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

Barış Ekim[1, 2], Kristoffer Sahlin[3], Paul Medvedev[4, 5, 6], Bonnie Berger[1, 2, *], and Rayan Chikhi[7, *]

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

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

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

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

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

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

[7]Department 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

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

Introduction

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

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

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

genome. Seeds are initial matches that serve as anchoring points of alignments: They allow a challenging instance to be split into a set of easier sub-instances by aligning only the shorter intervals in-between seeds. In the short-read era, state-of-the-art alignment algorithms (e.g., `BWA-MEM` (H Li 2013), `Bowtie2` (Langmead and Salzberg 2012), and `CORA` (Yörükoğlu et al. 2016; Shajii et al. 2021) typically relied on finding all possible seeds using a full-text index of the reference genome. For long reads, there has been a recent breakthrough by sampling and indexing only a relatively small number of short potential seeds from the reference genome, which has led to faster and more accurate mapping tools, e.g., `minimap2` (H Li 2018) and `Winnowmap2` (Jain et al. 2022b). Chaining consists of finding maximal ordered subsets of seeds that all agree on a certain genomic location (Jain et al. 2022a); seeds often have spurious matches due to their short lengths.

However, even the most recent long-read alignment tools are bottlenecks in analysis pipelines (Berger and Yu 2022). For instance, the popular `minimap2` software requires 12 CPU hours to map a typical PacBio HiFi dataset to the human genome, and `Winnowmap2` requires 15 CPU days, preventing both real-time analysis of sequencing data (Loose et al. 2016) and efficient re-analysis of previously-sequenced data collections (Edgar et al. 2022). A significant part of the `minimap2` and `Winnowmap2` running times are in their seeding and chaining steps (Kalikar et al. 2022). These state-of-the-art long-read alignment tools are sensitive and accurate, but their underlying seed constructs (k-mers) are tailored to noisy reads. These small seed sizes induce longer computation times due to the multiple potential mapping locations of seeds that need to be examined and filtered out. Recent advances in short-read alignment methods have demonstrated that 98% of many organism's genomes are non-repetitive and can be uniquely aligned to with longer seeds (Edgar 2020). Therefore, it seems natural to explore the use of longer seeds also in long reads: this idea is at the heart of our approach.

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

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

Related work

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

Other works have focused on improving the sensitivity and accuracy of `minimap2`, at the expense of speed. `Winnowmap` (Jain et al. 2020) and `Winnowmap2` (Jain et al. 2022b) use weighted minimizer sampling and minimal confidently-alignable substrings to better align in highly-repetitive regions, e.g., centromeres of chromosomes. `Winnowmap2` is around 15× slower than `minimap2` end-to-end, yet uses around 3× less memory.

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

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

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

Results

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

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

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

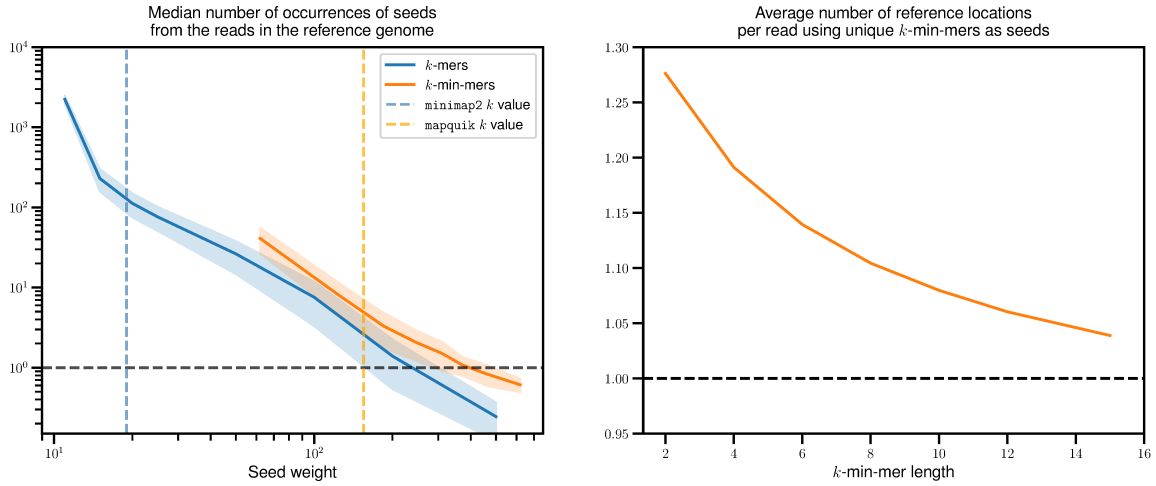

Figure 1. Increased sensitivity and specificity of k-min-mers versus long k-mers. Both panels use the human reference genome CHM13v2.0 and the HG002 DeepConsensus HiFi reads. Left panel: Each continuous line indicates the median abundance of read k-mers (darker blue line) and k-min-mers (lighter orange 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.

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

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

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

Overview of minimizer-space read mapping with `mapquik`

To allow computation in minimizer space (Figure 2), we here develop `mapquik`, a read mapper based on k-min-mer seeds. `mapquik` follows a seed-and-extend strategy used by most read mappers, with two exceptions: (1) Only the k-min-mers that appear exactly once in all reference sequences are indexed (Figure 2E-F), and (2) unlike a typical colinear chaining procedure that makes use of a dynamic programming formulation (Figure 2A-D), e.g., in `minimap2` (H Li 2018), a linear-time recursive extension step is performed for each initial k-min-mer match between the query and the reference, followed by a novel, provably linear-time step we call *pseudo-chaining* that ensures k-min-mer matches are approximately colinear. Figure 2G-I depicts the idea behind our pseudo-chaining algorithm. Given a set of maximal k-min-mer matches, we chain only the k-min-mer matches that are colinear with the match with the highest score (k-min-mer count). Unlike the DP formulations used in regular colinear chaining, this step can be performed in linear time by identifying the k-min-mer match with the highest score, and checking colinearity with the other matches through a linear scan. Note that the matches in the output chain after this step are not guaranteed to be *pairwise colinear*; however, thanks to the low number of k-min-mer matches, pseudo-chaining performs adequately in practice while offering a substantial speed-up over classical colinear chaining. The philosophy behind these drastic changes is that mapping long and accurate reads to close reference genomes is "easy enough" that long minimizer-space seeds are sufficient for the vast majority of the reads. The remaining few unmapped reads can in principle be fed to a more sensitive, albeit slower, read mapper such as `minimap2` or `Winnowmap2` (see Discussion).

Datasets and mapping evaluation

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

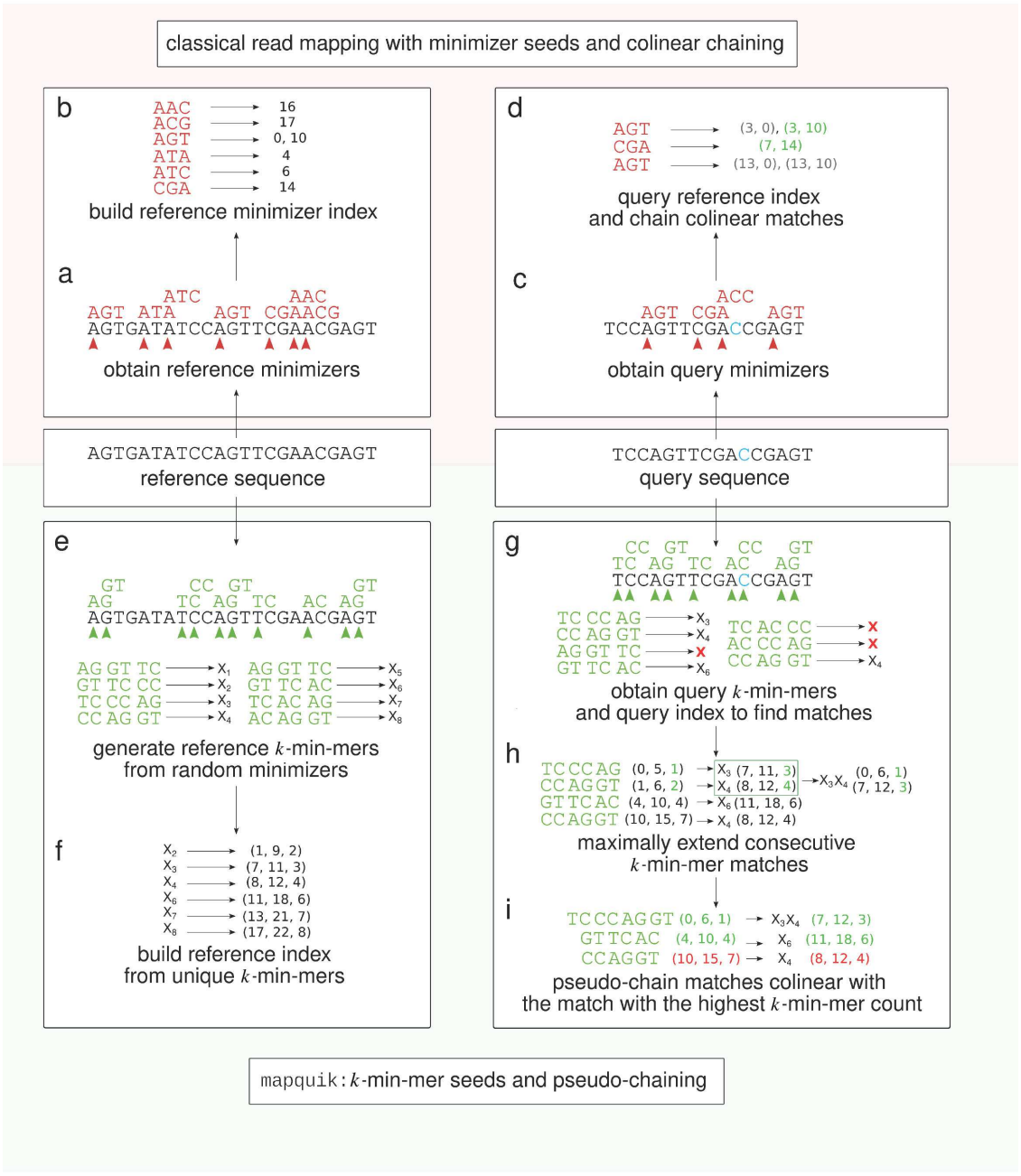

Figure 2. Overview of the long-read mapping pipeline using `mapquik` and comparison with state-of-the-art methods using minimizers as seeds. State-of-the-art read mappers such as `minimap2` and `Winnowmap2` (top, pink shaded) build an index for a reference sequence by computing window minimizers ($k = 3$, $w = 5$) (`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`).

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

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

`mapquik` achieves faster and accurate mapping of HiFi reads to the human and maize genomes

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

relative speedup over state-of-the-art mapping methods `minimap2` and `Winnowmap2`, respectively, for both simulated and real HiFi reads from the human genome. We further apply `mapquik` to mapping simulated HiFi reads from the highly-repetitive maize genome, and demonstrate a 410× speed-up over `minimap2`, with remarkably no loss of sensitivity nor accuracy.

On the simulated human dataset, `mapquik` is 54× faster than `minimap2`. `mapquik` maps 95.8% of reads at a MAPQ score of 60 (Q60), indicating a high-confidence match, with 2 errors. In contrast, other mappers map 96.1% to 98.6% of reads at Q60 with also no/almost no error. On the real dataset from HG002, a similar trend is observed. `mapquik` outperforms all mappers except `Winnowmap2` in terms of percentage of reads mapped (96.1%), as opposed to 92.2 to 97.9% for the other tools. The concordance of `minimap2` and `mapquik` mappings is 99.8% on the Q60 reads mapped by `minimap2`. In Supplemental Table S1, we provide the fraction of the reference genome covered by each tool for all experiments. All mappers required less than 14 gigabytes of memory on the human genome (`MashMap2` was not further evaluated as it took over 13 wall-clock hours on the simulated human dataset, and does not output mapping quality scores.)

A highlight of `mapquik` is a 410× mapping speed-up compared to `minimap2` on the maize genome. Remarkably, this speed-up comes with near-perfect precision for `mapquik` and no loss of sensitivity, as `mapquik` reports the second highest number of mapped reads at mapping quality 60 across all tools after `Winnowmap2`. Of note, `Winnowmap2` has faster performance on maize than the human genome and than that of `minimap2` on the maize genome.

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

Efficient genome indexing using k-min-mers

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

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

`mapquik` indexing is 9× faster than `minimap2` and 6× faster than `BLEND`. The `mapquik` seeding strategy provides ultra-fast construction of an index that records unique k-min-mer positions across the reference genome. This index is of independent interest, for example, for indexing larger databases such as RefSeq (W Li et al. 2021). We anticipate that this index will have other uses beyond long-read mapping, such as large-scale sequence search. We have already demonstrated the usefulness of performing k-min-mer search in anti-microbial resistance tracking (Ekim et al. 2021), 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)
`minimap2`	215,125,355	92%	10.1	33
`mm2-fast`	215,125,355	92%	10.1	28
`Winnowmap2`	23,616,987	41%	**2.6**	64
`BLEND`	111,799,540	97%	5.3	23
`mapquik`	39,603,738	100%	12.1	**3.4**

Table 2. Indexing a reference human genome using `mapquik` and other evaluated methods (`minimap2`, `mm2-fast`, `Winnowmap2`, `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.

Comparison with `BLEND`

While concurrently-developed, we compare to `BLEND` in Tables 1 and 2 for completeness, and demonstrate that `mapquik` achieves a 7× speed-up overall (indexing plus chaining) over `BLEND` on human data.

Robustness of mapping shorter reads with `mapquik`

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

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

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

Limitations of our study

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

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

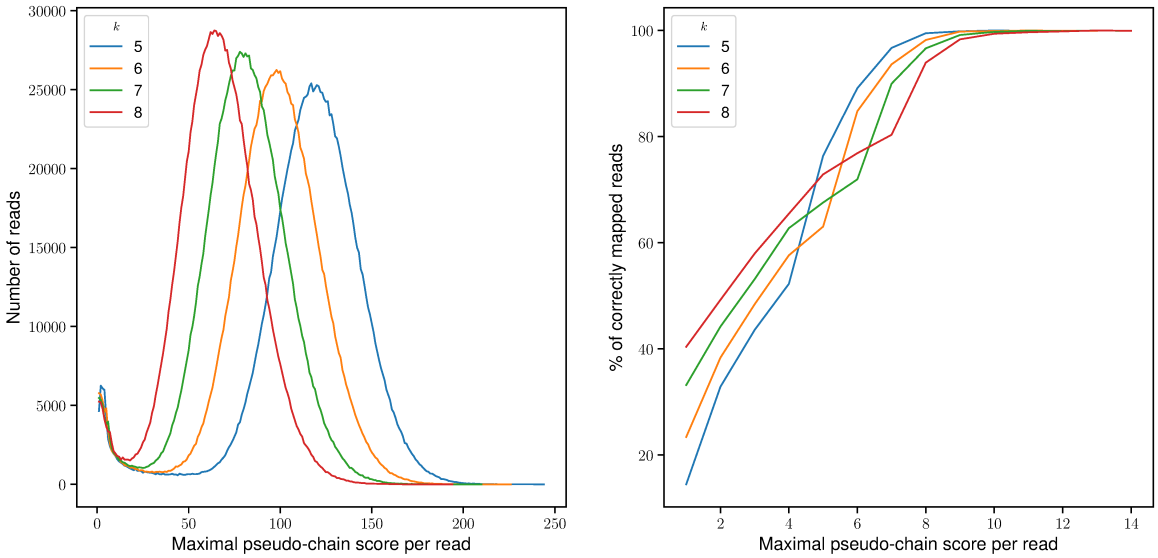

Figure 3. Effect of pseudo-chain score on mapping accuracy. The x-axis in both subfigures corresponds to the score of the maximal pseudo-chain per read; on the left, the y-axis denotes the total number of reads with the corresponding maximal pseudo-chain score, and on the right, the percentage of reads 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, $\ell = 31$, $\delta = 0.01$.

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

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

Discussion

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

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

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

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

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

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

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, ℓ, δ, ε, μ, β, f, and φ are pre-defined.

Output: Query-to-reference mappings S.

function $\textsc{Map}(R, Q)$
  $I \leftarrow \{\}$ ▷ Empty hash table of reference k-min-mers
  $L \leftarrow \{\}$ ▷ Empty hash table of sequence lengths
  $S \leftarrow \{\}$ ▷ To store mapping results
  for $r \in R$ **do**
    $X_r \leftarrow \textsc{Extract}(r)$ ▷ Extract k-min-mers from reference sequence into a list
    $L[r_{\text{ID}}] \leftarrow |r|$ ▷ Store length of r
    for every $x_r^i \in X_r$ **do**
      if φ_r^i is not a key in I **then**
        $I[\varphi_r^i] \leftarrow (r_{\text{ID}}, s_r^i, e_r^i, \pi_r^i, i_r)$ ▷ Record the reference ID r_{ID} and tuple of x_r^i
      else $I[\varphi_r^i] \leftarrow (\,)$ ▷ Assign empty entry
  for $q \in Q$ **do**
    $L[q_{\text{ID}}] \leftarrow |q|$ ▷ Store length of q
    $X_q \leftarrow \textsc{Extract}(q)$ ▷ Extract k-min-mers from query sequence into a list
    $H \leftarrow \textsc{Match}(X_q, I, q_{\text{ID}})$ ▷ Generate maximal k-min-mer matches
    $S[q] \leftarrow \textsc{PseudoChain}(H, q_{\text{ID}}, L, \varepsilon, \mu, \beta)$ ▷ Pseudo-chain generated k-min-mer matches
  return S

Methodological formalization

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

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

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

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

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

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

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

Indexing reference sequences

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

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

Informally, a *k-min-mer match* is a stretch of k-min-mer seeds which appear consecutively both in the reference and the query (under the hash function used). Note that all matches are unique, in the sense that all seed k-min-mers appear only once in the genome, by definition. Given a query q and a reference r, we formally define a *match* as a triple (i, j, c) such that $0 \leq i \leq |X_q| - c$, $0 \leq j \leq |X_r| - c$, $c \geq 1$, 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* 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 (2) either $i + c = |X_q|$, $j + c = |X_r|$, or $\varphi_q^{i+c} \neq \varphi_r^{j+c}$.

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

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

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

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

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

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

Concretely, let $h = (i, j, c)$ and $h' = (i', j', c')$ be two matches, and consider the coordinates (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 (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 and h' are *gap-bounded colinear* if

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

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

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

Let H be the list of all maximal k-min-mer matches between a read q and a reference sequence r. We define a *pseudo-chain* Ψ^i as the list of all matches in H that are colinear with the i^{th} match in H; we say that Ψ^i is *anchored at* i. Note that even though every match in Ψ^i is colinear with the i^{th} match in H, it is not necessarily true that every pair of matches in Ψ^i are pairwise colinear, 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.,

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

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

Computing high-scoring pseudo-chains in linear time

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

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

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

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

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

Software availability

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

Competing interest statement

The authors declare no competing interests.

Acknowledgments

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

References

Almodaresi F, Zakeri M, and Patro R. 2021. PuffAligner: A fast, efficient and accurate aligner based on the Pufferfish index. *Bioinformatics*. **37**: 4048–4055.

Alser M, Rotman J, Deshpande D, Taraszka K, Shi H, Baykal PI, Yang HT, Xue V, Knyazev S, Singer BD, et al. 2021. Technology dictates algorithms: Recent developments in read alignment. *Genome Biology*. **22**: 1–34.

Baid G, Cook DE, Shafin K, Yun T, Llinares-López F, Berthet Q, Belyaeva A, Töpfer A, Wenger AM, Rowell WJ, et al. 2022. DeepConsensus improves the accuracy of sequences with a gap-aware sequence transformer. *Nature Biotechnology*. 1–7.

Bankevich A, Bzikadze AV, Kolmogorov M, Antipov D, and Pevzner PA. 2022. Multiplex de Bruijn graphs enable genome assembly from long, high-fidelity reads. *Nature Biotechnology*. 1–7.

Berger B and Yu YW. 2022. Navigating bottlenecks and trade-offs in genomic data analysis. *Nature Reviews Genetics*. 1–16.

Denti L, Khorsand P, Bonizzoni P, Hormozdiari F, and Chikhi R. 2022. Improved structural variant discovery in hard-to-call regions using sample-specific string detection from accurate long reads. *bioRxiv*. DOI: 10.1101/2022.02.12.480198.

Edgar R. 2020. URMAP, an ultra-fast read mapper. *PeerJ*. **8**: e9338. DOI: 10.7717/peerj.9338.

Edgar R, Taylor J, Lin V, Altman T, Barbera P, Meleshko D, Lohr D, Novakovsky G, Buchfink B, Al-Shayeb B, et al. 2022. Petabase-scale sequence alignment catalyses viral discovery. *Nature*. **602**: 142–147.

Ekim B, Berger B, and Chikhi R. 2021. Minimizer-space de Bruijn graphs: Whole-genome assembly of long reads in minutes on a personal computer. *Cell Systems*. **12**: 958–968.

Fırtına C, Park J, Alser M, Kim JS, Çalı DŞ, Shahroodi T, Ghiasi NM, Singh G, Kanellopoulos K, Alkan C, et al. 2023. BLEND: A fast, memory-efficient, and accurate mechanism to find fuzzy seed matches in genome analysis. *NAR Genomics and Bioinformatics*. **5**: lqad004.

Galey M, Reed P, Wenger T, Beckman E, Chang IJ, Paschal CR, Buchan JG, Lockwood CM, Puia-Dumitrescu M, Garalde DR, et al. 2022. 3-hour genome sequencing and targeted analysis to rapidly assess genetic risk. *medRxiv*. DOI: 10.1101/2022.09.09.22279746.

Guo L, Lau J, Ruan Z, Wei P, and Cong J. 2019. Hardware acceleration of long read pairwise overlapping in genome sequencing: A race between FPGA and GPU. In *Proceedings of the 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM 2019)*, pp. 127–135.

Jain C, Dilthey A, Koren S, Aluru S, and Phillippy AM. 2017. A fast approximate algorithm for mapping long reads to large reference databases. In *Proceedings of the 21st International Conference on Research in Computational Molecular Biology (RECOMB 2017)*, pp. 66–81.

Jain C, Gibney D, and Thankachan SV. 2022a. Co-linear chaining with overlaps and gap costs. In *Proceedings of the 26th International Conference on Research in Computational Molecular Biology (RECOMB 2022)*, pp. 246–262.

Jain C, Koren S, Dilthey A, Phillippy AM, and Aluru S. 2018. A fast adaptive algorithm for computing whole-genome homology maps. *Bioinformatics*. **34**: i748–i756.

Jain C, Rhie A, Hansen NF, Koren S, and Phillippy AM. 2022b. Long-read mapping to repetitive reference sequences using Winnowmap2. *Nature Methods*. 1–6.

Jain C, Rhie A, Zhang H, Chu C, Walenz BP, Koren S, and Phillippy AM. 2020. Weighted minimizer sampling improves long read mapping. *Bioinformatics*. **36**: i111–i118.

Kalikar S, Jain C, Vasimuddin M, and Misra S. 2022. Accelerating minimap2 for long-read sequencing applications on modern CPUs. *Nature Computational Science*. **2**: 78–83.

Kazemi P, Wong J, Nikolić V, Mohamadi H, Warren RL, and Birol I. 2022. ntHash2: Recursive spaced seed hashing for nucleotide sequences. *Bioinformatics*. **38**: 4812–4813.

Kolmogorov M, Yuan J, Lin Y, and Pevzner PA. 2019. Assembly of long, error-prone reads using repeat graphs. *Nature Biotechnology*. **37**: 540–546.

Langmead B and Salzberg SL. 2012. Fast gapped-read alignment with Bowtie2. *Nature Methods*. **9**: 357–359.

Li H. 2013. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. *arXiv*. DOI: 10.48550/arXiv.1303.3997.

Li H. 2016. minimap and miniasm: Fast mapping and *de novo* assembly for noisy long sequences. *Bioinformatics*. **32**: 2103–2110.

Li H. 2018. minimap2: Pairwise alignment for nucleotide sequences. *Bioinformatics*. **34**: 3094–3100.

Li W, O'Neill KR, Haft DH, DiCuccio M, Chetvernin V, Badretdin A, Coulouris G, Chitsaz F, Derbyshire MK, Durkin AS, et al. 2021. RefSeq: Expanding the Prokaryotic Genome Annotation Pipeline reach with protein family model curation. *Nucleic Acids Research*. **49**: D1020–D1028.

Logsdon GA, Vollger MR, and Eichler EE. 2020. Long-read human genome sequencing and its applications. *Nature Reviews Genetics*. 1–18.

Loose M, Malla S, and Stout M. 2016. Real-time selective sequencing using Nanopore technology. *Nature Methods*. **13**: 751–754.

Marçais G and Kingsford C. 2011. A fast, lock-free approach for efficient parallel counting of occurrences of *k*-mers. *Bioinformatics*. **27**: 764–770.

McKenna A, Hanna M, Banks E, Sivachenko A, Cibulskis K, Kernytsky A, Garimella K, Altshuler D, Gabriel S, Daly M, et al. 2010. The Genome Analysis Toolkit: A MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Research*. **20**: 1297–1303.

Mohamadi H, Chu J, Vandervalk BP, and Birol I. 2016. ntHash: recursive nucleotide hashing. *Bioinformatics*. **32**: 3492–3494.

Needleman SB and Wunsch CD. 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*. **48**: 443–453.

Numanagić I, Gökkaya AŞ, Zhang L, Berger B, Alkan C, and Hach F. 2018. Fast characterization of segmental duplications in genome assemblies. *Bioinformatics*. **34**: i706–i714.

Nurk S, Koren S, Rhie A, Rautiainen M, Bzikadze AV, Mikheenko A, Vollger MR, Altemose N, Uralsky L, Gershman A, et al. 2022. The complete sequence of a human genome. *Science*. **376**: 44–53.

Nurk S, Walenz BP, Rhie A, Vollger MR, Logsdon GA, Grothe R, Miga KH, Eichler EE, Phillippy AM, and Koren S. 2020. HiCanu: Accurate assembly of segmental duplications, satellites, and allelic variants from high-fidelity long reads. *Genome Research*. **30**: 1291–1305.

Olson ND, Wagner J, McDaniel J, Stephens SH, Westreich ST, Prasanna AG, Johanson E, Boja E, Maier EJ, Serang O, et al. 2022. PrecisionFDA Truth Challenge V2: Calling variants from short and long reads in difficult-to-map regions. *Cell Genomics*. **2**: 100–129.

Ono Y, Asai K, and Hamada M. 2013. PBSIM: PacBio reads simulator - toward accurate genome assembly. *Bioinformatics*. **29**: 119–121.

Owen MJ, Lefebvre S, Hansen C, Kunard CM, Dimmock DP, Smith LD, Scharer G, Mardach R, Willis MJ, Feigenbaum A, et al. 2022. An automated 13.5 hour system for scalable diagnosis and acute management guidance for genetic diseases. *Nature Communications*. **13**: 1–14.

Quinlan AR and Hall IM. 2010. BEDTools: A flexible suite of utilities for comparing genomic features. *Bioinformatics*. **26**: 841–842.

Rizk G, Lavenier D, and Chikhi R. 2013. DSK: K-mer counting with very low memory usage. *Bioinformatics*. **29**: 652–653.

Roberts M, Hayes W, Hunt BR, Mount SM, and Yorke JA. 2004. Reducing storage requirements for biological sequence comparison. *Bioinformatics*. **20**: 3363–3369.

Sadasivan H, Maric M, Dawson E, Iyer V, Israeli J, and Narayanasamy S. 2023. Accelerating minimap2 for accurate long read alignment on GPUs. *Journal of Biotechnology and Biomedicine*. **6**: 13–23.

Şahinalp SC and Vishkin U. 1996. Efficient approximate and dynamic matching of patterns using a labeling paradigm. In *Proceedings of the 37th Conference on Foundations of Computer Science (FOCS 1996)*, pp. 320–328.

Sahlin K. 2021. Effective sequence similarity detection with strobemers. *Genome Research*. **31**: 2080–2094.

Schatz MC. 2009. CloudBurst: Highly sensitive read mapping with MapReduce. *Bioinformatics*. **25**: 1363–1369.

Schleimer S, Wilkerson DS, and Aiken A. 2003. Winnowing: local algorithms for document fingerprinting. In *Proceedings of the 22nd International conference on Management of Data (SIGMOD 2003)*, pp. 76–85.

Shajii A, Numanagić I, Leighton AT, Greenyer H, Amarasinghe S, and Berger B. 2021. A Python-based programming language for high-performance computational genomics. *Nature Biotechnology*. **39**: 1062–1064.

Snytsar R and Turakhia Y. 2020. Parallel approach to sliding window sums. In *Proceedings of the 20th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP 2020)*, pp. 19–26.

Yörükoğlu D, Yu YW, Peng J, and Berger B. 2016. Compressive mapping for next-generation sequencing. *Nature Biotechnology*. **34**: 374–376.

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