for example be the weighted Jaccard index, though we explore several others in this paper (Section "Score function"). We provide both a simulation-based and an analytical-based method for setting the score threshold (Section "Choosing a threshold") [1].

Other sketch-based mappers are heuristic: they typically find matching elements between the reference and the read sketches (i.e. anchors) and extend these into maps using chaining [2]. Our algorithm is more resource intensive than these heuristics, as is typical for exact algorithms. However, a problem formulation and an exact algorithm gives several long-term benefits. First, the exact algorithm could be used in place of a greedy heuristic when the input size is not too large. Second, the formulation can spur development of exact algorithms that are optimized for speed and could thus become competitive with heuristics. Third, the formulation could be used to find the most effective score functions, which can then guide the design of better heuristics. Finally, our exact algorithm can return all hits with a score above a threshold, rather than just the best mapping(s). This is important for tasks such as the detection of copy number variation [3] or detecting variation in multi-copy gene families [4].

We evaluate our algorithm (called ESKEMAP), using simulated long reads from the T2T human Y chromosome (Section "Results"). For the same level of precision, the recall of ESKEMAP is 0.88, compared to 0.76 of minimap2. This illustrates the power of ESKEMAP as a method to recover more of the correct hits than a heuristic method. We also compare against Winnowmap2 [5] and edlib [6], which give lower recall but higher precision than ESKEMAP.

Preliminaries

Sequences. Let t be a sequence of elements (e.g. k-mers) that may contain duplicates. We let $|t|$ denote the length of the sequence, and we let $t[i]$ refer to the i-th element in t, with $t[0]$ being the first element. For $0 \le i \le j < |t|$, let $t[i, j]$ represent the subsequence $(t[i], t[i+1], \ldots, t[j])$. The set of elements in t is denoted by $\bar{t}$, e.g. if $t = (\texttt{ACG}, \texttt{TTT}, \texttt{ACG})$ then $\bar{t} = \{\texttt{ACG}, \texttt{TTT}\}$. We let $\mathrm{occ}(x, t)$ represent the number of occurrences of an element x in t, e.g. $\mathrm{occ}(\texttt{ACG}, t) = 2$.

[1] Our algorithm runs in time $\mathcal{O}(|t| + |p| + \ell^2)$ and space $\mathcal{O}(\ell \log \ell)$, where ℓ is the number of times that k-mers from p occur in t

Sketch. Let T be a string and let t be the sequence of k-mers appearing in T. Note that t is a sequence of DNA sequences. For example, if $T = \texttt{ACGAC}$ and $k = 2$, then $t = (\texttt{AC}, \texttt{CG}, \texttt{GA}, \texttt{AC})$. For the purposes of this paper, a *sketch* of T is simply a subsequence of t, e.g. $(\texttt{AC}, \texttt{GA})$. This type of sketch could for example be a minimizer sketch [7, 8], a syncmer sketch [9], or a FracMinHash sketch [10, 11].

Scoring Scheme. A *scoring scheme* $(\mathrm{sc}, \mathrm{thr})$ is a pair of functions: the score function and the threshold function. The *score function* sc is a function that takes as input a pair of non-empty sketches and outputs a real number, intuitively representing the degree of similarity. We assume it is symmetric, i.e. $\mathrm{sc}(p, s) = \mathrm{sc}(s, p)$ for all sketches p and s. If the score function has a parameter, then we write $\mathrm{sc}(s, p; \theta)$, where θ is a vector of parameter values. The *threshold function* thr takes the length of a sketch and returns a score cutoff threshold, i.e. scores below this threshold are not considered similar. Note that the scoring scheme is not allowed to depend on the underlying nucleotide sequences besides what is captured in the sketch.

Miscellenous. We use U_k to denote the universe of all k-mers. Given two sequences p and s, the *weighted Jaccard* is defined as $\frac{\sum_{x \in U_k} \min(\mathrm{occ}(x,p), \mathrm{occ}(x,s))}{\sum_{x \in U_k} \max(\mathrm{occ}(x,p), \mathrm{occ}(x,s))}$. It is 0 when s and p do not share any elements, 1 when s is a permutation of p, and strictly between 0 and 1 otherwise. The weighted Jaccard is a natural extension of Jaccard similarity that accounts for multi-occurring elements.

Problem definition

In this section, we first motivate and then define the Sketch Read Mapping Problem. Fix a scoring scheme $(\mathrm{sc}, \mathrm{thr})$. Let p and t be two sketches, which we refer to as the pattern and the text, respectively. Define a *candidate mapping* as a subinterval $t[a, b]$ of t. A naive problem definition would ask to return all candidate mappings with $\mathrm{sc}(p, t[a, b]) \ge \mathrm{thr}(|p|)$.[2] However, a lower-scoring candidate mapping could contain a higher-scoring candidate mapping as a subinterval, with both scores above the threshold. This may arise due to a large candidate mapping containing a more conserved small candidate mapping, in which case both candidate mappings are of interest. But it may also arise spuriously, as a candidate mapping with a score sufficiently higher than $\mathrm{thr}(|p|)$ can

[2] Notice that in this framing, the threshold is not a single parameter but can vary depending on the read's (sequence or sketch) length. This gives flexibility to the scoring function, since the scores of candidate mappings of reads of different lengths do not need to be comparable to each other. Moreover, computing the threshold value is not a challenge since it needs to be computed just once for each read.

$p = (\texttt{AC}, \texttt{CG}, \texttt{GA}, \texttt{AC}) \quad t = (\texttt{AC}, \texttt{AC}, \texttt{CA}, \texttt{TA}, \texttt{CA}, \texttt{GA}, \texttt{CG}, \texttt{AC}, \texttt{GG}) \quad thr(|p|) = 1$

$a \backslash b$	0	1	2	3	4	5	6	7	8
0	−2	0	−1	−2	−3	−1	**1**	0	−1
1		−2	−3	−4	−5	−3	−1	**1**	0
2			−5	−6	−7	−5	−3	−1	−2
3				−5	−6	−4	−2	0	−1
4					−5	−3	−1	1	0
5						−2	0	**2**	1
6							−2	0	−1
7								−2	−3
8									−5

Fig. 1 An example of the Sketch Read Mapping Problem. We show all candidate mappings $t[a, b]$ for a given pattern p and a text t. Each candidate mapping is represented by its score calculated using $sc_\ell(p, t[a, b]; 1)$ (see Section "Score Function"). Reasonable candidate mappings are shown in black (rather than gray) and final mappings are further bolded

be extended with non-shared k-mers that decrease the score but not below the threshold.

To eliminate most of these spurious cases, we say that a candidate mapping $t[a, b]$ is *left-reasonable* if and only if $\text{occ}(t[a], t[a,b]) \le \text{occ}(t[a], p)$. Similarly, a candidate mapping $t[a, b]$ is *right-reasonable* if and only if $\text{occ}(t[b], t[a,b]) \le \text{occ}(t[b], p)$. A candidate mapping is *reasonable* if it is both left- and right-reasonable. In other words, a reasonable candidate mapping must start and end with a k-mer that has a match in the pattern. We also naturally do not wish to report a candidate mapping that is a subinterval of a longer candidate mapping with a larger score. Formally, we call a candidate mapping $t[a, b]$ *maximal* if there does not exist a candidate mapping $t[a', b']$, with $a' \le a \le b \le b'$ and $\text{sc}(t[a', b'], p) > \text{sc}(t[a, b], p)$. We can now formally define $t[a, b]$ to be a *final mapping* if it is both maximal and reasonable and $\text{sc}(t[a,b], p) \ge \text{thr}(|p|)$. The *Sketch Read Mapping Problem* is then to report all final mappings. An example is given in Fig. 1. We now restate the problem in a succinct manner:

Definition 1 (Sketch Read Mapping Problem). Given a pattern sketch p, a text sketch t, a score function sc, and a threshold function thr, the Sketch Read Mapping Problem is to find all $0 \le a \le b < |t|$ such that

- $\text{sc}(p, t[a,b]) \ge \text{thr}(|p|)$,
- $\text{occ}(t[a], t[a,b]) \le \text{occ}(t[a], p)$,
- $\text{occ}(t[b], t[a,b]) \le \text{occ}(t[b], p)$,
- there does not exist $a' \le a \le b \le b'$ such that $\text{sc}(t[a', b'], p) > \text{sc}(t[a, b], p)$, i.e. $t[a, b]$ is maximal.

Score function

In this section, we explore the design space of score functions and fix two score functions for the rest of the paper. Let p be the sketch of the pattern and let s be a continuous subsequence of the sketch of the text t, i.e. $s = t[a, b]$ for some $a \le b$. For example if $p = (\texttt{ACT}, \texttt{GTA}, \texttt{TAC})$ and $t = (\texttt{AAC}, \texttt{ACT}, \texttt{CCT}, \texttt{GTA})$, we could have $s = t[1, 3] = (\texttt{ACT}, \texttt{CCT}, \texttt{GTA})$. In the context of the Sketch Read Mapping Problem, p is fixed and s varies. Therefore, while the score function is symmetric, the threshold function sets the score threshold as a function of $|p|$. Since p is fixed, the threshold is a single number in the context of a single problem instance.

In the following, we exclusively consider score functions that calculate the similarity of s and p by ignoring the order of k-mers inside the sketches. Taking k-mer order into account would likely make it more complex to compute scores, while not necessarily giving better results on real data. However, score functions that do take order into account are possible and could provide better accuracy in some cases.

A good score function should reflect the number of k-mers shared between s and p. For a given k-mer x, we define

$$x_{\min} := \min(\text{occ}(x, p), \text{occ}(x, s))$$
$$x_{\max} := \max(\text{occ}(x, p), \text{occ}(x, s))$$
$$x_{\text{diff}} := x_{\max} - x_{\min}$$

Intuitively, x occurs a certain number of times in p and a certain number of times in s; we let $x_{\min}$ be the smaller of these two numbers and $x_{\max}$ be the larger of these two numbers. Similarly, x_{diff} is the absolute difference between how often x occurs in p and s. We say that the number of *shared* occurrences is $2x_{\text{min}}$ and the number of *non-shared* occurrences is x_{diff}. These quantities are governed by the relationships

$$|s| + |p| = \sum_{x \in U_k} \text{occ}(x,p) + \text{occ}(x,s) = \sum_{x \in U_k} x_{\text{min}} + x_{\text{max}} = \sum_{x \in U_k} 2x_{\text{min}} + x_{\text{diff}}. \tag{1}$$

A good score function should be (1) increasing with respect to the number of shared occurrences and (2) decreasing with respect to the number of non-shared occurrences. There are many candidate score functions within this space. The first score function we consider is the weighted Jaccard. Formally,

$$\text{sc}_{\text{j}}(s,p) := \frac{\sum_{x\in U_k} x_{\text{min}}}{\sum_{x\in U_k} x_{\text{max}}} = \frac{\sum_x x_{\text{min}}}{|s| + |p| - \sum_x x_{\text{min}}} = \frac{\sum_x x_{\text{min}}}{\sum_x (x_{\text{min}} + x_{\text{diff}})} \tag{2}$$

The above formula includes first the definition but then two algebraically equivalent versions of it, derived using Eq. 1. The weighted Jaccard has the two desired properties of a score function and is a well-known similarity score. However, it has two limitations. First, the use of a ratio makes it challenging to analyze probabilistically, as is the case with the non-weighted Jaccard [12]. Second, it does not offer a tuning parameter which would control the relative benefit of a shared occurrence to the cost of a non-shared occurrence. We therefore consider another score function, parameterized by a real-valued tuning parameter $w > 0$:

$$\text{sc}_\ell(s,p;w) := \sum_{x \in U_k} x_{\text{min}} - w x_{\text{diff}}.$$

It is sometimes more useful to use an equivalent formulation, obtained using Eq. 1:

$$\text{sc}_\ell(s,s';w) = \sum_{x \in U_k} (1+2w)x_{\text{min}} - w(|s| + |s'|). \tag{3}$$

As with the weighted Jaccard, sc_ℓ has the two desired properties of a score function. But, unlike the weighted Jaccard, it is linear and contains a tuning parameter w.

To understand how score functions relate to each other, we introduce the notion of domination and equivalence. Informally, a score function sc_1 dominates another score function sc_2 when sc_1 can always recover the separation between good and bad scores that sc_2 can. In this case, the solution obtained using sc_2 can always be obtained by using sc_1 instead. Formally, let sc_1 and sc_2 be two score functions, parameterized by θ_1 and θ_2, respectively. We say that sc_1 *dominates* sc_2 if and only if for any parameterization θ_2, threshold function thr_2, and pattern sketch p there exist a θ_1 and thr_1 such that, for all sequences s, we have that $\text{sc}_2(s,p;\theta_2) \geq \text{thr}_2(|p|)$ if and only if $\text{sc}_1(s,p;\theta_1) \geq \text{thr}_1(|p|)$. Furthermore, sc_1 dominates sc_2 *strictly* if and only if the opposite does not hold, i.e. sc_2 does not dominate sc_1. Otherwise, sc_1 and sc_2 are said to be *equivalent*, i.e. if and only if each one dominates the other.

We can now precisely state the relationship between sc_ℓ and sc_{j}, i.e. that sc_ℓ strictly dominates sc_{j}. In other words, any solution to the Sketch Read Mapping Problem that is obtained by sc_{j} can also be obtained by sc_ℓ, but not vice-versa. Formally,

Theorem 1 sc_ℓ *strictly dominates the weighted Jaccard score function* sc_{j}.

Proof We start by proving that sc_ℓ dominates sc_{j}. Let p be a pattern sketch and let thr_j be the threshold function associated with sc_{j}. We will use the shorthand $t = \text{thr}_j(|p|)$. First, consider the case that $t < 1$. Let $w = \frac{t}{1-t}$ and let thr_ℓ evaluate to zero for all inputs. Let s be any sketch. The following is a series of equivalent transformations that proves domination.

$$\begin{aligned}
\text{sc}_{\text{j}}(s,p) &\geq t \\
\frac{\sum_x x_{\text{min}}}{\sum_x x_{\text{min}} + x_{\text{diff}}} &\geq t \\
\sum_x x_{\text{min}} &\geq \sum_x t x_{\text{min}} + t x_{\text{diff}} \\
\sum_x (1-t)x_{\text{min}} - t x_{\text{diff}} &\geq 0 \\
\sum_x x_{\text{min}} - \frac{t}{1-t} x_{\text{diff}} &\geq 0 \\
\text{sc}_\ell(s,p;w) &\geq \text{thr}_\ell(|p|)
\end{aligned}$$

Next, consider the case $t > 1$. In this case, for all s, $\text{sc}_{\text{j}}(s,p) < t$, since the weighted Jaccard can never exceed one. Observe that $\text{sc}_\ell(s,p;w) \leq |p|$ for any non-negative w. Therefore, we can set $\text{thr}_\ell(|p|) = |p| + 1$ and let w be any non-negative number, guaranteeing that for all s, $\text{sc}_\ell(s,p;w) < \text{thr}_\ell(|p|)$.

Finally consider the case that $t = 1$. Then, $\text{sc}_{\text{j}}(s,p) \geq t$ if and only if s and p are permutations of each other, i.e. $x_{\text{diff}} = 0$ for all x. Setting $\text{thr}_\ell(|p|) = |p|$ and letting w be any strictly positive number guarantees that $\mathrm{sc}_\ell(s,p;\,w) \geq \mathrm{thr}_\ell(|p|)$ if and only if s and p are permutations of each other.

To prove that sc_ℓ is not dominated by sc_j, we fix $w = 1$ (though any value could be used) and give a counterexample family to show that sc_j cannot recover the separation that sc_ℓ can. Pick an integer $i \geq 1$ to control the size of the counterexample. Let p be a pattern sketch of length $4i$ consisting of arbitrary k-mers. We construct two sketches, s_1 and s_2. The sequence s_1 is an arbitrary subsequence of p of length i. Observe that $\sum_{x\in\overline{p}\cup\overline{s_1}} x_{\min} = \sum_x \mathrm{occ}(x,s_1) = i$. The sequence s_2 is p appended with arbitrary k-mers to get a length $12i$. Observe that $\sum_{x\in\overline{p}\cup\overline{s_2}} x_{\min} = \sum_x \mathrm{occ}(x,p) = 4i$. Using Eq. 3 for sc_ℓ and Eq. 2 for sc_j,

$$\mathrm{sc}_\ell(s_1,p) = -2i \qquad \mathrm{sc}_\mathrm{j}(s_1,p) = 1/4$$
$$\mathrm{sc}_\ell(s_2,p) = -4i \qquad \mathrm{sc}_\mathrm{j}(s_2,p) = 1/3$$

Under sc_ℓ, s_1 has a higher score, while under sc_j, s_2 has a higher score. If thr_ℓ is set to accept s_1 but not s_2 (e.g. $\mathrm{thr}_\ell = -3i$), then it is impossible to set thr_j to achieve the same effect. In other words, since $\mathrm{sc}_\mathrm{j}(s_2) > \mathrm{sc}_\mathrm{j}(s_1)$, any threshold that accepts s_1 must also accept s_2. □

Next, we show that many other natural score functions are equivalent to sc_ℓ. Consider the following score functions:

$$\mathrm{sc}_\mathrm{A}(s,p;\,a_1) := \sum_{x\in U_k}(a_1x_{\min} - x_{\mathrm{diff}}) \qquad \text{with } a_1 > 0$$
$$\mathrm{sc}_\mathrm{B}(s,p;\,b_1,b_2) := \sum_{x\in U_k}(b_1x_{\min} - b_2x_{\mathrm{diff}}) \qquad \text{with } b_1 > 0 \text{ and } b_2 > 0$$
$$\mathrm{sc}_\mathrm{C}(s,p;\,c_1,c_2) := \sum_{x\in U_k}(c_1x_{\min} - c_2x_{\max}) \qquad \text{with } c_1 > c_2 > 0$$
$$\mathrm{sc}_\mathrm{D}(s,p;\,d_1,d_2) := \sum_{x\in U_k}(d_1x_{\min}) - d_2|s| \qquad \text{with } d_1 > 2d_2 \text{ and } d_2 > 0$$

The conditions on the parameters are there to enforce the two desired properties of a score function. Each of these score functions is natural in its own way, e.g. sc_A is similar to sc_ℓ but places the weight on $x_{\min}$ rather than on x_{diff}. One could also have two separate weights, as in the score sc_B. One could then replace x_{diff} with $x_{\max}$, as in sc_C, which is the straightforward reformulation of the weighted Jaccard score as a difference instead of a ratio. Or one could replace x_{diff} with the length of s, as in sc_D. The following theorem shows that the versions turn out to be equivalent to sc_ℓ and to each other. The proof is a straightforward algebraic manipulation and is left for the appendix.

Theorem 2 *The score functions* sc_ℓ, sc_A, sc_B, sc_C, *and* sc_D *are pairwise equivalent.*

Choosing a threshold

In this section, we propose two ways to set the score threshold. The first is analytical (Section "Analytical analysis") and the second is with simulations (Section "Simulation-based analysis"). The analytical approach gives a closed form formula for the expected value of the score under a mutation model. However, it only applies to the FracMinHash sketch, assumes a read has little internal homology, and does not give a confidence interval. The simulation approach can apply to any sketch but does not offer any analytical insight into the behavior of the score. The choice of approach ultimately depends on the use case.

We first need to define a generative mutation model to capture both the sequencing and evolutionary divergence process:

Definition 2 (Mutation model). Let S be a circular string[3] with n characters. The mutation model produces a new string S' by first setting $S' = S$ and then taking the following steps:

1. For every $0 \leq i < n$, draw an action $a_i \in \{sub, del, unchanged\}$ with probability of p_{sub} for sub, p_{del} for del, and $1 - p_{\mathrm{sub}} - p_{\mathrm{del}}$ for unchanged. Also, draw an insertion length b_i from a geometric distribution with mean p_{ins}.[4]
2. Let *track* be a function mapping from a position in S to its corresponding position in S'. Initially,

[3] We assume the string is circular to avoid edge cases in the analysis but, for long enough strings, this assumption is unlikely to affect the accuracy of the results.

[4] Here, a geometric distribution is the number of failures before the first success of a Bernoulli trial. This geometric distribution has parameter $\frac{1}{p_{\mathrm{ins}}+1}$.

$track(i) = i$, but as we delete and add characters to S', we assume that *track* is updated to keep track of the position of $S[i]$ in S'.

3. For every i such that $a_i = sub$, replace $S'[i]$ with one of the three nucleotides not equal to $S[i]$, chosen uniformly at random.
4. For every $0 \le i < n$, insert b_i nucleotides (chosen uniformly at random) before $S'[track(i)]$.
5. For every i such that $a_i = del$, remove $S'[track(i)]$ from S'.

Analytical analysis

To derive an expected score under the mutation model, we need to specify a sketch. We will use the FracMinHash sketch [10], due to its simplicity of analysis [11].

Definition 3 (FracMinHash). Let h be a hash function that maps a k-mer to a real number between 0 and 1, inclusive. Let $0 < q \le 1$ be a real-valued number called the sampling rate. Let S be a string. Then the *FracMinHash* sketch of S, denoted by s, is the sequence of all k-mers x of S, ordered as they appear in S, such that $h(x) \le q$.

Consider an example with $k = 2$, $S =$ CGGACGGT, and the only k-mers hashing to a value $\le q$ being CG and GG. Then, $s = (\text{CG}, \text{GG}, \text{CG}, \text{GG})$.

We make an assumption, which we refer to as the *mutation-distinctness assumption*, that the mutations on S never create a k-mer that is originally in S. Based on previous work [13], we find this necessary to make the analysis mathematically tractable (for us). The results under this assumption become increasingly inaccurate as the read sequence contains increasingly more internal similarity. For example, reads coming from centromeres might violate this assumption. In such cases, it may be better to choose a threshold using the technique in Section "Simulation-based Analysis".

We can now derive the expected value of the score under the mutation model and FracMinHash.

Theorem 3 *Let S be a circular string and let S' be generated from S under the mutation model with the mutation-distinctness assumption and with parameters p_{sub}, p_{del}, and p_{ins}. Let s and s' be the FracMinHash sketches of S and S', respectively, with sampling rate q. Then, for all real-valued tuning parameters $w > 0$,*

$$E[\text{sc}_\ell(s, s'; w)] = |s| q(\alpha + w(2\alpha - 2 + p_{\text{del}} - p_{\text{ins}})),$$

where $\alpha = \frac{(1-p_{\text{del}}-p_{\text{sub}})^k}{(p_{\text{ins}}+1)^{k-1}}$.

Proof Observe that under mutation-distinctness assumption, the number of occurrences of a k-mer that is in s can only decrease after mutation, and a k-mer that is newly created after mutation has an $x_{\min}$ of 0. Therefore, applying Eq. 3,

$$\text{sc}_\ell(s, s'; w) = \sum_{x \in \bar{s}} (1 + 2w)\text{occ}(x, s') - w(|s| + |s'|)$$

(Recall that $\bar{s}$ is the set of all k-mers in s.) We will first compute the score conditioned on the hash function of the sketch being fixed. Note that when h is fixed, then the sketch s becomes fixed and s' becomes only a function of S'. By linearity of expectation,

$$E[\text{sc}_\ell(s, s'; w) \mid h] = \sum_{x \in \bar{s}} (1 + 2w) E[\text{occ}(x, s') \mid h] - w(|s| + E[|s'| \mid h]) \tag{4}$$

It remains to compute $E[|s'| \mid h]$ and $E[\text{occ}(x, s') \mid h]$. Observe that the number of elements in s' is the number of elements in s minus the number of deletions plus the sum of all the insertion lengths. By linearity of expectation,

$$E[|s'| \mid h] = |s| - p_{\text{del}}|s| + p_{\text{ins}}|s| = |s|(1 - p_{\text{del}} + p_{\text{ins}})$$

Next, consider a k-mer $x \in \bar{s}$ and $E[\text{occ}(x, s')]$. Recall by our mutation model that no new occurrenes of x are introduced during the mutation process. So $\text{occ}(x, s')$ is equal to the number of occurrences of x in S that remain unaffected by mutations. Consider an occurrence of x in s. The probability that it remains is the probability that all actions on the k nucleotides of x were "unchanged" and the length of all insertions in-between the nucleotides was 0. Therefore,

$$E[\text{occ}(x, s') \mid h] = \text{occ}(x, s)(1 - p_{\text{del}} - p_{\text{sub}})^k \left(\frac{1}{p_{\text{ins}} + 1}\right)^{k-1} = \alpha\,\text{occ}(x, s)$$

Putting it all together,

$$\begin{aligned} E[\mathrm{sc}_\ell(s,s';w) \mid h] &= \sum_{x\in\bar{s}}(1+2w)E[\mathrm{occ}(x,s') \mid h] - w(|s| + E[|s'| \mid h]) \\ &= \alpha(1+2w)\sum_{x\in\bar{s}}\mathrm{occ}(x,s) - w(|s| + |s|(1-p_{\mathrm{del}}+p_{\mathrm{ins}})) \\ &= \alpha(1+2w)|s| - w(|s| + |s|(1-p_{\mathrm{del}}+p_{\mathrm{ins}})) \\ &= |s|(\alpha(1+2w) - w(2-p_{\mathrm{del}}+p_{\mathrm{ins}})) \\ &= |s|(\alpha + w(2\alpha - 2 + p_{\mathrm{del}} - p_{\mathrm{ins}})) \end{aligned}$$

To add the sketching step, we know from [11] that the expected size of a sketch is the size of the original text times q. Then,

$$\begin{aligned} E[\mathrm{sc}_\ell(s,s';w)] &= E[E[\mathrm{sc}_\ell(s,s';w) \mid h]] \\ &= E[|s|(\alpha + w(2\alpha - 2 + p_{\mathrm{del}} - p_{\mathrm{ins}}))] \\ &= E[|s|](\alpha + w(2\alpha - 2 + p_{\mathrm{del}} - p_{\mathrm{ins}})) \\ &= |s|q(\alpha + w(2\alpha - 2 + p_{\mathrm{del}} - p_{\mathrm{ins}})) \end{aligned}$$

□

Simulation-based analysis

First, we choose the parameters of the mutation model according to the target sequence divergence between the reads and the reference caused by sequencing errors, but also due to the evolutionary distance between the reference and the organism sequenced. If one is also interested in mapping reads to homologous regions within the reference that are related more distantly, e.g. if there exist multiple copies of a gene, the mutation parameters can be increased further.

To generate a threshold for a given read length, we generate sequence pairs (S, S'), where S is a uniformly random DNA sequence of the given length and S' is mutated from S under the above model. We then calculate the sketch of S and S', which we call s and s', respectively. The sketch can for example be a minimizer sketch, a syncmer sketch, or a FracMinHash sketch. We can then use the desired score function to calculate a score for each pair (s, s'). For a sufficiently large number of pairs, their scores will form an estimate of the underlying score distribution for sequences that evolved according to the used model. It is then possible to choose a threshold such that the desired percentage of mappings would be reported by our algorithm. For example, one could choose a threshold to cover a one sided 95% confidence interval of the score.

In order to be able to adjust thresholds according to the variable length of reads produced from a sequencing run, the whole process may be repeated several times for different lengths of S. Thresholds can then be interpolated dynamically for dataset reads whose lengths were not part of the simulation.

Algorithm for the Sketch Read Mapping Problem

In this section,[5] we describe a dynamic programming algorithm for the Sketch Read Mapping Problem under both the weighted Jaccard and the linear scores (sc_j and sc_ℓ, respectively). Let t be the sketch of the text, let p be the sketch of the pattern, let L be the sequence of positions in t that have a k-mer that is in $\bar{p}$, in increasing order, and let $\ell = |L|$. Our algorithm takes advantage of the fact that p is typically much shorter than t and hence the number of elements of t that are shared with p is much smaller than $|t|$ (i.e. $\ell \ll |t|$). In particular, it suffices to consider only candidate mappings that begin and end in positions listed in L, since by definition, if $t[a, b]$ is a reasonable candidate mapping, then $t[a] \in \bar{p}$ and $t[b] \in \bar{p}$.

We present our algorithm as two parts. In the first part (Section "Computing S"), we compute a matrix S with ℓ rows and ℓ columns so that $S(i,j) = \sum_x \min(\mathrm{occ}(x,p), \mathrm{occ}(x, t[L[i], L[j]])$. S is only defined for $j \geq i$. We also store an index i^* for each column j indicating the smallest index for which $t[L[i^*], L[j]]$ is a right-reasonable candidate mapping. In the second part (Section "Computing maximality"), we scan through S and output the candidate mapping $t[i, j]$ if and only if it is reasonable, maximal and has a score above the threshold.

The reason that $S(i,j)$ is not defined to store the score of the candidate mapping $t[L[i], L[j]]$ is that the score can be computed from $S(i,j)$ in constant time, for both sc_j and sc_ℓ. To see this, let $x_{\min} := \min(\mathrm{occ}(x,p), \mathrm{occ}(x, t[L[i], L[j]])$. Recall that Eq. 2 allows us to express $\mathrm{sc}_\mathrm{j}(t[i,j], p)$ as a function of $\sum x_{\min}$, $|p|$, and the length of the candidate mapping, i.e. $j - i + 1$. Similarly, we can apply Eq. 1 to express sc_ℓ as

[5] This section has been substantially updated in comparison to the conference version of this paper [14]. In particular, Algorithm 1 has been modified so that it computes S in a right-to-left scan, rather than left-to-right. In this way, S does not need to be stored in memory, eliminating the $\Theta(\ell^2)$ space requirement.

$$\begin{aligned}\mathrm{sc}_\ell(t[i,j],p;\,w) &:= \sum_x (x_{\min} - w x_{\mathrm{diff}}) = \left(\sum_x x_{\min}\right) - w(|s| + |p| - \sum_x 2x_{\min})\\ &= (1+2w)\sum_x x_{\min} - w(j-i+1+|p|)\end{aligned}$$

Thus, once $\sum_x x_{\min}$ is computed, either of the scores can be computed trivially.

Computing *S*

We compute S using dynamic programming. We will give a recurrence for S so that it can be filled in from right-to-left. For the base case of the last column, we start by setting $S(\ell-1,\ell-1)=1$. This is trivially correct since $L[\ell-1]\in\overline{p}$ by definition. We then fill in the last column, starting from $i=\ell-2$ and going down to $i=0$. If $\mathrm{occ}(t[L[i]], t[L[i], L[\ell-1]]) \le \mathrm{occ}(t[L[i]], p)$, then we set $S(i,\ell-1)=S(i+1,\ell-1)+1$. Otherwise, we set $S(i,\ell-1)=S(i+1,\ell-1)$. For the remaining cells of S, i.e. for all i and j such that $0\le i\le j<\ell-1$, we use the recursive formula:

$$S(i,j) = S(i,j+1) - \begin{cases} 1 & \text{if } \mathrm{occ}(t[L[j+1]], t[L[i], L[j+1]]) \le \mathrm{occ}(t[L[j+1]], p)\\ 0 & \text{otherwise.}\end{cases} \qquad (5)$$

To see the correctness of this formula, observe that all the elements of $t[L[j]+1, L[j+1]-1]$ are, by definition, not in $\overline{p}$ and hence their minimum occurrence value is 0. If the element $x=t[L[j+1]]$ helps to increase $\min(\mathrm{occ}(x, t[L[i], L[j+1]]), \mathrm{occ}(x,p))$, then excluding it decreases the minimum count by one, otherwise the minimum occurrence does not change. To maintain the smallest value of i for which $S(i,j+1)$ is right-reasonable, it is enough to observe that $S(i,j+1)$ is right-reasonable if and only if the top case is used.

To design an efficient algorithm based on Eq. 5, we need two auxiliary data structures. The first is a hash table P_{cnt} that stores, for every k-mer in $\overline{p}$, how often it occurs in p. A second hash table T_{cnt} is used to store a count for every k-mer in $\overline{p}$. T_{cnt} maintains the invariant that at the start of processing column j, $T_{\mathrm{cnt}}[x] = \mathrm{occ}(x, t[L[0], L[j+1]])$. This invariant can be maintained by 1) initializing $T_{\mathrm{cnt}}[x]$ to $\mathrm{occ}(x, t[L[0], L[\ell-1]])$ for every k-mer $x\in\overline{p}$, and 2) when starting to process column j, decrementing $T_{\mathrm{cnt}}[L[j+1]]$ by 1.

To compute S, the P_{cnt} hash table is constructed via a scan through p and $T_{\mathrm{cnt}}[x]$ is set to 0 for every k-mer $x\in\overline{p}$. Next, we simultaneously compute the last column of S and initialize T_{cnt} so that it holds the counts of all k-mers from $\overline{p}$ in $t[L[0], L[\ell-1]]$, as follows. We trivially initialize $S(\ell-1,\ell-1)$ and then compute $S(i,\ell-1)$ starting from $i=\ell-2$ down to $i=0$. Before computing $S(i,\ell-1)$, we increment the count stored at $T_{\mathrm{cnt}}[t[L[i]]]$ by 1. After this increment, $T_{\mathrm{cnt}}[L[i]] = \mathrm{occ}(L[i], t[L[i], L[\ell-1]])$ and can be used to distinguish between the two cases necessary for computing $S(i,\ell-1)$.

Algorithm 1 details the rest of the algorithm for computing S, column-by-column using Eq. 5. The non-trivial part is to determine which case of Eq. 5 to use in constant-time (i.e. to compute $\mathrm{occ}(t[L[j+1]], t[L[i], L[j+1]])$). We let $c_1 = \mathrm{occ}(x, t[0, L[i-1]])$ and let $c_2 = \mathrm{occ}(x, t[0, L[j+1]])$ and observe that $\mathrm{occ}(x, t[L[i], L[j+1]]) = c_2 - c_1$. Notice that $c_2 = T_{\mathrm{cnt}}[x]$, so it only remains to compute c_1. When computing a column $j<\ell-1$, we are processing all the rows starting from 0 up to $\ell-1$. Thus we can update c_1 by initially setting $c_1=0$ (Line 8) and then, for each new row i, incrementing c_1 if $t[L[i]]=x$ (Line 18).

Algorithm 1 Part 1 of ESKEMAP algorithm

```
Input: two sketches $t$ and $p$.
Output: the matrix $S$ and an integer $i^*$ for each column of $S$.
1:  Construct $L$ array.
2:  Construct $P_{\text{cnt}}$.
3:  Simultaneously fill in $S(i, \ell - 1)$, for all $0 \leq i < \ell$, and initialize $T_{\text{cnt}}$.
4:  for $j = \ell - 2$ down to 0 do
5:      $i^* \leftarrow 0$
6:      $x \leftarrow t[L[j + 1]]$
7:      $c_2 \leftarrow T_{\text{cnt}}[x]$                    ▷ This holds $\text{occ}(x, t[0, L[j + 1]])$
8:      $c_1 \leftarrow 0$                                   ▷ This will hold $\text{occ}(x, t[0, L[i - 1]])$
9:      $T_{\text{cnt}}[x] \leftarrow T_{\text{cnt}}[x] - 1$
10:     for $i = 0$ to $j$ do
11:         if $c_2 - c_1 \leq P_{\text{cnt}}[x]$ then
12:             $S(i, j) \leftarrow S(i, j + 1) - 1$
13:             Save $i^* \leftarrow \min(i^*, i)$ for column $j + 1$     ▷ Update $i^*$ for column $j + 1$
14:         else
15:             $S(i, j) \leftarrow S(i, j + 1)$
16:         end if
17:         if $t[L[i]] = x$ then
18:             $c_1 \leftarrow c_1 + 1$
19:         end if
20:     end for
21: end for
```

Computing maximality

In the second step, we identify which of the candidate mappings in S are final. Our algorithm is shown in Algorithm 2. We traverse S column-by-column starting with the last column and then row-by-row, starting from the first row. While traversing S, we maintain a list M of all maximal, reasonable candidate mappings above the threshold found so far and their scores. M has the invariant that the candidate mappings are increasingly ordered by their start positions.

To maintain the invariant that M is sorted by start position, we maintain a pointer *cur* to a location in M (Lines 7-11). At the start of a new column traversal, when the row $i = 0$, *cur* points to the start of M. As the row is increased, we move *cur* forward until it hits the first value in M with a start larger than i. When a new final mapping is added to M, we do so at *cur*, which guarantees the order invariant of M (Lines 16-20).

Due to the order cells in S are processed during our traversal, a candidate mapping $t[L[i], L[j]]$ is maximal if and only if its score is larger than the score of all other final mappings in M with position $i' \leq i$. For a given column, since we are processing the candidate mappings in increasing order of i, we can simultenously maintain a running variable *maxSoFar* that holds the maximum value in M up to *cur* (Line 8). We can then determine if a candidate mapping is maximal by simply checking its score against *maxSoFar* (Line 14). Note that as long as we have not yet seen any final mapping at some position $i' \leq i$, a candidate mapping is already maximal if its score equals $\text{thr}(|p|)$. This is ensured via a flag *supMpFnd* and an additional satisfiable subclause (Line 14). As soon as *maxSoFar* is updated, *supMpFnd* is set (Line 10).

To check for the reasonability of a candidate mapping corresponding to $S(i, j)$, we need to verify that $i \geq i^*$ and $S(i,j) = S(i+1,j) + 1$, for $0 \leq i < j$. To see that this is correct, first observe that i^* is the smallest index of a cell in column j of S belonging to a right-reasonable candidate mapping that could be found during the execution of Algorithm 1. Thus, there is no candidate mapping belonging to a cell $S(i, j)$ with $i < i^*$. Vice versa, every candidate mapping $t[L[i], L[j]]$ with $i > i^*$ has to be right-reasonable, because it is a substring of $t[L[i^*], L[j]]$ and $\text{occ}(t[L[j]], t[L[i], L[j]]) \leq \text{occ}(t[L[j]], t[L[i^*], L[j]])$ must hold. The requirement of the second condition $(S(i,j) = S(i+1,j) + 1,\ \text{for}\ 0 \leq i < j)$ to ensure left-reasonability can be justified by a similar argument as given for the correctness of Eq. 5. All candidate mappings of cells $S(i, j)$ where $i = j$ are always reasonable, because they contain only 1 k-mer $x \in \overline{p}$.

Algorithm 2 Part 2 of ESKEMAP algorithm

Input: two sketches t and p, the matrix S computed by Algorithm 1 with an integer i^* for each column, a score function, and a threshold function thr.
Output: all final mappings that are reasonable, maximal, and have a score of at least $\text{thr}(|p|)$.

```
Let $M$ be an empty linked list with $(i, j, s)$ tuples.
for $j = \ell - 1$ down to 0 do
    $maxSoFar \leftarrow \text{thr}(|p|)$
    $cur \leftarrow M.start$
    $supMpFnd \leftarrow$ false
    for $i = 0$ to $j$ do
        while $cur \neq M.end$ and $cur.i \leq i$ do
            $maxSoFar = \max(maxSoFar, cur.s)$
            cur++
            $supMpFnd \leftarrow$ true
        end while
        if $S(i, j)$ is reasonable then
            $s \leftarrow$ score of $S(i, j)$
            if $s > maxSoFar$ or $(\neg supMpFnd \wedge s = \text{thr}(|p|))$ then
                Output $t[L[i], L[j]]$
                if $cur \neq M.start$ then
                    $M.insertBefore(cur, (i, j, s))$
                else
                    $M.insertAt(cur, (i, j, s))$
                end if
            end if
        end if
    end for
end for
```

Runtime and memory analysis

The runtime for Algorithm 1 is $\Theta(\ell^2 + |t| + |p|)$. Note that this time bound refers to an expected time if assuming a hash table with constant time for insertion and lookup. The P_{cnt} table can be constructed in a straightforward manner in $\mathcal{O}(|p|)$ time, the L array is constructed in $\mathcal{O}(|t|)$. Algorithm 2 runs two for loops with constant time internal operations, with the exception of the while loop to fast forward the *cur* pointer. The total time for the loop is amortized to $O(\ell)$ for each column. Therefore, the total time for Algorithm 2 is $\Theta(\ell^2)$. This gives the total running time for our algorithm as $\mathcal{O}(|t| + |p| + \ell^2)$.

The total space used by the algorithm is the sum of the space used by P_{cnt}, T_{cnt}, L, S and M. The P_{cnt} table stores $|p|$ integers with values up to $|p|$. However, notice that when $|p| > \ell$, we can limit the table to only store k-mers that are in $\bar{t}$, i.e. only ℓ k-mers. We can also replace integer values greater than ℓ with ℓ, as it would not affect the algorithm. Therefore, the P_{cnt} table uses $\mathcal{O}(\ell \log \ell)$ space. The T_{cnt} table stores at most ℓ entries with values at most ℓ and therefore takes $\mathcal{O}(\ell \log \ell)$ space. To store S completely would require $\Theta(\ell^2)$ space. However, as S is filled column by column and since the recursion in Eq. 5 only depends on the previously computed column, we can dynamically compute S by only storing two columns of S in memory. Further, we can apply Algorithm 2 separately on each column after it has been computed by Algorithm 1 as long as M is retained throughout the whole processing of S. Hence, we do not need to spend more than $\Theta(\ell)$ space for storing S. To limit the space required for storing M to $\mathcal{O}(\ell \log \ell)$, we can observe that, in fact, it is not necessary to keep all final mappings stored in M to check for maximality during the execution of our algorithm. Instead, it is enough to retain the last candidate mapping found for each start position. As soon as we find a final candidate mapping starting at the same position as a previously found one, we can output the latter and overwrite its entry in M with the newly found candidate mapping. Thus our algorithm uses a total of $\mathcal{O}(\ell \log \ell)$ space.

Results

We implemented the ESKEMAP algorithm described in "Algorithm for the Sketch Read Mapping Problem" section using sc_ℓ as score function and compared it to other methods in a read mapping scenario. For better comparability, we implemented it with the exact same minimizer sketching approach as used by minimap2. Source code of our implementation as well as a detailed documentation of our comparison described below including exact program calls is available from https://github.com/medvedevgroup/eskemap.

Datasets

For our evaluation, we used the T2T reference assembly of human chromosome Y (T2T-CHM13v2.0; [GenBank:NC_060948.1]) [15]. The chromosome contains many ampliconic regions with duplicated genes from several gene families. Identifying a single best hit for reads from such regions is not helpful and instead it is necessary to find all good mappings [16]. Such a reference poses a challenge to heuristic algorithms and presents an opportunity for an all-hits mapper like ESKEMAP to be worth the added compute.

We simulated a read dataset on this assembly imitating characteristics of a PacBio Hifi sequencing run [17]. For each read, we randomly determined its length r according to a gamma distribution with a 9000bp mean and a standard deviation of 7000bp. Afterwards, a random integer $i \in [1, n - r + 1]$ was drawn as the read's start position, where n refers to the length of the chromosome. Sequencing errors were simulated by introducing mutations into each read's sequence using the mutation model described in Definition 2 and a total mutation rate of 0.2% distributed with a ratio of 6:50:54 between substitution/insertion/deletion, as suggested in [18]. Aiming for a sequencing depth of 10x, we simulated 69401 reads.

The T2T assembly of the human chromosome Y contains long centromeric and telomeric regions which consist of short tandem and higher order repeats. Mapping reads in such regions results in thousands of hits that are meaningless for many downstream analyses and significantly increases the runtime of mapping. Therefore, we excluded all reads from the initially simulated set which could be aligned to more than 20 different, non-overlapping positions using edlib (see below). After filtering, a set of 32295 reads remained.

Tools

We compared ESKEMAP to two other sketch-based approaches and an exact alignment approach. The sketch-based approaches were minimap2 (version 2.24-r1122) and Winnowmap2 (version 2.03), run using default parameters. In order to be able to compare our results also to an exact, alignment-based mapping approach, we used the C/C++ library of Edlib [6] (version 1.2.7) to implement a small script that finds all non-overlapping substrings of the reference sequence a read could be aligned to with an edit distance of at most T. We tried values $T \in \{0.01r, 0.02r, 0.03r\}$, where recall that r is the read length. We refer to this script as simply *edlib*.

For ESKEMAP, we aimed to make the results as comparable as possible to minimap2. We therefore used a minimizer sketch with the same k-mer and window size as minimap2 ($k = 15, w = 10$). However, we excluded minimizers that occurred > 100 times inside the reference sketch, to limit the $\mathcal{O}(\ell^2)$ memory use of ESKEMAP, even as this exclusion may potentially hurt ESKEMAP 's accuracy. We used the default $w = 1$ as the tuning parameter in the linear score. To set the score threshold, we used the dynamic procedure described in Section "Simulation-based analysis". In particular, we used five different sequence lengths for simulations and used a divergence of 1%. We used the same sequencing error profile as for read simulation. Four thresholds were then chosen so at to cover the one-sided confidence interval of 70%, 80%, 90%, and 95%, respectively.

Accuracy measure

We compared the reference substrings corresponding to each reported mapping location of any tool to the mapped read's sequence using BLAST [19]. If a pairwise comparison of both sequences resulted either in a single BLAST hit with an E-value not exceeding 0.01[6] and covering at least 90% of the substring or the read sequence or if a set of non-overlapping BLAST hits was found of which none had an E-value above 0.01 and their lengths summed up to at least 90% of either the reference substring's or the read sequence's length, we considered the reference mapping location as homologous.

For each read, we combine all the homologous reference substrings found across all tools into a ground truth set for that read. We then measure the accuracy of a mapping as follows. We determined for each k-mer of the reference sequence's sketch whether it is either a *true positive* (TP), *false positive* (FP), *true negative* (TN) or *false negative* (FN). A k-mer was considered a TP if it was covered by both a mapping and a ground truth substring. It was considered a FP if it was covered by a mapping, but not by any ground truth substring. Conversely, it was considered a TN if it was covered by neither a mapping nor a ground truth substring and considered a FN if it was covered by a substring of the ground truth exclusively. The determination was performed for each read independently and results were accumulated per tool to calculate precision and recall measures.

[6] In order to ensure robustness of results, BLAST runs were also repeated for E-value thresholds of 0.005 and 0.001 causing only neglectable differences for subsequent analyses.

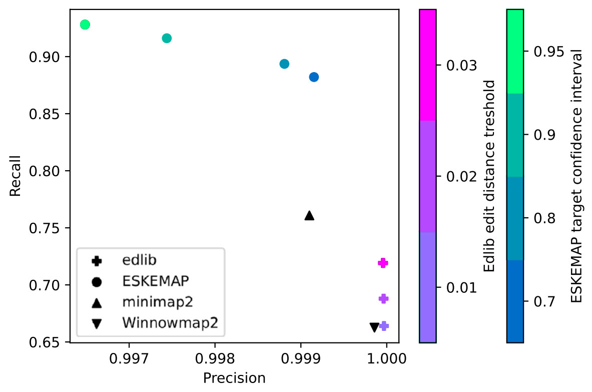

Fig. 2 Mapping accuracies of all tools. For edlib, the color of the cross encodes the various edit distance thresholds (0.01, 0.02, 0.03). For ESKEMAP, the color of the circles indicate the score threshold used, in terms of the target confidence interval used (0.7, 0.8, 0.9, 0.95). The ground truth is determined by combining the mappings from all tools and filtering out those with bad BLAST scores. The most lenient thresholds for edlib and ESKEMAP were used

Accuracy results

The precision and recall of the various tools is shown in Fig. 2. The most controlled comparison can be made with respect to minimap2, since the sketch used by ESKEMAP is a subset of the one used by minimap2. At a score threshold corresponding to 70% recovery, ESKEMAP achieves the same precision (0.999) as minimap2. However, the recall of ESKEMAP is 0.88, compared to 0.76 of minimap2. We see that the extra effort of minimap2 to take k-mer order inside the sketches into account does not seem to pay off in this experiment compared to the approach of ESKEMAP which avoids this. The increased recall of ESKEMAP can be explained by the fact that ESKEMAP is able to find all promising but slightly worse suboptimal mapping positions for a read originating, e.g., from the sequence of a duplicated gene on the chromosome. Many of these are not reported by minimap2 as it only reports a fixed number of suboptimal mappings per read. This illustrates the potential of ESKEMAP as a method to recover more of the correct hits than a heuristic method. More generally, ESKEMAP achieves a recall around 90%, while all other tools have a recall of at most 76%. However, both edlib and Winnowmap2 achieve a slightly higher precision (by 0.001).

Time and memory results

We compared the runtimes and memory usage of all sketch-based methods (Table 1). Calculations were performed on a virtual machine with 28 cores and 256 GB of RAM. We did not include edlib in this alignment since, as an exact alignment-based method, it took much longer to complete (i.e. running highly parallelized for many days on a system with many cores). We see that both heuristics are significantly faster than our exact algorithm. However, they also find many fewer mapping positions per read. E.g., only one mapping position is reported for 67% and 75% of all reads by minimap2 and Winnowmap2, respectively. In comparison, ESKEMAP finds more than one mapping position for almost every second read (49%). When the runtime is normalized per output mapping, ESKEMAP is actually more than an order of magnitude faster than the other tools.

ESKEMAP 's memory usage was also found to be smaller than that of minimap2 and Winnowmap2 in our experiment.[7]

Table 1 Runtime and memory usage comparison of all sketch-based methods

Tool	User time [s]		Memory [GB]
	Total	Per mapping	
ESKEMAP	133,621	0.02	2
minimap2	26,232	0.55	4.5
Winnowmap2	9207	0.19	7

The tools were called to map 32295 simulated PacBio Hifi sequencing reads on the T2T assembly of human chromosome Y. Runtimes are shown both as total values and normalized by the number of reported mapping positions

Conclusion

In this work, we formally defined the Sketch Read Mapping Problem, i.e. to find all positions inside a reference sketch with a certain minimum similarity to a read sketch under a given similarity score function. We also proposed an exact dynamic programming algorithm called ESKEMAP to solve the problem, running in $\mathcal{O}(|t| + |p| + \ell^2)$ time and $\Theta(\ell^2)$ space. We evaluated ESKEMAP 's performance by mapping a simulated long read dataset to the T2T assembly of human chromosome Y and found it to have a superior recall for a similar level of precision compared to minimap2, while offering precision/recall tradeoffs compared with edlib or Winnowmap2.

In order to further improve on ESKEMAP 's runtime, a strategy could be to develop filters that prune the result's search space. This could be established, e.g., by terminating score calculations for a column once it is clear an optimal solution would not make use of the rest of that column. Our prototype implementation of ESKEMAP would also benefit from additional

[7] Note that ESKEMAP 's memory usage is significantly reduced relative to the conference version of the paper.

engineering of the code base, potentially leading to substantial improvements of runtime and memory in practice.

Having an exact sketch-based mapping algorithm at hand also opens the door for the exploration of novel score functions to determine sequence similarity on the level of sketches. Using our algorithm, combinations of different sketching approaches and score functions may be easily tested. Eventually, this may lead to a better understanding of which sketching methods and similarity measures are most efficient considering sequences with certain properties like high repetitiveness or evolutionary distance.

Appendix A: Proofs

Proof of Theorem 2 Observe that domination is a transitive property, i.e. if sc_1 dominates sc_2 and sc_2 dominates sc_3, then sc_1 dominates sc_3. To prove equivalence, we will prove the following circular chain of domination: $\mathrm{sc}_\ell \leftarrow \mathrm{sc}_\mathrm{B} \leftarrow \mathrm{sc}_\mathrm{C} \leftarrow \mathrm{sc}_\mathrm{D} \leftarrow \mathrm{sc}_\mathrm{A} \leftarrow \mathrm{sc}_\ell$.

First, observe that sc_B trivially dominates sc_ℓ by keeping the threshold function the same and setting $b_1 = 1$ and $b_2 = w$.

Next, we prove that sc_C dominates sc_B. Let p be a pattern and let $t = \mathrm{thr}_\mathrm{B}(|p|)$. Set $\mathrm{thr}_\mathrm{C} = \mathrm{thr}_\mathrm{B}$ and $c_1 = b_1 + b_2$ and $c_2 = b_2$. Then, for all s, the following series of equivalent transformations proves that sc_C dominates sc_B.

$$\mathrm{sc}_\mathrm{B}(s,p;\, b_1,b_2) \geq t$$
$$\sum_x b_1 x_\mathrm{min} - b_2 x_\mathrm{diff} \geq t$$
$$\sum_x b_1 x_\mathrm{min} - b_2(x_\mathrm{max} - x_\mathrm{min}) \geq t$$
$$\sum_x (b_1 + b_2) x_\mathrm{min} - b_2 x_\mathrm{max} \geq t$$
$$\mathrm{sc}_\mathrm{C}(s,p;\, c_1,c_2) \geq \mathrm{thr}_\mathrm{C}(|p|)$$

Next, we prove that sc_D dominates sc_C. Let p be a pattern and let $t = \mathrm{thr}_\mathrm{C}(|p|)$. Set $d_1 = c_1 + c_2$, $d_2 = c_2$, and $\mathrm{thr}_\mathrm{D}(i) = \mathrm{thr}_\mathrm{C}(i) + ic_2$. Then, for all s, the following series of equivalent transformations proves that sc_D dominates sc_C.

$$\mathrm{sc}_\mathrm{C}(s,p;\, c_1,c_2) \geq \mathrm{thr}_\mathrm{C}(|p|)$$
$$\sum_x c_1 x_\mathrm{min} - c_2 x_\mathrm{max} \geq t$$
$$\sum_x c_1 x_\mathrm{min} - c_2\left(|s| + |p| - \sum_x x_\mathrm{min}\right) \geq t$$
$$\sum_x (c_1 + c_2) x_\mathrm{min} - c_2|s| - c_2|p| \geq t$$
$$\sum_x (c_1 + c_2) x_\mathrm{min} - c_2|s| \geq t + c_2|p|$$
$$\mathrm{sc}_\mathrm{D}(s,p;\, d_1,d_2) \geq \mathrm{thr}_\mathrm{D}(|p|)$$

Next, we prove that sc_A dominates sc_D. Let p be a pattern and let $t = \mathrm{thr}_\mathrm{D}(|p|)$. Set $a_1 = \frac{d_1}{d_2} - 2$ and $\mathrm{thr}_\mathrm{A}(i) = \frac{\mathrm{thr}_\mathrm{D}(i)}{d_2} - i$. Then, for all s, the following series of equivalent transformations proves that sc_D dominates sc_C.

$$\mathrm{sc}_\mathrm{D}(s,p;\, d_1,d_2) \geq \mathrm{thr}_\mathrm{D}(|p|)$$
$$\left(\sum_x d_1 x_\mathrm{min}\right) - d_2|s| \geq t$$
$$\left(\sum_x d_1 x_\mathrm{min}\right) - d_2\left(\sum_x 2x_\mathrm{min} + \sum_x x_\mathrm{diff} - |p|\right) \geq t$$
$$\sum_x ((d_1 - 2d_2)x_\mathrm{min} - d_2 x_\mathrm{diff}) + d_2|p| \geq t$$
$$\sum_x \left(\frac{d_1 - 2d_2}{d_2} x_\mathrm{min} - x_\mathrm{diff}\right) + |p| \geq \frac{t}{d_2}$$
$$\sum_x \left((\frac{d_1}{d_2} - 2)x_\mathrm{min} - x_\mathrm{diff}\right) \geq \frac{t}{d_2} - |p|$$
$$\mathrm{sc}_\mathrm{A}(s,p;\, a_1) \geq \mathrm{thr}_\mathrm{A}(|p|)$$

Finally, we prove that sc_ℓ dominates sc_A. Let p be a pattern and let $t = \mathrm{thr}_\mathrm{A}(|p|)$. Set $w = \frac{1}{a_1}$ and $\mathrm{thr}_\ell(i) = \frac{\mathrm{thr}_\mathrm{A}(i)}{a_1}$. Then, for all s, the following series of equivalent transformations proves that sc_ℓ dominates sc_A.

$$\mathrm{sc}_\mathrm{A}(s,p;\, a_1) \geq \mathrm{thr}_\mathrm{A}(|p|)$$
$$\sum_x (a_1 x_\mathrm{min} - x_\mathrm{diff}) \geq t$$
$$\sum_x (x_\mathrm{min} - \frac{1}{a_1} x_\mathrm{diff}) \geq \frac{t}{a_1}$$
$$\mathrm{sc}_\ell(s,p;\, w) \geq \mathrm{thr}_\ell(|p|)$$

□

Acknowledgements
We would like to thank K. Sahlin for helpful early feedback.

Funding
Open Access funding enabled and organized by Projekt DEAL. This material is based upon work supported by the National Science Foundation under grant nos. 2138585. Research reported in this publication was also supported by the National Institutes of Health under Grant NIH R01GM146462 (to P.M.). The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health. Research is also funded in part by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie agreement [872539] (to T.S.). This work was supported by the BMBF-funded de.NBI Cloud within the German Network for Bioinformatics Infrastructure (de.NBI) (031A532B, 031A533A, 031A533B, 031A534A, 031A535A, 031A537A, 031A537B, 031A537C, 031A537D, 031A538A).

Availability of data and materials
Additional documentation of performed experiments and used data sets are available from https://github.com/medvedevgroup/eskemap.

Received: 1 November 2023 Accepted: 19 March 2024
Published online: 04 May 2024

References
1. Li H. Minimap2: pairwise alignment for nucleotide sequences. Bioinformatics. 2018;34(18):3094–100.
2. Sahlin K, Baudeau T, Cazaux B, Marchet C. A survey of mapping algorithms in the long-reads era. Genom Biol. 2023;24(1):1–23.
3. Medvedev P, Stanciu M, Brudno M. Computational methods for discovering structural variation with next-generation sequencing. Nat Method. 2009;6:13.
4. Alkan C, Kidd JM, Marques-Bonet T, Aksay G, Antonacci F, Hormozdiari F, Kitzman JO, Baker C, Malig M, Mutlu O, et al. Personalized copy number and segmental duplication maps using next-generation sequencing. Nat Genet. 2009;41(10):1061–7.
5. Jain C, Rhie A, Hansen NF, Koren S, Phillippy AM. Long-read mapping to repetitive reference sequences using Winnowmap2. Nat Method. 2022;19:705–10.
6. Šošić M, Šikić M. Edlib: a c/c++ library for fast, exact sequence alignment using edit distance. Bioinformatics. 2017;33(9):1394–5.
7. Roberts M, Hayes W, Hunt BR, Mount SM, Yorke JA. Reducing storage requirements for biological sequence comparison. Bioinformatics. 2004;20(18):3363–9.
8. Schleimer S, Wilkerson DS, Aiken A. Winnowing: Local algorithms for document fingerprinting. In: Proceedings of the 22nd International Conference on Management of Data (SIGMOD 2003), 2003;76–85.
9. Edgar R. Syncmers are more sensitive than minimizers for selecting conserved k-mers in biological sequences. Peer J. 2021;9:10805.
10. Irber L, Brooks PT, Reiter T, Pierce-Ward NT, Hera MR, Koslicki D, Brown CT. Lightweight compositional analysis of metagenomes with FracMinHash and minimum metagenome covers. bioRxiv (2022) https://doi.org/10.1101/2022.01.11.475838.
11. Hera MR, Pierce-Ward NT, Koslicki D. Debiasing FracMinHash and deriving confidence intervals for mutation rates across a wide range of evolutionary distances. bioRxiv (2022).
12. Belbasi M, Blanca A, Harris RS, Koslicki D, Medvedev P. The minimizer jaccard estimator is biased and inconsistent. Bioinformatics. 2022;38(Supplement_1):169–76. https://doi.org/10.1093/bioinformatics/btac244.
13. Blanca A, Harris RS, Koslicki D, Medvedev P. The statistics of k-mers from a sequence undergoing a simple mutation process without spurious matches. J Comput Biol. 2022;29(2):155–68.
14. Schulz T, Medvedev P. Exact Sketch-Based Read Mapping. In: Belazzougui, D., Ouangraoua, A. (eds.) 23rd International Workshop on Algorithms in Bioinformatics (WABI 2023). Leibniz International Proceedings in Informatics (LIPIcs), vol. 273, pp. 14–11419. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2023). https://doi.org/10.4230/LIPIcs.WABI.2023.14 . https://drops.dagstuhl.de/opus/volltexte/2023/18640.
15. Nurk S, Koren S, Rhie A, Rautiainen M, Bzikadze AV, Mikheenko A, Vollger MR, Altemose N, Uralsky L, Gershman A, et al. The complete sequence of a human genome. Science. 2022;376(6588):44–53.
16. Cechova M, Vegesna R, Tomaszkiewicz M, Harris RS, Chen D, Rangavittal S, Medvedev P, Makova KD. Dynamic evolution of great ape y chromosomes. Proc Natl Acad Sci. 2020;117(42):26273–80.
17. Hon T, Mars K, Young G, Tsai Y-C, Karalius JW, Landolin JM, Maurer N, Kudrna D, Hardigan MA, Steiner CC, et al. Highly accurate long-read hifi sequencing data for five complex genomes. Sci Data. 2020;7(1):399.
18. Ono Y, Asai K, Hamada M. Pbsim2: a simulator for long-read sequencers with a novel generative model of quality scores. Bioinformatics. 2021;37(5):589–95.
19. Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ. Basic local alignment search tool. J Mol Biol. 1990;215(3):403–10. https://doi.org/10.1016/S0022-2836(05)80360-2.