



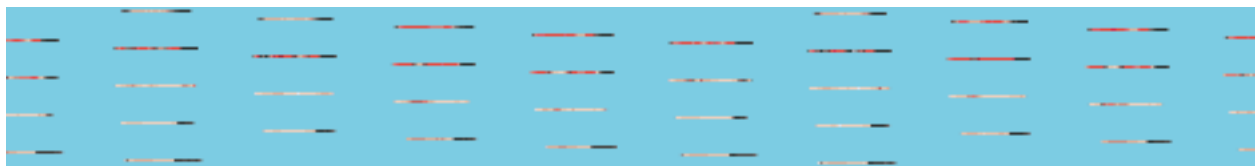
Efficient mapping of accurate long reads in minimizer space with mapquik

Baris Ekim, Kristoffer Sahlin, Paul Medvedev, et al.

Genome Res. published online June 30, 2023

Access the most recent version at doi:[10.1101/gr.277679.123](https://doi.org/10.1101/gr.277679.123)

P<P	Published online June 30, 2023 in advance of the print journal.
Accepted Manuscript	Peer-reviewed and accepted for publication but not copyedited or typeset; accepted manuscript is likely to differ from the final, published version.
Open Access	Freely available online through the <i>Genome Research</i> Open Access option.
Creative Commons License	This manuscript is Open Access. This article, published in <i>Genome Research</i> , is available under a Creative Commons License (Attribution-NonCommercial 4.0 International license), as described at http://creativecommons.org/licenses/by-nc/4.0/ .
Email Alerting Service	Receive free email alerts when new articles cite this article - sign up in the box at the top right corner of the article or click here .



To subscribe to *Genome Research* go to:
<https://genome.cshlp.org/subscriptions>

Published by Cold Spring Harbor Laboratory Press

Efficient mapping of accurate long reads in minimizer space with **mapquik**

Barış Ekim^{1, 2}, Kristoffer Sahlin³, Paul Medvedev^{4, 5, 6}, Bonnie Berger^{1, 2, *}, and Rayan Chikhi^{7, *}

¹Computer Science and Artificial Intelligence Laboratory (CSAIL), Massachusetts Institute of Technology (MIT), Cambridge, MA, USA

²Department of Mathematics, Massachusetts Institute of Technology (MIT), Cambridge, MA, USA

³Department of Mathematics, Science for Life Laboratory, Stockholm University, Stockholm, Sweden

⁴Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA, USA

⁵Department of Biochemistry and Molecular Biology, The Pennsylvania State University, University Park, PA, USA

⁶Huck Institutes of the Life Sciences, The Pennsylvania State University, University Park, PA, USA

⁷Department of Computational Biology, Institut Pasteur, Paris, France

Corresponding authors: bab@mit.edu, rchikhi@pasteur.fr

Running title: *Efficient mapping of long reads in minimizer space*

This research is partially based upon work supported by the National Science Foundation under Grants DBI-2138585 and OAC-1931531 (to P.M.). Research reported in this publication was also supported by the National Institutes of Health under Grants NIH R01HG010959 (for B.E. to B.B.), NIH 1R35GM141861 (to B.B.), and NIH R01GM146462 (to P.M.). This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grants agreements No. 872539 and 956229 (R.C.). R.C. was supported by ANR Transipedia, SeqDigger, Inception and PRAIRIE grants (ANR-18-CE45-0020, ANR-19-CE45-0008, PIA/ANR16-CONV-0005, ANR-19-P3IA-0001). The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health.

Abstract

1
2 DNA sequencing data continues to progress towards longer reads with increasingly lower sequencing
3 error rates. We focus on the critical problem of mapping, or aligning, low-divergence sequences
4 from long reads (e.g., PacBio HiFi) to a reference genome, which poses challenges in terms of
5 accuracy and computational resources when using cutting-edge read mapping approaches that are
6 designed for all types of alignments. A natural idea would be to optimize efficiency with longer
7 seeds to reduce the probability of extraneous matches; however, contiguous exact seeds quickly
8 reach a sensitivity limit. We introduce **mapquik**, a novel strategy that creates accurate longer seeds
9 by anchoring alignments through matches of k consecutively-sampled minimizers (k -min-mers) and
10 only indexing k -min-mers that occur once in the reference genome, thereby unlocking ultra-fast
11 mapping while retaining high sensitivity. We demonstrate that **mapquik** significantly accelerates
12 the seeding and chaining steps — fundamental bottlenecks to read mapping — for both the human
13 and maize genomes with $> 96\%$ sensitivity and near-perfect specificity. On the human genome,
14 for both real and simulated reads, **mapquik** achieves a $37\times$ speed-up over the state-of-the-art tool
15 **minimap2**, and on the maize genome, a $410\times$ speed-up over **minimap2**, making **mapquik** the fastest
16 mapper to date. These accelerations are enabled not only from minimizer-space seeding but also a
17 novel heuristic $\mathcal{O}(n)$ pseudo-chaining algorithm, which improves upon the long-standing $\mathcal{O}(n \log n)$
18 bound. Minimizer-space computation builds the foundation for achieving real-time analysis of
19 long-read sequencing data.

20 Introduction

21 Recent advances in DNA sequencing enable the rapid production of long reads with low error rates;
22 for example, Pacific Biosciences (PacBio) HiFi reads are 10 to 25 kbp in length with a $\leq 1\%$ error
23 rate. High-quality long reads have been used to accurately assemble genomes (Kolmogorov et al.
24 2019; Nurk et al. 2020; Ekim et al. 2021; Bankevich et al. 2022); complete the human genome (Nurk
25 et al. 2022); accurately detect small variants in challenging genomic regions (Olson et al. 2022); and
26 further elucidate the landscape of large structural variants in human genomes (Denti et al. 2022).
27 Critical to these successes are algorithms that perform genomic data analysis, such as reconstructing
28 a reference from reads (genome assembly) (Logsdon et al. 2020), or mapping reads to a reference
29 genome (read mapping) (Alser et al. 2021). With up to hundreds of gigabytes of sequenced data
30 per sample, analysis algorithms need to balance efficiency with high sensitivity and accuracy (the
31 percentage of reads mapped correctly) (Berger and Yu 2022), which is especially critical in rapid
32 sequencing-to-diagnostics (Owen et al. 2022; Galey et al. 2022).

33 We recently introduced the concept of minimizer-space computation (Ekim et al. 2021), where
34 only a small fraction of the sequenced bases are retained as a latent representation of the sequencing
35 data, enabling orders of magnitude improvements in efficiency without loss of accuracy. Minimizers
36 are sequences that are selected under some local or global minimum criteria (Roberts et al. 2004;
37 Schleimer et al. 2003), similar to locally consistent parsing (Şahinalp and Vishkin 1996). We
38 applied the minimizer-space concept to perform genome assembly of long and accurate reads in
39 minutes instead of hours — even for Human, and hypothesized that other types of genome analysis
40 tasks would benefit from it in the future (Ekim et al. 2021). We now pursue the intuition that
41 read mapping would also be amenable to minimizer-space computation, but there are multiple
42 algorithmic challenges to overcome due to the repetitive nature of genomes, biological variation
43 between samples and references, and sizable input data.

44 Two cornerstones of read alignment/mapping algorithms — ubiquitous in sequence analysis
45 pipelines — are the *seeding* and *chaining* steps, where each read is locally placed at a homologous
46 location in a reference genome. Seeding is carried out by finding pairs of matching seeds, which are
47 snippets of DNA with high-confidence (exact or inexact) matches between a query and a reference

48 genome. Seeds are initial matches that serve as anchoring points of alignments: They allow a
49 challenging instance to be split into a set of easier sub-instances by aligning only the shorter
50 intervals in-between seeds. In the short-read era, state-of-the-art alignment algorithms (e.g., BWA-
51 MEM (H Li 2013), Bowtie2 (Langmead and Salzberg 2012), and CORA (Yörükoğlu et al. 2016; Shajii
52 et al. 2021) typically relied on finding all possible seeds using a full-text index of the reference
53 genome. For long reads, there has been a recent breakthrough by sampling and indexing only a
54 relatively small number of short potential seeds from the reference genome, which has led to faster
55 and more accurate mapping tools, e.g., minimap2 (H Li 2018) and Winnomap2 (Jain et al. 2022b).
56 Chaining consists of finding maximal ordered subsets of seeds that all agree on a certain genomic
57 location (Jain et al. 2022a); seeds often have spurious matches due to their short lengths.

58 However, even the most recent long-read alignment tools are bottlenecks in analysis pipelines (Berger
59 and Yu 2022). For instance, the popular minimap2 software requires 12 CPU hours to map a typi-
60 cal PacBio HiFi dataset to the human genome, and Winnomap2 requires 15 CPU days, preventing
61 both real-time analysis of sequencing data (Loose et al. 2016) and efficient re-analysis of previously-
62 sequenced data collections (Edgar et al. 2022). A significant part of the minimap2 and Winnomap2
63 running times are in their seeding and chaining steps (Kalikar et al. 2022). These state-of-the-art
64 long-read alignment tools are sensitive and accurate, but their underlying seed constructs (k -mers)
65 are tailored to noisy reads. These small seed sizes induce longer computation times due to the
66 multiple potential mapping locations of seeds that need to be examined and filtered out. Re-
67 cent advances in short-read alignment methods have demonstrated that 98% of many organism’s
68 genomes are non-repetitive and can be uniquely aligned to with longer seeds (Edgar 2020). There-
69 fore, it seems natural to explore the use of longer seeds also in long reads: this idea is at the heart
70 of our approach.

71 Here, we provide a highly efficient and accurate read mapping tool for state-of-the-art and low-
72 error long-read data. We introduce mapquik, which instead of using a single minimizer as a seed
73 for a reference sequence (e.g., minimap2), builds accurate longer seeds by anchoring alignments
74 through matches of k consecutively-sampled minimizers (k -min-mers). Our approach borrows from
75 natural language processing where the tokens of the k -mers are the minimizers instead of base-

76 pair letters. We asked whether long-read mapping can be sped up by newly substituting k -mer
77 seeds with k -min-mers that occur uniquely in the genome. In this work, we evaluate the extent to
78 which k -min-mers can act as suitable seeds for accurate long-read alignment, and the performance
79 of `mapquik` in making use of them for HiFi-read alignment. Our work represents the promise of
80 unique, inexact seeds, such as k -min-mers, for ultra-fast long-read mapping and beyond.

81 Related work

82 `minimap2` (H Li 2018) is a *de facto* standard for mapping accurate long reads to a reference genome.
83 It applies a seed-and-extend strategy. Specifically, seeds are short k -mer minimizers, i.e., sequences
84 of length k that are lexicographically-minimal within a window of w consecutive k -mers. The
85 extension step is performed using an optimized implementation of the Needleman-Wunsch algo-
86 rithm (Needleman and Wunsch 1970) with an affine gap penalty. Several attempts have been
87 made to improve mapping performance compared to `minimap2`. `MashMap` (Jain et al. 2017) and
88 `MashMap2` (Jain et al. 2018) compute read-versus-genome and genome-versus-genome mappings
89 without an alignment step, and use $5\times$ less memory than `minimap2` at the expense of longer run-
90 time. In very recent work developed concurrently to ours, the aligner `BLEND` (Firtina et al. 2023)
91 uses strobemers (Sahlin 2021) and locality-sensitive hashing (inspired by Şahinalp and Vishkin
92 1996) to speed up `minimap2` end-to-end by about $2\times$; however, their seeding approach is integrated
93 into `minimap2`'s codebase, which is implemented and optimized for exact short seeds (minimizers
94 by Roberts et al. 2004), thus suffering from similar limitations (for the sake of completeness, we
95 compare to `BLEND` in Results).

96 Other works have focused on improving the sensitivity and accuracy of `minimap2`, at the expense
97 of speed. `Winnomap` (Jain et al. 2020) and `Winnomap2` (Jain et al. 2022b) use weighted minimizer
98 sampling and minimal confidently-alignable substrings to better align in highly-repetitive regions,
99 e.g., centromeres of chromosomes. `Winnomap2` is around $15\times$ slower than `minimap2` end-to-end,
100 yet uses around $3\times$ less memory.

101 In recent years, many research groups focusing on low-level and/or hardware-specific accelera-
102 tion have proposed ways to accelerate `minimap2`. `mm2-fast` (Kalikar et al. 2022), a CPU acceleration

103 of `minimap2` developed by Intel, achieved a $1.5\times$ acceleration over `minimap2`, end-to-end on HiFi
104 reads. The bulk of the speed-up was obtained in the alignment phase, not in the seeding-chaining
105 phase (Figure 1 in Kalikar et al. 2022). The domain-specific language Seq was developed to speed
106 up genomic sequence analysis, and achieved two orders of magnitude improvement in the homology
107 table reconstruction of the `CORA` read mapper (Yörükoğlu et al. 2016; Shajii et al. 2021).

108 There also exist efficient implementations that employ specialized hardware. `mm2-ax` (Sadasivan
109 et al. 2023), a recent GPU acceleration of `minimap2` developed by NVIDIA, achieved $2.5\times$ to $5.4\times$
110 acceleration over `mm2-fast` in the chaining step. Guo et al. also proposed GPU and FPGA
111 accelerations of `minimap2` that respectively achieve $7\times$ and $28\times$ speed-ups on the distinct task of
112 detecting pairwise read overlaps (Guo et al. 2019). These specialized hardware accelerators are
113 outside the scope of this work. Methods that accelerate short-read alignment, e.g., using cloud
114 computing resources (Schatz 2009), or optimized k -mer indexing (Almodaresi et al. 2021) are also
115 not considered here, given that they do not support long reads.

116 Results

117 Why use k -min-mers as alignment seeds instead of k -mers?

118 To motivate why k -min-mers are superior alignment seeds, compared to k -mers, for accurate long
119 reads, we formulate and verify the following two hypotheses: (1) Long exact k -mer seeds are
120 inadequate for accurate long-read alignment due to lack of sensitivity, whereas (2) k -min-mers are
121 adequate and also offer near-perfect specificity. An empirical analysis on actual HiFi reads with
122 average error rate of 0.1% over the entire human genome justifies these observations (Figure 1).
123 In the experiments that follow, we compared `Jellyfish` (Marçais and Kingsford 2011), `DSK` (Rizk
124 et al. 2013), `rust-mdbg` (Ekim et al. 2021) and `mapquik`. All code and data are available in the
125 `mapquik` repository (`experiments/figure-seeds/` folder).

126 To assess hypothesis (1), we examined the specificity of k -mers and k -min-mers as seeds (Fig-
127 ure 1, left panel) by recording their number of occurrences in the CHM13v2.0 as a proxy for the
128 number of potential mapping locations. The x -axis reports the *seed weight*, which for k -mers cor-

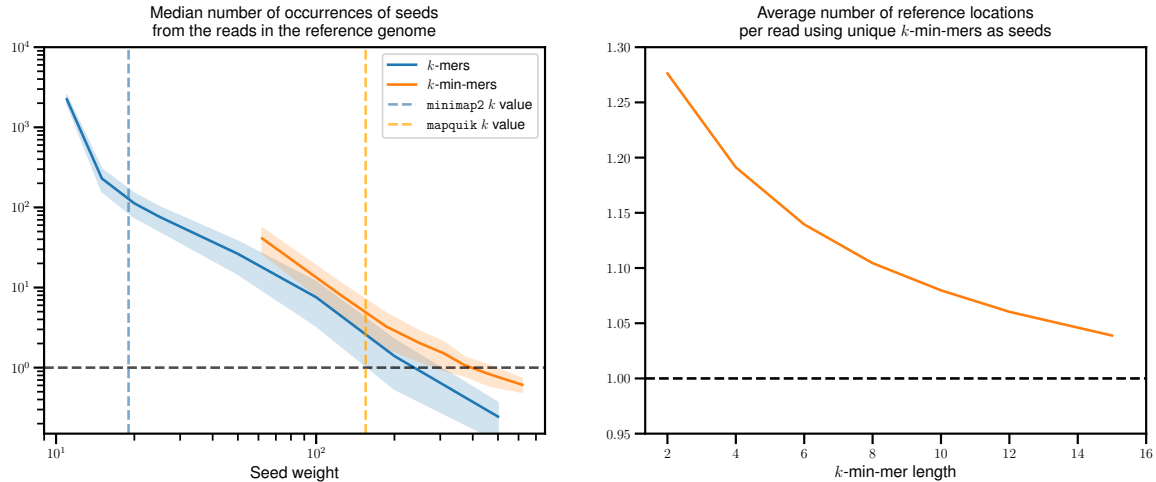


Figure 1. Increased sensitivity and specificity of k -min-mers versus long k -mers. Both panels use the human reference genome CHM13v2.0 and the HG002 DeepConsensus HiFi reads. Left panel: Each continuous line indicates the median abundance of read k -mers (darker blue line) and k -min-mers (lighter orange line) in the reference, averaged across all reads (the closer to 1, the better). The vertical dashed darker blue line (respectively, the lighter orange line) corresponds to the seed length chosen by `minimap2` (respectively, by `mapquik`). The median is computed from a random sub-sample of 50,000 HG002 reads. Right panel: Average number of reference genome locations indicated by seed matches for each read using k -min-mers (the closer to 1, the better). K -min-mer parameters are $\ell = 31$, $\delta = 0.01$ with $k = 2$ to 10 (left) and 2 to 15 (right). Regular k -mer lengths are $k = 12$ to 500.

129 responds to their length, and for k -min-mers corresponds to $\ell \times k$, i.e., the total number of bases
 130 in the k -min-mer minimizers. As indicated by the plot, k -mer seeds either have too many matches
 131 in the reference (from tens to thousands in the $k = 10$ to 100 range), or too few (below one match
 132 for $k > 300$). Notably, `minimap2` uses a default k value of 19 for HiFi reads, reflecting that it has
 133 to sift through hundreds of false matches for each read on average. On the other hand, k -min-mers
 134 have orders-of-magnitude fewer potential matches to examine, on average one to tens depending
 135 on k , owing to their longer lengths being less affected by genomic repetition.

136 To verify hypothesis (2), we showed that selecting all the k -min-mers that are seen only once
 137 in the reference genome is a viable indexing strategy (Figure 1, right). Indeed, the reads have, on
 138 average, 1.05 to 1.30 candidate reference genome locations when all their k -min-mers are queried
 139 on such an index. This finding hints that a read mapping algorithm based on k -min-mers is likely
 140 to immediately identify the correct genome location by querying all read k -min-mers, only paying
 141 attention to those that occur once in the genome. This algorithm potentially would not even

142 require a subsequent chaining step, given the low number of false matches to remove. This is in
143 stark contrast with existing k -mer-based algorithms, for which colinear chaining removes hundreds
144 of false seed matches per read.

145 Overview of minimizer-space read mapping with mapquik

146 To allow computation in minimizer space (Figure 2), we here develop `mapquik`, a read mapper based
147 on k -min-mer seeds. `mapquik` follows a seed-and-extend strategy used by most read mappers, with
148 two exceptions: (1) Only the k -min-mers that appear exactly once in all reference sequences are
149 indexed (Figure 2E-F), and (2) unlike a typical colinear chaining procedure that makes use of a
150 dynamic programming formulation (Figure 2A-D), e.g., in `minimap2` (H Li 2018), a linear-time
151 recursive extension step is performed for each initial k -min-mer match between the query and
152 the reference, followed by a novel, provably linear-time step we call *pseudo-chaining* that ensures
153 k -min-mer matches are approximately colinear. Figure 2G-I depicts the idea behind our pseudo-
154 chaining algorithm. Given a set of maximal k -min-mer matches, we chain only the k -min-mer
155 matches that are colinear with the match with the highest score (k -min-mer count). Unlike the
156 DP formulations used in regular colinear chaining, this step can be performed in linear time by
157 identifying the k -min-mer match with the highest score, and checking colinearity with the other
158 matches through a linear scan. Note that the matches in the output chain after this step are not
159 guaranteed to be *pairwise colinear*; however, thanks to the low number of k -min-mer matches,
160 pseudo-chaining performs adequately in practice while offering a substantial speed-up over classical
161 colinear chaining. The philosophy behind these drastic changes is that mapping long and accurate
162 reads to close reference genomes is “easy enough” that long minimizer-space seeds are sufficient for
163 the vast majority of the reads. The remaining few unmapped reads can in principle be fed to a
164 more sensitive, albeit slower, read mapper such as `minimap2` or `Winnomap2` (see Discussion).

165 Datasets and mapping evaluation

166 We used the complete human reference genome CHM13v2.0 for our evaluations. We constructed
167 a simulated dataset of long reads with 99% base-level accuracy and 24 kbp mean length using

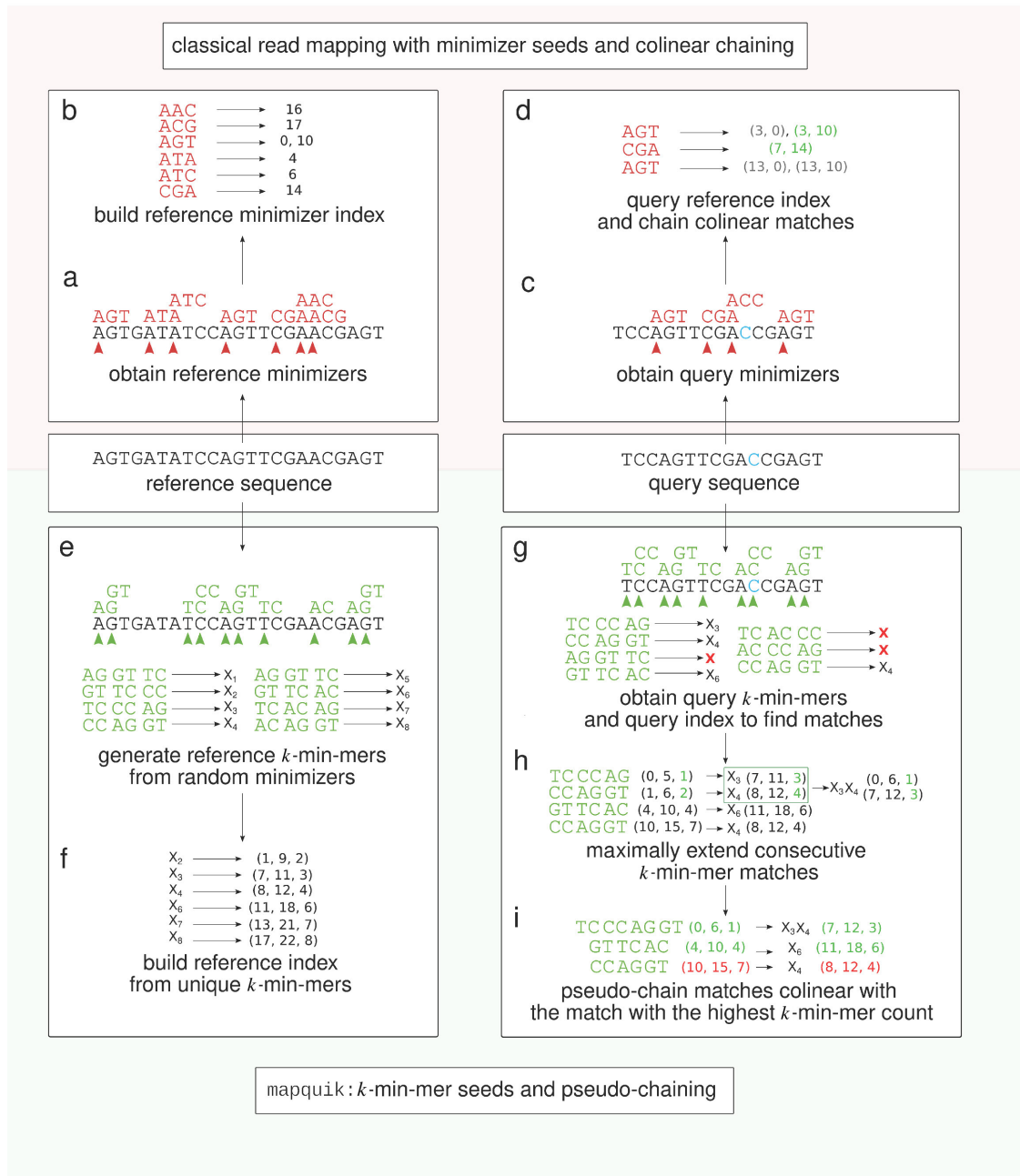


Figure 2. Overview of the long-read mapping pipeline using mapquik and comparison with state-of-the-art methods using minimizers as seeds. State-of-the-art read mappers such as minimap2 and Winnowmap2 (top, pink shaded) build an index for a reference sequence by computing window minimizers ($k = 3$, $w = 5$) (a), and storing the positions of the minimizers in the index (b). In order to map a query sequence using the reference index (top right, nucleotide C in blue denotes a sequencing error), mappers compute the minimizers on the query sequence (c), and find matches between the minimizers of the query and those in the reference index. Once minimizer matches are found, minimap2 and Winnowmap2 perform a colinear chaining step to output a high-scoring set of matches, using dynamic programming (d). In contrast, mapquik (bottom, green shaded) indexes reference sequences by generating *k*-min-mers, *k* consecutive, randomly-selected minimizers of length ℓ ($k = 3$, $\ell = 2$) (e), and storing only the *k*-min-mers that appear exactly once in the reference (f). mapquik stores the start and end position of each *k*-min-mer, along with the order in which the *k*-min-mers appear. To map a query sequence using the *k*-min-mer index, mapquik first obtains matches between the query and the reference index by querying the index with each query *k*-min-mer (g). *k*-min-mer matches are extended if the next immediate pair of *k*-min-mers also match (h). Instead of a colinear chaining step, mapquik performs a linear-time pseudo-chaining step to locate matches that are colinear with the match with the highest number of *k*-min-mers (i).

168 `pbsim` (Ono et al. 2013), mimicking HiFi reads at $10\times$ genome coverage. We also used real HiFi
169 reads for the HG002 individual corrected using DeepConsensus (Baid et al. 2022), at $30\times$ genome
170 coverage. For maize, we simulated reads from the maize RefSeq genome (GCF_902167145.1) at $30\times$
171 coverage using the same protocol as the human simulated reads. `mapquik` was run with default
172 parameters ($k = 5$, $\ell = 31$, $\delta = 0.01$, $\beta = 4$, $\mu = 11$, $\varepsilon = 2000$); we provide an extensive evaluation
173 of the parameters k , ℓ , and δ in Supplemental Note S8 and Supplemental Figure S3, and of β and
174 μ in Figure 3. All other tools were run with default parameters in HiFi mapping mode. Command
175 lines and versions are given in Supplemental Note S2.

176 For simulated reads, we assessed mapping accuracy using the `mapeval` command of the `paftools`
177 software distributed in the `minimap2` package. A read is considered to be correctly mapped if the
178 intersection between the true and mapped reference intervals is at least 10% of their union. For
179 real reads, we evaluated the concordance between the alignments of `minimap2` and `mapquik` using
180 a custom script (`experiments/intersect_pafs.py` in the GitHub repository and Supplemental
181 Code), similar to `mapeval`. In our evaluation of both the simulated and real datasets, we focus
182 on reads of the highest mapping quality (Q60). Mappers report a *mapping quality* metric for
183 each read, indicating their confidence that the read is mapped at the right location, as an integer
184 between 0 and 60 where 60 corresponds to the highest confidence. Reads with low mapping quality
185 are less frequent and are often removed in downstream applications (e.g., the popular variant
186 calling pipeline GATK (McKenna et al. 2010) filters out reads with $\text{MAPQ} \leq 20$ by default). With
187 `minimap2` results, `mapeval` reports only two mapping quality groups (Q0 and Q60), with no value
188 in-between. With `mapquik`, we report a MAPQ of 60 if the output pseudo-chain has score $\geq \mu$ or
189 length $\geq \beta$, and 0 otherwise.

190 **mapquik achieves faster and accurate mapping of HiFi reads to the human and** 191 **maize genomes**

192 Table 1 demonstrates the enhanced overall performance of `mapquik` and other evaluated methods
193 (`minimap2`, `mm2-fast`, `Winnowmap2`, and concurrently-developed BLEND) on mapping simulated and
194 real PacBio HiFi reads to the human and maize genomes. We demonstrate `mapquik`'s $37\times$ and $975\times$

195 relative speedup over state-of-the-art mapping methods `minimap2` and `Winnomap2`, respectively,
196 for both simulated and real HiFi reads from the human genome. We further apply `mapquik` to
197 mapping simulated HiFi reads from the highly-repetitive maize genome, and demonstrate a 410×
198 speed-up over `minimap2`, with remarkably no loss of sensitivity nor accuracy.

199 On the simulated human dataset, `mapquik` is 54× faster than `minimap2`. `mapquik` maps 95.8%
200 of reads at a MAPQ score of 60 (Q60), indicating a high-confidence match, with 2 errors. In
201 contrast, other mappers map 96.1% to 98.6% of reads at Q60 with also no/almost no error. On
202 the real dataset from HG002, a similar trend is observed. `mapquik` outperforms all mappers except
203 `Winnomap2` in terms of percentage of reads mapped (96.1%), as opposed to 92.2 to 97.9% for
204 the other tools. The concordance of `minimap2` and `mapquik` mappings is 99.8% on the Q60 reads
205 mapped by `minimap2`. In Supplemental Table S1, we provide the fraction of the reference genome
206 covered by each tool for all experiments. All mappers required less than 14 gigabytes of memory
207 on the human genome (`MashMap2` was not further evaluated as it took over 13 wall-clock hours on
208 the simulated human dataset, and does not output mapping quality scores.)

209 A highlight of `mapquik` is a 410× mapping speed-up compared to `minimap2` on the maize
210 genome. Remarkably, this speed-up comes with near-perfect precision for `mapquik` and no loss of
211 sensitivity, as `mapquik` reports the second highest number of mapped reads at mapping quality 60
212 across all tools after `Winnomap2`. Of note, `Winnomap2` has faster performance on maize than the
213 human genome and than that of `minimap2` on the maize genome.

214 We further investigated why some reads were mapped at lower qualities than Q60 or were not
215 mapped at all. Out of 58,004 reads from the simulated human dataset that were not mapped at Q60
216 by `mapquik`, 94.4% of these reads intersected with centromeric/satellite regions of chromosomes, as
217 reported by `BEDTools` (Quinlan and Hall 2010) using the `chm13v2.0_censat_v2.0.bed` annotation
218 from the Telomere-to-Telomere (T2T) consortium. Thus, the vast majority of reads not aligned at
219 Q60 correspond to challenging genomic regions that would likely have been masked in downstream
220 analyses, making their lack of alignment potentially inconsequential. We hypothesize that similar
221 conclusions hold on real data, but this cannot be ascertained as the true reference interval of each
222 read is not known.

Tool name	Mapped Q60	Q < 60 or missed	Wrong Q60	Memory (GB)	Time (s)	Speed-up
CHM13 10× coverage simulated 24 kbp HiFi reads						
minimap2	1,340,993	27,819	0	13.1	978	1x
mm2-fast	1,340,993	27,819	0	13.1	805	1.21x
Winnowmap2	1,350,016	18,796	6	11.8	21,009	0.05x
BLEND	1,315,676	53,136	1	6.8	188	5.2x
mapquik	1,310,808	58,004	2	12.2	18	54.33x
HG002 30× coverage real 24 kbp HiFi reads (DeepConsensus)						
minimap2	3,611,990	303,304	N/A	10.8	3,146	1x
mm2-fast	3,611,983	303,311	N/A	10.8	2,693	1.17x
Winnowmap2	3,835,225	80,069	N/A	8.8	83,180	0.04x
BLEND	3,708,582	206,712	N/A	6.0	626	5.03x
mapquik	3,760,677	154,617	N/A	12.1	85	37.01x
Maize 30× coverage simulated 24 kbp HiFi reads						
minimap2	2,807,058	58,730	1	15.1	17,194	1x
mm2-fast	2,807,059	58,729	1	15.0	14,693	1.17x
Winnowmap2	2,854,041	11,747	93	14.1	15,376	1.12x
BLEND	2,836,244	29,544	5	4.8	349	49.27x
mapquik	2,837,524	28,264	2	13.1	42	409.38x

Table 1. Mapping statistics of **mapquik** and other evaluated methods (**minimap2**, **mm2-fast**, **Winnowmap2**, **BLEND**) on simulated and real HiFi human reads, and simulated maize HiFi reads. Only reads with reported mapping quality over 60 were included in columns 1 and 3. Incorrectly-aligned reads were detected using **paftools mapeval**. “Time” column consists of wall-clock times and includes on-the-fly reference indexing. Reads were **ungzipped**. Tools were run on 10 threads. For **Winnowmap2**, the time for reference k -mer counting (**meryl**) was not included. The last column indicates the speed-up over **minimap2** taken as a baseline.

223 Efficient genome indexing using k -min-mers

224 Table 2 demonstrates the computing resources necessary to index a human genome for **mapquik** as
225 compared to **minimap2**, **mm2-fast**, **Winnowmap2** and the concurrently-developed **BLEND**. Although
226 indexes created by the alternative mapping methods to **mapquik** are different in nature, a similar
227 order of magnitude of numbers of sequences (tens to hundreds of millions) end up being indexed.
228 **minimap2** and **mm2-fast** index positions of windowed minimizers. Those minimizers are different
229 from the (ℓ, δ) -minimizers defined in this article. **Winnowmap2** indexes weighted minimizers to
230 increase the accuracy of seeds. **BLEND** (in HiFi mapping mode) indexes a locality-sensitive seed
231 built from strobemers (Sahlin 2021), which are chains of consecutive windowed minimizers (Roberts

232 et al. 2004), different in nature from k -min-mers.

233 `mapquik` indexing is $9\times$ faster than `minimap2` and $6\times$ faster than `BLEND`. The `mapquik` seeding
 234 strategy provides ultra-fast construction of an index that records unique k -min-mer positions across
 235 the reference genome. This index is of independent interest, for example, for indexing larger
 236 databases such as RefSeq (W Li et al. 2021). We anticipate that this index will have other uses
 237 beyond long-read mapping, such as large-scale sequence search. We have already demonstrated the
 238 usefulness of performing k -min-mer search in anti-microbial resistance tracking (Ekim et al. 2021),
 239 albeit in earlier work that did not benefit from the speed-up of such an index presented here.

Tool name	Indexed sequences	Singletons	Memory (GB)	Wall-clock (seconds)
<code>minimap2</code>	215,125,355	92%	10.1	33
<code>mm2-fast</code>	215,125,355	92%	10.1	28
<code>Winnomap2</code>	23,616,987	41%	2.6	64
<code>BLEND</code>	111,799,540	97%	5.3	23
<code>mapquik</code>	39,603,738	100%	12.1	3.4

Table 2. Indexing a reference human genome using `mapquik` and other evaluated methods (`minimap2`, `mm2-fast`, `Winnomap2`, `BLEND`). The CHM13v2.0 reference sequence was given as input to each tool. Tools were run using 10 threads with warm cache, i.e., the reference genome file was already pre-loaded in memory. The “Indexed sequences” column indicates the number of distinct sequences that are keys of the final index. The “Singletons” column indicates how many indexed sequences have only one position in the reference genome.

240 Comparison with `BLEND`

241 While concurrently-developed, we compare to `BLEND` in Tables 1 and 2 for completeness, and demon-
 242 strate that `mapquik` achieves a $7\times$ speed-up overall (indexing plus chaining) over `BLEND` on human
 243 data.

244 Robustness of mapping shorter reads with `mapquik`

245 We next asked how `mapquik` would perform on smaller reads than those offered by HiFi, which are
 246 currently available for lengths ranging from 10 kbp to 25 kbp. Future technologies may also offer
 247 long high-accuracy reads, although HiFi is currently the only high-throughput solution available.
 248 To determine whether `mapquik` is also suitable for mapping HiFi reads shorter than 24 kbp, we
 249 tested `mapquik` on other datasets with overall shorter reads using both real and simulated data.

250 To assess `mapquik`'s robustness in mapping reads of varying length, we ran `mapquik` on real HG002
251 HiFi 16 kbp length reads from DeepConsensus (Baid et al. 2022), using identical parameters as
252 in the run with 24 kbp read length. `mapquik`'s performance on 16 kbp reads is strikingly similar
253 to that on 24 kbp reads: the running time increased by only 4%, memory usage remains nearly
254 identical, and the concordance with `minimap2` remains 99.8%.

255 To determine if `mapquik` is still robust to larger variation in read length, we simulated seven
256 samples with $10\times$ coverage from CHM13, using the same protocol, except that a different average
257 read length is selected for each sample (from 2 kbp to 14 kbp, by 2 kbp increments). Supplemental
258 Figure S2 reports that both `minimap2` and `mapquik` map over 93% of the reads at Q60 in sam-
259 ples having read lengths of 10 kbp or higher, remarkably without tuning their parameters. Both
260 `minimap2` and `mapquik` are challenged by read lengths of 2 kbp; with their default HiFi parame-
261 ters, they respectively map 89% and 27% of the reads at Q60. Note that `mapquik` maps 63% of
262 the remaining 2 kbp reads at Q0, with a low mapping error rate (1%) as reported by `mapeval`. We
263 expect `mapquik` has more difficulty mapping these 2 kbp reads due to its longer seed weight than
264 `minimap2`.

265 Limitations of our study

266 We evaluated the limitations of our method with respect to several aspects: the choice of k , inter-
267 nal cut-off parameters, and the requirement of having low divergence between the reads and the
268 reference.

269 As seen in Table 1, `mapquik` is unable to map some of the reads, mostly in low-complexity regions
270 of the genome, partially because no indexed k -min-mer exists, but also because of the lack of a long
271 enough (i.e., high-scoring) pseudo-chain. We examined the magnitude of the second effect. Figure 3
272 (left) shows the total number of reads (y -axis) whose highest pseudo-chain score is given on the
273 x -axis. Read numbers follow approximately a Gaussian distribution, confirming the soundness of
274 filtering out the leftmost tail containing erroneous pseudo-chains. Figure 3 (right) depicts filtering
275 thresholds by showing the percentage of reads mapped correctly (y -axis) at each pseudo-chain score
276 per read (x -axis), depending on the choice of k . The monotonically-increasing relationship suggests

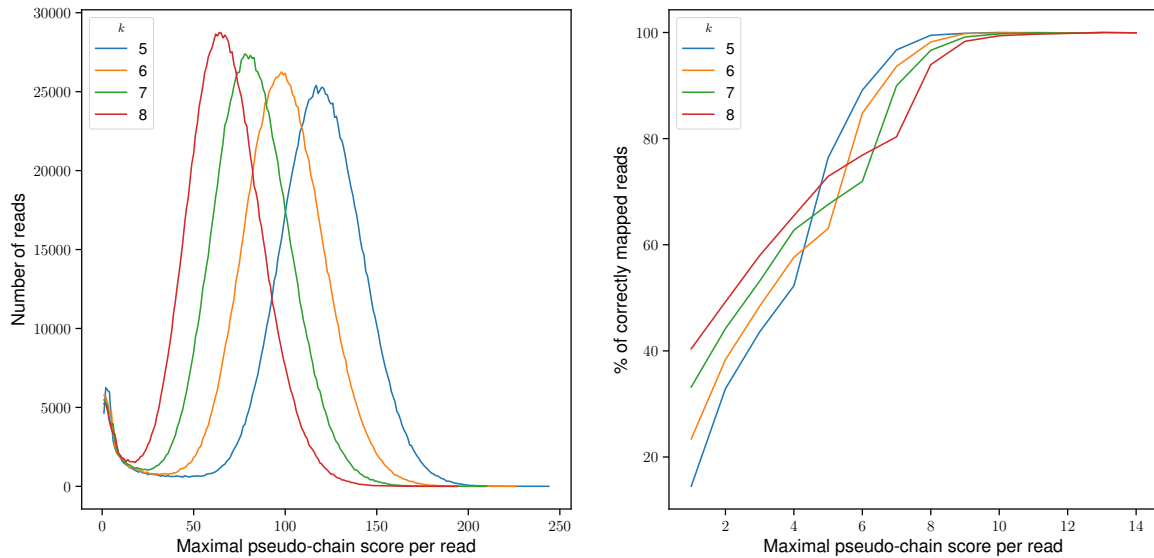


Figure 3. Effect of pseudo-chain score on mapping accuracy. The x -axis in both subfigures corresponds to the score of the maximal pseudo-chain per read; on the left, the y -axis denotes the total number of reads with the corresponding maximal pseudo-chain score, and on the right, the percentage of reads that are correctly mapped (assessed by `paftools mapeval`) with the corresponding maximal pseudo-chain score. Only the scores below a threshold s , where s denotes the maximum score at which a read was mapped incorrectly, are plotted. The parameters used for `mapquik` were $k = 5$ to 8 , $\ell = 31$, $\delta = 0.01$.

277 that the pseudo-chain score is a reliable proxy for evaluating mapping accuracy. Mapping accuracy
 278 plateaus around pseudo-chain scores of 9 to 11. In our implementation, we apply a threshold of
 279 pseudo-chain score $\geq \mu$ with $\mu = 11$ by default (Algorithm 3 in Supplemental Note S1).

280 The mapping performance of `mapquik` degrades markedly when identity between reads and the
 281 reference is lower than 97%, and less than 1% of the reads are mapped at Q60 for identities below
 282 93% (Supplemental Figure S1). Therefore, `mapquik` is not suitable for mapping PacBio CLR reads,
 283 and potentially also Oxford Nanopore reads until base-calling consistently reaches identity levels
 284 above 98%. Remarkably, the mapping error rate at Q60 remains negligible at all identity levels.

285 Discussion

286 Our work demonstrates that minimizer-space computation can be successfully applied to read
 287 mapping, and overcomes a significant barrier to real-time analysis of sequencing data that simply
 288 using longer k -mers or minimizers as seeds cannot. We show for the first time that indexing

289 only the long minimizer-space seeds (k -min-mers) that occur uniquely in the genome is sufficient
290 for sensitive and specific mapping. By leveraging the high specificity of these seeds, we are able
291 to devise a provably $\mathcal{O}(n)$ time (heuristic) pseudo-chaining algorithm, which improves upon the
292 subsequent best $\mathcal{O}(n \log n)$ runtime of all other known colinear chaining methods (Jain et al. 2022a),
293 without loss of performance in practice.

294 We previously used minimizer-space computation for fast and accurate genome assembly; how-
295 ever, long-read mapping is entirely different as no de Bruijn graph is constructed. This work thus
296 establishes the versatility of k -min-mers in algorithms for biological sequences. As sequencing reads
297 are getting longer and more accurate, we anticipate that our approach will particularly benefit from
298 technological advances: longer reads will be increasingly easier to map with minimizer-space seeds.

299 A potential concern with providing a faster alignment method is the loss of sensitivity in “hard-
300 to-map” regions, such as centromeres or structural variant breakpoints. In Supplemental Note S9
301 and Supplemental Figures S4 and S5, we investigated the missed genomic regions by both `minimap2`
302 and `mapquik` and found that they overlap by more than 90%. One could partially mitigate this
303 concern by performing a conservative, but fast alignment of reads using `mapquik`, and remapping
304 the unmapped reads with `minimap2` or `Winnowmap2` to increase alignment sensitivity, while keeping
305 the efficiency of `mapquik`. Another possible direction would be to use a pangenomic reference to
306 provide more indexable k -min-mers.

307 Future extensions of this work include implementing base-level alignment, which will allow
308 the design of a complete single-nucleotide variant calling pipeline, as well as one with structural
309 variant calling built on top of `mapquik`. Currently, `mapquik` is directly usable for quickly finding
310 genomic positions of HiFi reads, which enables many downstream applications such as sorting reads
311 by genome position, separating them by chromosome, filtering non-human reads, etc. Since the
312 k -min-mer matches have k exact matches of minimizers of length ℓ , only the regions in-between
313 minimizers and in-between neighboring k -min-mer matches would need to be aligned, which po-
314 tentially lowers both the memory usage and runtime of the alignment step. Another potential
315 improvement to `mapquik` would be to refine the mapping quality scores, in light of the observa-
316 tions made in Figure 3 that the pseudo-chain score reliably tracks mapping accuracy. We expect

317 minimizer-space computation to further mitigate challenges in other sequencing data analysis tasks,
 318 such as sequence-to-graph alignment, metagenomic binning, and similarity search.

319 Methods

Although Figure 2 and Algorithm 1 describe the steps of `mapquik`, we go into more detail below.

Algorithm 1 The `mapquik` algorithm

Input: A collection of reference sequences R and a collection of query sequences Q . We assume global parameters $k, \ell, \delta, \varepsilon, \mu, \beta, f$, and φ are pre-defined.

Output: Query-to-reference mappings S .

```

1: function MAP( $R, Q$ )
2:    $I \leftarrow \{\}$                                 ▷ Empty hash table of reference  $k$ -min-mers
3:    $L \leftarrow \{\}$                                 ▷ Empty hash table of sequence lengths
4:    $S \leftarrow \{\}$                                 ▷ To store mapping results
5:   for  $r \in R$  do
6:      $X_r \leftarrow \text{EXTRACT}(r)$                 ▷ Extract  $k$ -min-mers from reference sequence into a list
7:      $L[r_{\text{ID}}] \leftarrow |r|$                     ▷ Store length of  $r$ 
8:     for every  $x_r^i \in X_r$  do
9:       if  $\varphi_r^i$  is not a key in  $I$  then
10:         $I[\varphi_r^i] \leftarrow (r_{\text{ID}}, s_r^i, e_r^i, \pi_r^i, i_r)$   ▷ Record the reference ID  $r_{\text{ID}}$  and tuple of  $x_r^i$ 
11:       else  $I[\varphi_r^i] \leftarrow ()$                 ▷ Assign empty entry
12:   for  $q \in Q$  do
13:      $L[q_{\text{ID}}] \leftarrow |q|$                     ▷ Store length of  $q$ 
14:      $X_q \leftarrow \text{EXTRACT}(q)$                 ▷ Extract  $k$ -min-mers from query sequence into a list
15:      $H \leftarrow \text{MATCH}(X_q, I, q_{\text{ID}})$           ▷ Generate maximal  $k$ -min-mer matches
16:      $S[q] \leftarrow \text{PSEUDOCHAIN}(H, q_{\text{ID}}, L, \varepsilon, \mu, \beta)$   ▷ Pseudo-chain generated  $k$ -min-mer matches
17:   return  $S$ 

```

320

321 Methodological formalization

322 For a fixed integer $\ell > 1$, let $f : \Sigma^\ell \mapsto [0, H]$ be a random hash function that maps strings of length
 323 ℓ to integers between 0 and H . In practice, we use a 64-bit hash function. Moreover, we require
 324 f to be invariant with respect to reverse-complements, i.e., an ℓ -mer and its reverse complement
 325 map to the same integer. For a density $0 < \delta < 1$, we define $U_{\ell, \delta}$ as the set of all ℓ -mers m with
 326 $f(m) < \delta \cdot H$. We refer to the elements of $U_{\ell, \delta}$ as (ℓ, δ) -minimizers. Note that whether an ℓ -mer is
 327 a (ℓ, δ) -minimizer does not depend on any sequence besides the ℓ -mer itself.

328 Let S be a sequence of length $\geq \ell$. We define its *minimizer-space representation* M_S as an
 329 ordered list of (ℓ, δ) -minimizers that appear in S . Since the contents of $U_{\ell, \delta}$ only depend on f , and
 330 as long as the same hash function f is used, M_S will always be a subset of $U_{\ell, \delta}$. We omit S from
 331 the subscript when it is obvious from the context.

332 Let M be a minimizer-space representation of S and let $k > 0$ be a fixed integer parameter.
 333 We define a *k-min-mer* x^i of S as an ordered list of k consecutive minimizers in M starting from
 334 index i , i.e., $x^i = (m_i, \dots, m_{i+k-1})$. We denote the ordered list of all k -min-mers $(x^0, \dots, x^{|M|-k})$
 335 of S as X_S . We omit S from the subscript when it is obvious from the context.

336 In order to avoid explicitly storing nucleotide sequences of minimizers, we use a random hash
 337 function φ that maps sequences of k ℓ -mers to 64-bit hash values. We define φ so that it is invariant
 338 to reversing the order of the k -min-mer, i.e., $\varphi(x^i) = \varphi(\text{REV}(x^i))$, where $\text{REV}(x^i)$ denotes the list of
 339 minimizers of x^i with the order reversed. This is achieved by hashing x^i and $\text{REV}(x^i)$, and taking
 340 $\varphi(x^i)$ to be the minimum value.

341 Then, instead of storing X as an ordered list of k -min-mer sequences, we store the i^{th} k -min-mer
 342 x^i of X as a tuple $(i, \varphi^i, s^i, e^i, \pi^i)$, where

- 343 • i is the rank of x^i in X ,
- 344 • $\varphi^i = \varphi(x^i)$,
- 345 • s^i and e^i the nucleotide start and end positions of x^i , i.e., the start position of minimizer m_i
 346 and the end position of minimizer m_{i+k-1} , respectively, on S ,
- 347 • π^i a Boolean variable that evaluates to 1 if, in the construction of φ , the hash of $\text{REV}(x^i)$ was
 348 smaller than that of x^i , and 0 otherwise.

349 We call this the *tuple of x^i* . When the sequence S is not obvious from the context, we add it as
 350 a subscript in the above notation, e.g., π_S^i . The hash functions f and φ are instantiated prior to
 351 constructing the k -min-mer lists for the input sequences. We consistently use the same functions f
 352 and φ when selecting minimizers and consequently building k -min-mer lists for both the reference
 353 and query sequences throughout.

354 Indexing reference sequences

355 The `mapquik` index I is a hash table that associates a k -min-mer x to its unique position in the
 356 reference genome, whether or not it appears reverse-complemented, and its rank in the list of k -
 357 min-mers X . As described in “Methodological formalization”, this information is represented by a
 358 tuple in the form $(r_{ID}, s^i, e^i, \pi^i, i)$. We construct I using a two-pass approach. First, we call the
 359 `EXTRACT` function. It builds the list X by a linear scan through the reference sequence, during
 360 which it identifies minimizers and outputs each k consecutive minimizers, together with their hash
 361 values. We use the same efficient algorithm as `rust-mdbg` (Ekim et al. 2021), which runs in $\mathcal{O}(|X|)$
 362 time, so we do not include the pseudo-code for `EXTRACT` here. Second, we load every entry of X
 363 into a hash table I , indexed by the hash value. In this step, we discard from I any k -min-mer that
 364 appears in more than one reference location. Therefore, the hash table holds a single value per
 365 distinct k -min-mer key.

366 Locating and extending query-to-reference k -min-mer matches

367 Informally, a *k -min-mer match* is a stretch of k -min-mer seeds which appear consecutively both in
 368 the reference and the query (under the hash function used). Note that all matches are unique, in
 369 the sense that all seed k -min-mers appear only once in the genome, by definition. Given a query
 370 q and a reference r , we formally define a *match* as a triple (i, j, c) such that $0 \leq i \leq |X_q| - c$,
 371 $0 \leq j \leq |X_r| - c$, $c \geq 1$, and, for all $0 \leq c' < c$, $\varphi_q^{i+c'} = \varphi_r^{j+c'}$. A match (i, j, c) is said to be *maximal*
 372 if cannot be further extended to the right or left, i.e., (1) either $i = 0$, $j = 0$, or $\varphi_q^{i-1} \neq \varphi_r^{j-1}$, and
 373 (2) either $i + c = |X_q|$, $j + c = |X_r|$, or $\varphi_q^{i+c} \neq \varphi_r^{j+c}$.

374 For each query q , the `mapquik` algorithm first builds the list of k -min-mers X_q , sorted in increas-
 375 ing order of location (line 14). Then, it runs the `MATCH` routine (line 15), which finds all maximal
 376 matches between q and the reference. `MATCH` works by scanning through X_q and, for each seed
 377 $x \in X_q$, using the reference index I to see if x exists in the reference. If it does, then it marks
 378 the start of a match and proceeds to extend the match to the right as long as the seeds continue
 379 to match. Since we only have to query the index with the hash value of each k -min-mer in X_q ,
 380 the extension procedure can be done during a single linear pass over the elements of X_q , and thus

381 takes $\mathcal{O}(|X_q|)$ time (assuming $\mathcal{O}(1)$ hashing, look-ups, and insertions). Care must be taken due to
 382 reverse complements, which can change the direction of matching, but we omit these details. For
 383 completeness, the full algorithm (Algorithm 2) and the proof of maximality of k -min-mer matches
 384 (Supplemental Note S7) are in Supplemental Material.

385 In theory, generating a single 64-bit hash value for each unique k -min-mer could lead to hash
 386 collisions and, consequently, lead to false k -min-mer matches. However, since a k -min-mer match
 387 can only be extended with a consecutive k -min-mer match, k -min-mers that match an entry in the
 388 reference due to a hash collision are likely to be singletons and get filtered out in the pseudo-chaining
 389 step, with no decrease in final accuracy.

390 From maximal k -min-mer matches to pseudo-chains

391 Recall that k -min-mer matches are extended based solely on whether the next immediate k -min-
 392 mer of q matches the next immediate k -min-mer of r . However, k -min-mer matches on q might
 393 occur in multiple non-overlapping positions on r (due to sequencing errors or biological variation
 394 in q), i.e., for two matches (i, j, c) and (i', j', c') between q and r , it is not necessarily true that
 395 $|i - j| = |i' - j'|$.

396 In order to output a list of matches that are likely to be true positives while avoiding a
 397 computationally-expensive dynamic programming procedure, `mapquik` uses a *pseudo-chaining* pro-
 398 cedure which finds k -min-mer matches between a query q and a reference r that are gap-bounded
 399 colinear, but not all pairwise colinear.

400 Concretely, let $h = (i, j, c)$ and $h' = (i', j', c')$ be two matches, and consider the coordinates
 401 (s_q, e_q, π) and (s_r, e_r, π) of respectively the first and last k -min-mers of h , and (s'_q, e'_q, π') and
 402 (s'_r, e'_r, π') for the k -min-mers of h' . Let $g > 0$ be a fixed-integer gap upper bound. We say that h
 403 and h' are *gap-bounded colinear* if

- 404 • the matches are on the same relative strand, i.e., $\pi = \pi'$,
- 405 • the reference start positions of the matches agree with the order of the matches, i.e., if $\pi = 0$,
- 406 $s_r < s'_r$; or if $\pi = 1$, $s'_r < s_r$, and

- 407 • the length of the gap between the two matches in the query is similar to that on the reference,
 408 i.e., if $\pi = 0$, $|(s'_q - e_q) - (s'_r - e_r)| < g$; or if $\pi = 1$, $|(s'_q - e_q) - (s_r - e'_r)| < g$.

409 This last *gap length difference* condition, on top of the traditional definition of colinearity (the
 410 first two conditions), ensures that the regions outside the matches are similar in length. A similar
 411 parameter (ϵ) is used in `minimap` (H Li 2016).

412 Let H be the list of all maximal k -min-mer matches between a read q and a reference sequence
 413 r . We define a *pseudo-chain* Ψ^i as the list of all matches in H that are colinear with the i^{th} match
 414 in H ; we say that Ψ^i is *anchored at i* . Note that even though every match in Ψ^i is colinear with
 415 the i^{th} match in H , it is not necessarily true that every pair of matches in Ψ^i are pairwise colinear,
 416 thus Ψ^i does not satisfy the criteria of chains as defined in other works (e.g., H Li 2018).

The *score* of a pseudo-chain Ψ^i is the number of matching k -min-mers in Ψ^i , i.e.,

$$\text{SCORE}(\Psi^i) = \sum_{h \in \Psi^i} c(h)$$

417 where $c(h)$ denotes the number of matching k -min-mers in match h . Since the maximal matches
 418 in Ψ^i are guaranteed to not share any query k -min-mers because both the start and end locations
 419 of each maximal match are distinct, the cumulative sum of the number of matching k -min-mers in
 420 each match in Ψ^i equals the number of total matching k -min-mers in Ψ^i .

421 Computing high-scoring pseudo-chains in linear time

422 We now introduce a novel algorithm for computing a single high-scoring pseudo-chain Ψ^* for a query
 423 q given a list of maximal matches, and prove that it runs in $\mathcal{O}(n)$ time. The match extension step
 424 outputs a hash table H of maximal k -min-mer matches per reference, indexed by their reference
 425 identifier. In the pseudo-chaining step, however, the objective is to output a single list of matches
 426 between q and a single reference, even though H might contain matches between q and more than
 427 one reference sequence. We first initialize $\Psi^* = []$, and iterate over the key-value tuples in H ,
 428 processing each list of maximal matches $H_{q,r}$ for a single reference r one by one. In every iteration,
 429 we obtain a candidate pseudo-chain $\Psi_{q,r}$ from the list of maximal matches $H_{q,r}$ by computing

430 the pseudo-chain anchored at the match in $H_{q,r}$ with the highest number of matching k -min-
 431 mers. After computing $\Psi_{q,r}$, we compare its score to that of Ψ^* , and replace Ψ^* with $\Psi_{q,r}$ if
 432 $\text{SCORE}(\Psi_{q,r}) > \text{SCORE}(\Psi^*)$. At the end of the loop, Ψ^* will be the highest-scoring pseudo-chain
 433 out of all possible candidate pseudo-chains per reference sequence in H .

434 Finally, if the pseudo-chain Ψ^* has score $\geq \mu$ or length $\geq \beta$, where μ and β are user-defined
 435 parameters, we retrieve the query and reference coordinates of the region covered by the matches
 436 in Ψ^* . The final query and reference coordinates for a mapping between query q and reference r
 437 is computed by extending the start and end coordinates of the first and last matches in Ψ^* to the
 438 length of the query. In Supplemental Material, Algorithm 3 provides a complete description of the
 439 pseudo-chaining procedure, and Algorithm 4 describes the coordinate computation step.

440 **Proof of pseudo-chaining algorithm’s complexity.** The complexity of computing pseudo-
 441 chain Ψ^i for each read q is as follows. Let n be the total number of matches in H . To determine
 442 Ψ^* , each candidate pseudo-chain $\Psi_{q,r}$ for a single reference sequence r needs to be computed.
 443 Computing a single pseudo-chain $\Psi_{q,r}$ requires determining the match with the highest number
 444 of matching k -min-mers and comparing each match in $H_{q,r}$ to this match, which can both be
 445 performed in $\Theta(|H_{q,r}|)$ time. Moreover, every single candidate pseudo-chain (for every reference
 446 in H) needs to be computed to determine Ψ^* . Then, the running time of the pseudo-chaining
 447 procedure is $\Theta(\sum_{r \in H} |H_{q,r}|)$. Note that $|H_{q,r}| \leq n$, and the number of reference sequences that
 448 appear in H is upper bounded by the total number of reference sequences, which is $\mathcal{O}(1)$. Hence,
 449 the pseudo-chaining procedure runs in $\mathcal{O}(n)$ time, where n is the total number of matches in H .

450 Note that colinear chaining (as implemented by state-of-the-art read mappers) has an asymp-
 451 totic complexity of $\mathcal{O}(n \log n)$. We also implemented two alternative heuristics that (1) computes
 452 c pseudo-chains anchored at c matches with the highest number of k -min-mers (thus running in
 453 $\mathcal{O}(cn)$ time), and (2) sets $c = n$ and computes all possible pseudo-chains (thus running in $\mathcal{O}(n^2)$
 454 time). However, we observed that the runtime of the $\mathcal{O}(n)$ pseudo-chaining procedure is faster in
 455 practice: In our tests, the $\mathcal{O}(n)$ pseudo-chaining procedure performed $\sim 20\% - 50\%$ faster than the
 456 other heuristics, with little decrease in accuracy.

457 **Software availability**

458 The data and source code are freely available as Supplemental Material, and on GitHub at <https://github.com/ekimb/mapquik>, under the MIT License.

460 **Competing interest statement**

461 The authors declare no competing interests.

462 **Acknowledgments**

463 The authors thank Erik Garrison, Younhun Kim, Victoria Popic, and Rohit Singh for valuable
464 feedback.

465 References

- 466 Almodaresi F, Zakeri M, and Patro R. 2021. PuffAligner: A fast, efficient and accurate aligner
467 based on the Pufferfish index. *Bioinformatics*. **37**: 4048–4055.
- 468 Alser M, Rotman J, Deshpande D, Taraszka K, Shi H, Baykal PI, Yang HT, Xue V, Knyazev S,
469 Singer BD, et al. 2021. Technology dictates algorithms: Recent developments in read alignment.
470 *Genome Biology*. **22**: 1–34.
- 471 Baid G, Cook DE, Shafin K, Yun T, Llinares-López F, Berthet Q, Belyaeva A, Töpfer A, Wenger
472 AM, Rowell WJ, et al. 2022. DeepConsensus improves the accuracy of sequences with a gap-
473 aware sequence transformer. *Nature Biotechnology*. 1–7.
- 474 Bankevich A, Bzikadze AV, Kolmogorov M, Antipov D, and Pevzner PA. 2022. Multiplex de Bruijn
475 graphs enable genome assembly from long, high-fidelity reads. *Nature Biotechnology*. 1–7.
- 476 Berger B and Yu YW. 2022. Navigating bottlenecks and trade-offs in genomic data analysis. *Nature*
477 *Reviews Genetics*. 1–16.
- 478 Denti L, Khorsand P, Bonizzoni P, Hormozdiari F, and Chikhi R. 2022. Improved structural variant
479 discovery in hard-to-call regions using sample-specific string detection from accurate long reads.
480 *bioRxiv*. DOI: 10.1101/2022.02.12.480198.
- 481 Edgar R. 2020. URMAP, an ultra-fast read mapper. *PeerJ*. **8**: e9338. DOI: 10.7717/peerj.9338.
- 482 Edgar R, Taylor J, Lin V, Altman T, Barbera P, Meleshko D, Lohr D, Novakovsky G, Buchfink B,
483 Al-Shayeb B, et al. 2022. Petabase-scale sequence alignment catalyses viral discovery. *Nature*.
484 **602**: 142–147.
- 485 Ekim B, Berger B, and Chikhi R. 2021. Minimizer-space de Bruijn graphs: Whole-genome assembly
486 of long reads in minutes on a personal computer. *Cell Systems*. **12**: 958–968.
- 487 Firtina C, Park J, Alser M, Kim JS, Çalı DŞ, Shahroodi T, Ghiasi NM, Singh G, Kanellopoulos K,
488 Alkan C, et al. 2023. BLEND: A fast, memory-efficient, and accurate mechanism to find fuzzy
489 seed matches in genome analysis. *NAR Genomics and Bioinformatics*. **5**: lqad004.
- 490 Galey M, Reed P, Wenger T, Beckman E, Chang IJ, Paschal CR, Buchan JG, Lockwood CM,
491 Puia-Dumitrescu M, Garalde DR, et al. 2022. 3-hour genome sequencing and targeted analysis
492 to rapidly assess genetic risk. *medRxiv*. DOI: 10.1101/2022.09.09.22279746.

- 493 Guo L, Lau J, Ruan Z, Wei P, and Cong J. 2019. Hardware acceleration of long read pairwise
494 overlapping in genome sequencing: A race between FPGA and GPU. In *Proceedings of the 27th*
495 *Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM*
496 *2019)*, pp. 127–135.
- 497 Jain C, Dilthey A, Koren S, Aluru S, and Phillippy AM. 2017. A fast approximate algorithm
498 for mapping long reads to large reference databases. In *Proceedings of the 21st International*
499 *Conference on Research in Computational Molecular Biology (RECOMB 2017)*, pp. 66–81.
- 500 Jain C, Gibney D, and Thankachan SV. 2022a. Co-linear chaining with overlaps and gap costs.
501 In *Proceedings of the 26th International Conference on Research in Computational Molecular*
502 *Biology (RECOMB 2022)*, pp. 246–262.
- 503 Jain C, Koren S, Dilthey A, Phillippy AM, and Aluru S. 2018. A fast adaptive algorithm for
504 computing whole-genome homology maps. *Bioinformatics*. **34**: i748–i756.
- 505 Jain C, Rhie A, Hansen NF, Koren S, and Phillippy AM. 2022b. Long-read mapping to repetitive
506 reference sequences using Winnowmap2. *Nature Methods*. 1–6.
- 507 Jain C, Rhie A, Zhang H, Chu C, Walenz BP, Koren S, and Phillippy AM. 2020. Weighted minimizer
508 sampling improves long read mapping. *Bioinformatics*. **36**: i111–i118.
- 509 Kalikar S, Jain C, Vasimuddin M, and Misra S. 2022. Accelerating minimap2 for long-read sequenc-
510 ing applications on modern CPUs. *Nature Computational Science*. **2**: 78–83.
- 511 Kazemi P, Wong J, Nikolić V, Mohamadi H, Warren RL, and Birol I. 2022. ntHash2: Recursive
512 spaced seed hashing for nucleotide sequences. *Bioinformatics*. **38**: 4812–4813.
- 513 Kolmogorov M, Yuan J, Lin Y, and Pevzner PA. 2019. Assembly of long, error-prone reads using
514 repeat graphs. *Nature Biotechnology*. **37**: 540–546.
- 515 Langmead B and Salzberg SL. 2012. Fast gapped-read alignment with Bowtie2. *Nature Methods*.
516 **9**: 357–359.
- 517 Li H. 2013. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. *arXiv*.
518 DOI: 10.48550/arXiv.1303.3997.
- 519 Li H. 2016. minimap and miniasm: Fast mapping and *de novo* assembly for noisy long sequences.
520 *Bioinformatics*. **32**: 2103–2110.

- 521 Li H. 2018. minimap2: Pairwise alignment for nucleotide sequences. *Bioinformatics*. **34**: 3094–3100.
- 522 Li W, O’Neill KR, Haft DH, DiCuccio M, Chetvernin V, Badretdin A, Coulouris G, Chitsaz F,
523 Derbyshire MK, Durkin AS, et al. 2021. RefSeq: Expanding the Prokaryotic Genome Annotation
524 Pipeline reach with protein family model curation. *Nucleic Acids Research*. **49**: D1020–D1028.
- 525 Logsdon GA, Vollger MR, and Eichler EE. 2020. Long-read human genome sequencing and its
526 applications. *Nature Reviews Genetics*. 1–18.
- 527 Loose M, Malla S, and Stout M. 2016. Real-time selective sequencing using Nanopore technology.
528 *Nature Methods*. **13**: 751–754.
- 529 Marçais G and Kingsford C. 2011. A fast, lock-free approach for efficient parallel counting of
530 occurrences of k -mers. *Bioinformatics*. **27**: 764–770.
- 531 McKenna A, Hanna M, Banks E, Sivachenko A, Cibulskis K, Kernytzky A, Garimella K, Altshuler
532 D, Gabriel S, Daly M, et al. 2010. The Genome Analysis Toolkit: A MapReduce framework for
533 analyzing next-generation DNA sequencing data. *Genome Research*. **20**: 1297–1303.
- 534 Mohamadi H, Chu J, Vandervalk BP, and Birol I. 2016. ntHash: recursive nucleotide hashing.
535 *Bioinformatics*. **32**: 3492–3494.
- 536 Needleman SB and Wunsch CD. 1970. A general method applicable to the search for similarities
537 in the amino acid sequence of two proteins. *Journal of Molecular Biology*. **48**: 443–453.
- 538 Numanagić I, Gökkaya AŞ, Zhang L, Berger B, Alkan C, and Hach F. 2018. Fast characterization
539 of segmental duplications in genome assemblies. *Bioinformatics*. **34**: i706–i714.
- 540 Nurk S, Koren S, Rhie A, Rautiainen M, Bzikadze AV, Mikheenko A, Vollger MR, Altemose N,
541 Uralsky L, Gershman A, et al. 2022. The complete sequence of a human genome. *Science*. **376**:
542 44–53.
- 543 Nurk S, Walenz BP, Rhie A, Vollger MR, Logsdon GA, Grothe R, Miga KH, Eichler EE, Phillippy
544 AM, and Koren S. 2020. HiCanu: Accurate assembly of segmental duplications, satellites, and
545 allelic variants from high-fidelity long reads. *Genome Research*. **30**: 1291–1305.
- 546 Olson ND, Wagner J, McDaniel J, Stephens SH, Westreich ST, Prasanna AG, Johanson E, Boja E,
547 Maier EJ, Serang O, et al. 2022. PrecisionFDA Truth Challenge V2: Calling variants from short
548 and long reads in difficult-to-map regions. *Cell Genomics*. **2**: 100–129.

- 549 Ono Y, Asai K, and Hamada M. 2013. PBSIM: PacBio reads simulator - toward accurate genome
550 assembly. *Bioinformatics*. **29**: 119–121.
- 551 Owen MJ, Lefebvre S, Hansen C, Kunard CM, Dimmock DP, Smith LD, Scharer G, Mardach R,
552 Willis MJ, Feigenbaum A, et al. 2022. An automated 13.5 hour system for scalable diagnosis
553 and acute management guidance for genetic diseases. *Nature Communications*. **13**: 1–14.
- 554 Quinlan AR and Hall IM. 2010. BEDTools: A flexible suite of utilities for comparing genomic
555 features. *Bioinformatics*. **26**: 841–842.
- 556 Rizk G, Lavenier D, and Chikhi R. 2013. DSK: *K*-mer counting with very low memory usage.
557 *Bioinformatics*. **29**: 652–653.
- 558 Roberts M, Hayes W, Hunt BR, Mount SM, and Yorke JA. 2004. Reducing storage requirements
559 for biological sequence comparison. *Bioinformatics*. **20**: 3363–3369.
- 560 Sadasivan H, Maric M, Dawson E, Iyer V, Israeli J, and Narayanasamy S. 2023. Accelerating
561 minimap2 for accurate long read alignment on GPUs. *Journal of Biotechnology and Biomedicine*.
562 **6**: 13–23.
- 563 Şahinalp SC and Vishkin U. 1996. Efficient approximate and dynamic matching of patterns using a
564 labeling paradigm. In *Proceedings of the 37th Conference on Foundations of Computer Science*
565 (*FOCS 1996*), pp. 320–328.
- 566 Sahlin K. 2021. Effective sequence similarity detection with strobemers. *Genome Research*. **31**:
567 2080–2094.
- 568 Schatz MC. 2009. CloudBurst: Highly sensitive read mapping with MapReduce. *Bioinformatics*.
569 **25**: 1363–1369.
- 570 Schleimer S, Wilkerson DS, and Aiken A. 2003. Winnowing: local algorithms for document finger-
571 printing. In *Proceedings of the 22nd International conference on Management of Data (SIGMOD*
572 *2003*), pp. 76–85.
- 573 Shajii A, Numanagić I, Leighton AT, Greenyer H, Amarasinghe S, and Berger B. 2021. A Python-
574 based programming language for high-performance computational genomics. *Nature Biotech-*
575 *nology*. **39**: 1062–1064.

- 576 Snytsar R and Turakhia Y. 2020. Parallel approach to sliding window sums. In *Proceedings of the*
577 *20th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP*
578 *2020)*, pp. 19–26.
- 579 Yörükoğlu D, Yu YW, Peng J, and Berger B. 2016. Compressive mapping for next-generation
580 sequencing. *Nature Biotechnology*. **34**: 374–376.