Supplemental Notes

List of Supplemental Figures

List of Supplemental Tables

S1 Algorithms

Algorithms for specific steps of `mapquik` are given here for completeness.

Algorithm 2 Initiating and extending k-min-mer matches

Input: Query k-min-mer list X_q, reference k-min-mer index I, query ID q_{ID}.
Output: Array of maximal matches H.

function MATCH(X_q, I, q_{ID})
  $H \leftarrow \{\}$ ▷ Empty hash table to store matches indexed by reference ID
  $i \leftarrow 0$
  while $i < |X_q|$ **do**
    $(\varphi_q, s_q, e_q, \pi_q, i_q) \leftarrow X_q[i]$ ▷ Retrieve metadata of ith k-min-mer in X_q
    $c \leftarrow 1$ ▷ Counter for the extension step
    if $I[\varphi_q]$ is not empty **then** ▷ If the index contains a non-empty entry for hash value φ_q of x_q^i
      $(r_{\text{ID}}, s_r, e_r, \pi_r, i_r) \leftarrow I[\varphi_q]$ ▷ Retrieve metadata of matching k-min-mer x_r^j
      $\pi \leftarrow \pi_q \oplus \pi_r$ ▷ Retrieve relative strand of x_q^i
      $h \leftarrow (q_{\text{ID}}, r_{\text{ID}}, s_q, e_q, s_r, e_r, \pi, c)$ ▷ Initialize a match with k-min-mer metadata
      $c \leftarrow$ EXTEND(h, X_q, I, i_q, c) ▷ Extend initialized k-min-mer
      Append $H[r_{\text{ID}}]$ with h ▷ Add match to hash table entry for reference r
    $i \leftarrow i + c$
  return H
function EXTEND(h, X_q, I, i_q, c)
  if $i_q = |X_q| - 1$ **then return** c ▷ Break if current k-min-mer is the last element in X_q
  $(q_{\text{ID}}, r_{\text{ID}}, s_q, e_q, s_r, e_r, \pi, c) \leftarrow h$ ▷ Retrieve metadata of current match h
  $(\varphi_q', s_q', e_q', \pi_q', i_q') \leftarrow X_q[i_q + 1]$ ▷ Retrieve metadata of next k-min-mer in X_q
  if $I[\varphi_q']$ is not empty **then** ▷ If the index has a non-empty entry for hash value of next k-min-mer
    $(r_{\text{ID}}', s_r', e_r', \pi_r', i_r') \leftarrow I[\varphi_q']$ ▷ Retrieve metadata of matching k-min-mer x_r'
    if $r_{\text{ID}}' = r_{\text{ID}}$ **and** $\pi = \pi_q' \oplus \pi_r'$ **then** ▷ If the next k-min-mer is on the same reference and strand
      if ($\pi = 0$ **and** $i_r' = i_r + 1$) **or** ($\pi = 1$ **and** $i_r' = i_r - 1$ **then**) ▷ If the next k-min-mer matches
        $c \leftarrow c + 1$ ▷ Increase counter
        if $\pi = 0$ **then**
          $h \leftarrow (q_{\text{ID}}, r_{\text{ID}}, s_q, e_q', s_r, e_r', \pi, c)$ ▷ Extend current match
        else
          $h \leftarrow (q_{\text{ID}}, r_{\text{ID}}, s_q, e_q', s_r', e_r, \pi, c)$
        return EXTEND(h, X_q, I, i_q', c) ▷ Recursively extend current match
  return c

Algorithm 3 Pseudo-chaining k-min-mer matches

Input: Index of maximal k-min-mer matches H, query ID q_{ID}, sequence length index L, parameters ε, μ, β.

Output: A mapping $(q_{\text{ID}}, r_{\text{ID}}, x_q, y_q, x_r, y_r, c)$.

function PSEUDOCHAIN($H, q_{\text{ID}}, L, \varepsilon, \mu, \beta$)
    $\Psi^* = [\,]$ ▷ Highest-scoring maximal pseudo-chain among all possible references
    for $(r'_{\text{ID}}, H_{q,r'}) \in H$ **do** ▷ Iterate over the matches for each reference
        $\Psi_{q,r'} \leftarrow$ HSPSEUDOCHAIN($H_{q,r'}, \varepsilon$) ▷ Find candidate highest-scoring pseudo-chain
        if SCORE($\Psi_{q,r'}$) > SCORE(Ψ^*) **then** $\Psi^* \leftarrow \Psi_{q,r'}$ ▷ Update highest-scoring pseudo-chain
    if SCORE(Ψ^*) $\geq \mu$ **or** $|\Psi^*| \geq \beta$ **then** ▷ If the highest-scoring pseudo-chain has score $\geq \mu$ or has length $\geq \beta$
        $(q_{\text{ID}}, r_{\text{ID}}, s_q, e_q, s_r, e_r, \pi, c) \leftarrow \Psi^*[0]$ ▷ First match in highest-scoring pseudo-chain
        $(q_{\text{ID}}, r_{\text{ID}}, s'_q, e'_q, s'_r, e'_r, \pi, c') \leftarrow \Psi^*[-1]$ ▷ Last match in highest-scoring pseudo-chain
        if $\pi = 0$ **then return** COORDINATES($q_{\text{ID}}, r_{\text{ID}}, s_q, e'_q, s_r, e'_r, \pi,$ SCORE(Ψ^*), L)
        else return COORDINATES($q_{\text{ID}}, r_{\text{ID}}, s_q, e'_q, s'_r, e_r, \pi,$ SCORE(Ψ^*), L)

function HSPSEUDOCHAIN($H_{q,r}, \varepsilon$)
    $\Psi^* \leftarrow [\,]$
    $h^* \leftarrow \arg\max_{h \in H_{q,r}} c(h)$ ▷ Retrieve match with maximum k-min-mer count
    for $h \in H_{q,r}$ **do** ▷ Iterate over the matches in $H_{q,r}$
        if ISCOLINEAR(h^*, h, ε) **then** $\Psi^* \leftarrow \Psi^* + [h']$ ▷ Extend pseudo-chain by appending h' if colinear
    return Ψ^*

function ISCOLINEAR(h, h', ε)
    $(x_q, y_q, x_r, y_r, \pi) \leftarrow h$ ▷ Retrieve metadata of match h
    $(x'_q, y'_q, x'_r, y'_r, \pi') \leftarrow h'$ ▷ Retrieve metadata of match h'
    if $h = h'$ **then return** True ▷ Return true for identical matches
    if $\pi \neq \pi'$ **then return** False ▷ Return false for matches in opposite direction
    if $\pi = 0$ **then** ▷ If query is in the forward direction
        $g_q \leftarrow |(x'_q - y_q)|$ ▷ Gap length between h and h' on the query
        $g_r \leftarrow |(x'_r - y_r)|$ ▷ Gap length between h and h' on the reference
        if $x_r < x'_r$ **and** $|g_q - g_r| < \varepsilon$ **then return** True ▷ Check gap length difference and match order
    else if $\pi = 1$ **then** ▷ If query is in the reverse direction
        $g_q \leftarrow |(x'_q - y_q)|$ ▷ Gap length between h and h' on the query
        $g_r \leftarrow |(x_r - y'_r)|$ ▷ Gap length between h and h' on the reference
        if $x'_r < x_r$ **and** $|g_q - g_r| < \varepsilon$ **then return** True ▷ Check gap length difference and match order
    return False

Algorithm 4 Obtaining final mapping coordinates

Input: Query ID q_{ID}, reference ID r_{ID}, maximal pseudo-chain query positions x_q, y_q, maximal pseudo-chain reference positions x_r, y_r, relative strand π, maximal pseudo-chain score c, sequence length index L.

Output: Final mapping $(q_{\text{ID}}, r_{\text{ID}}, x_q, y_q, x_r, y_r, \pi, c)$.

function COORDINATES($q_{\text{ID}}, r_{\text{ID}}, x_q, y_q, x_r, y_r, \pi, c, L$)
  $z_x \leftarrow 0$
  $z_y \leftarrow 0$
  if $\pi = 0$ **then**
    if $x_r \geq x_q$ **then**
      $z_x \leftarrow x_q$
      $x_r \leftarrow x_r - x_q$
    else
      $z_x \leftarrow x_r$
      $x_r \leftarrow 0$
    if $y_r + (L[q_{\text{ID}}] - y_q - 1) \leq L[r_{\text{ID}}] - 1$ **then**
      $z_y \leftarrow (L[q_{\text{ID}}] - y_q - 1)$
      $y_r \leftarrow y_r + (L[q_{\text{ID}}] - y_q - 1)$
    else
      $z_y \leftarrow (L[r_{\text{ID}}] - y_r - 1)$
      $y_r \leftarrow L[r_{\text{ID}}] - 1$
  if $\pi = 1$ **then**
    if $y_r + q_s \leq L[r_{\text{ID}}] - 1$ **then**
      $y_r \leftarrow y_r + q_s$
      $z_x \leftarrow q_s$
    else
      $z_x \leftarrow (L[r_{\text{ID}}] - y_r - 1)$
      $y_r \leftarrow L[r_{\text{ID}}] - 1$
    if $x_r \geq (L[q_{\text{ID}}] - y_q - 1)$ **then**
      $x_r \leftarrow x_r - (L[q_{\text{ID}}] - y_q - 1)$
      $z_y \leftarrow L[q_{\text{ID}}] - y_q - 1$
    else
      $z_y \leftarrow x_r$
      $x_r \leftarrow 0$
  $x_q \leftarrow x_q + z_x$
  $y_q \leftarrow y_q + z_y$
return $(q_{\text{ID}}, r_{\text{ID}}, x_q, y_q, x_r, y_r, \pi, c)$

S2 Command lines and versions

Simulated human HiFi reads were simulated using `pbsim` at 10× coverage depth with the following command lines:

```
ref=chm13v2.0.fa
reads=simulated-chm13v2.0-10X
pbsim $ref --model_qc PBSIM-PacBio-Simulator/data/model_qc_clr --accuracy-mean 0.99 \
           --accuracy-sd 0 --depth 10 --prefix $reads --length-mean 24000
paftools.js pbsim2fq $ref.fai "$reads"_*.maf > $reads.fa
```

Mappers were run using the following command lines:

```
mapquik $reads.fa --reference chm13v2.0.fa --threads 10
minimap2 -x map-hifi -t 10 chm13v2.0.fa $reads.fa
mm2-fast -x map-hifi -t 10 chm13v2.0.fa $reads.fa
blend -t 10 -x map-hifi chm13v2.0.fa $reads.fa
winnowmap -t 10 -W repetitive_k15.txt -x map-pb chm13v2.0.fa $reads.fa
```

Tool versions:

```
minimap 2.24-r1122
mm2-fast 2.24-r1122
BLEND 1.0
winnowmap 2.03
mapquik 0.1.0.
```

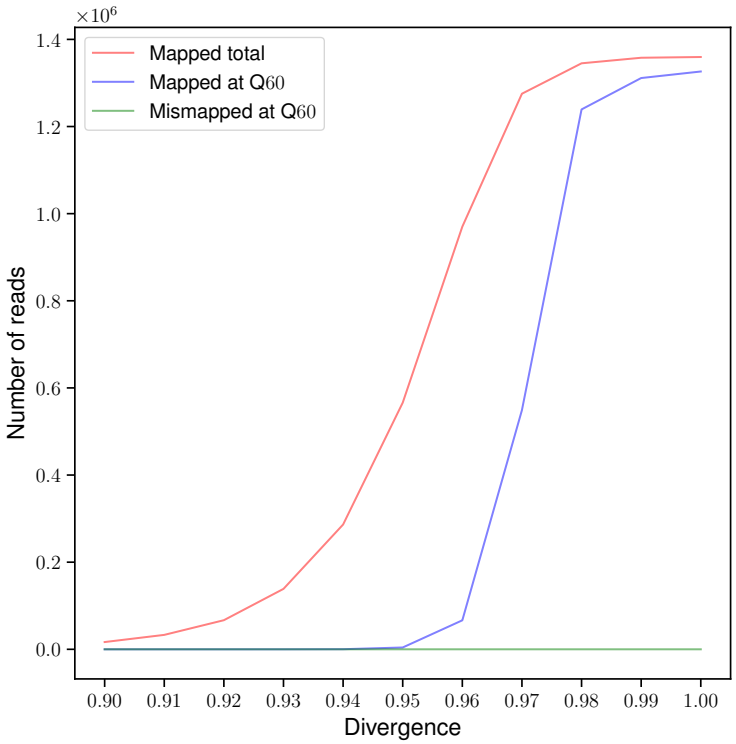
S3 Complete simulated mapping evaluations

In this section, we report the complete results of `paftools mapeval` evaluations on the simulated 10× coverage human genome experiment from Table 1, including the reads that were mapped at MAPQ below 60. Column 2 is the mapping quality, column 3 is the number of mapped reads at this quality, and column 4 is the number of erroneously mapped reads at this quality. Column 5 is the cumulative fraction of erroneously mapped reads, and column 6 is the cumulative number of mapped reads.

```
$ paftools.js mapeval mapquik-sim10X.paf
Q       60      1310808 2       0.000001526     1310808
Q       0       49307   24829   0.018256544     1360115
$ paftools.js mapeval winnowmap-sim10X.paf
Q       60      1350016 6       0.000004444     1350016
Q       58      195     2       0.000005925     1350211
Q       56      159     3       0.000008146     1350370
Q       54      193     1       0.000008885     1350563
Q       52      106     1       0.000009625     1350669
Q       51      221     1       0.000010364     1350890
Q       49      108     2       0.000011843     1350998
Q       48      126     1       0.000012582     1351124
Q       47      134     2       0.000014061     1351258
Q       46      232     3       0.000016278     1351490
Q       44      144     2       0.000017756     1351634
Q       43      121     2       0.000019234     1351755
Q       42      146     2       0.000020712     1351901
Q       41      117     1       0.000021449     1352018
Q       40      110     2       0.000022927     1352128
Q       39      226     1       0.000023662     1352354
Q       37      108     3       0.000025879     1352462
Q       36      102     1       0.000026616     1352564
Q       35      125     3       0.000028831     1352689
Q       34      146     3       0.000031046     1352835
Q       33      146     4       0.000033999     1352981
Q       32      157     3       0.000036212     1353138
Q       31      180     2       0.000037685     1353318
Q       30      169     4       0.000040636     1353487
Q       29      191     3       0.000042846     1353678
Q       28      171     5       0.000046534     1353849
Q       27      180     6       0.000050959     1354029
Q       26      161     7       0.000056122     1354190
Q       25      156     10      0.000063499     1354346
Q       24      136     3       0.000065708     1354482
Q       23      156     6       0.000070129     1354638
Q       22      169     9       0.000076764     1354807
```

```
Q       21      193     11      0.000084871     1355000
Q       20      172     3       0.000087074     1355172
Q       19      194     14      0.000097391     1355366
Q       18      207     8       0.000103277     1355573
Q       17      269     15      0.000114320     1355842
Q       16      286     11      0.000122407     1356128
Q       15      269     17      0.000134916     1356397
Q       14      264     23      0.000151843     1356661
Q       13      213     13      0.000161400     1356874
Q       12      232     15      0.000172426     1357106
Q       11      178     21      0.000187875     1357284
Q       10      186     16      0.000199636     1357470
Q       9       217     21      0.000215072     1357687
Q       8       221     24      0.000232711     1357908
Q       7       227     30      0.000254761     1358135
Q       6       239     32      0.000278274     1358374
Q       5       207     20      0.000292953     1358581
Q       4       176     29      0.000314258     1358757
Q       3       167     32      0.000337767     1358924
Q       2       215     22      0.000353901     1359139
Q       1       481     25      0.000372163     1359620
Q       0       9192    4766    0.003851515     1368812
$ paftools.js mapeval minimap-sim10X.paf
Q       60      1340993 0       0.000000000     1340993
Q       2       4185    1       0.000000743     1345178
Q       1       6326    5       0.000004439     1351504
Q       0       17308   5211    0.003811334     1368812
$ paftools.js mapeval mm2-fast-sim10X.paf
Q       60      1340993 0       0.000000000     1340993
Q       2       4185    1       0.000000743     1345178
Q       1       6326    5       0.000004439     1351504
Q       0       17308   5211    0.003811334     1368812
$ paftools.js mapeval blend-sim10X.paf
Q       60      1315676 1       0.000000760     1315676
Q       48      1471    1       0.000001518     1317147
Q       46      2880    1       0.000002273     1320027
Q       42      724     1       0.000003029     1320751
Q       41      1496    1       0.000003781     1322247
Q       39      676     1       0.000004535     1322923
Q       38      777     1       0.000005288     1323700
Q       37      740     3       0.000007550     1324440
Q       36      754     1       0.000008301     1325194
Q       35      739     1       0.000009050     1325933
Q       34      782     5       0.000012814     1326715
Q       33      737     2       0.000014313     1327452
```

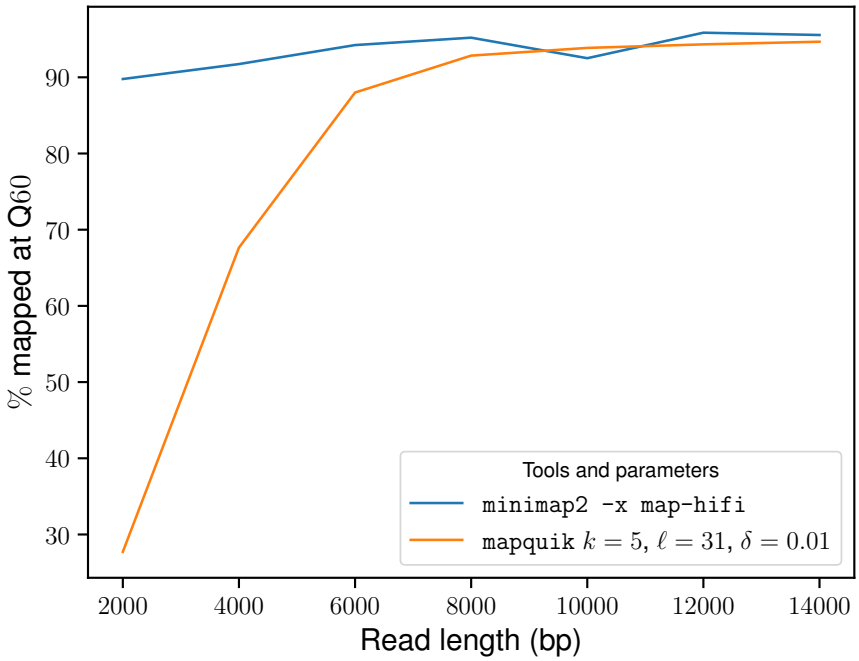
708	Q	32	795	2	0.000015810	1328247
709	Q	31	741	3	0.000018059	1328988
710	Q	30	696	5	0.000021810	1329684
711	Q	29	793	2	0.000023300	1330477
712	Q	28	743	5	0.000027043	1331220
713	Q	27	768	6	0.000031532	1331988
714	Q	26	770	7	0.000036766	1332758
715	Q	25	792	10	0.000044243	1333550
716	Q	24	779	12	0.000053210	1334329
717	Q	23	804	11	0.000061417	1335133
718	Q	22	753	11	0.000069617	1335886
719	Q	21	826	13	0.000079299	1336712
720	Q	20	773	15	0.000090468	1337485
721	Q	19	787	18	0.000103865	1338272
722	Q	18	830	14	0.000114256	1339102
723	Q	17	854	19	0.000128362	1339956
724	Q	16	844	31	0.000151402	1340800
725	Q	15	875	29	0.000172918	1341675
726	Q	14	854	16	0.000184726	1342529
727	Q	13	867	40	0.000214382	1343396
728	Q	12	928	45	0.000247708	1344324
729	Q	11	927	36	0.000274298	1345251
730	Q	10	962	36	0.000300844	1346213
731	Q	9	1026	61	0.000345893	1347239
732	Q	8	961	59	0.000389408	1348200
733	Q	7	1044	75	0.000444693	1349244
734	Q	6	1151	79	0.000502816	1350395
735	Q	5	1148	90	0.000568979	1351543
736	Q	4	1256	104	0.000645329	1352799
737	Q	3	1241	133	0.000742962	1354040
738	Q	2	1350	137	0.000843300	1355390
739	Q	1	1193	148	0.000951656	1356583
740	Q	0	12229	6637	0.005791884	1368812



Supplemental Figure S1. Degradation of performance of `mapquik` with higher divergences between reads and reference. The "Mapped total" line corresponds to the total number of mapped reads by `mapquik`. The "Mapped Q60" line corresponds to the number of reads mapped at a MAPQ of 60. The "Wrong Q60" line corresponds to wrongly mapped reads as assessed by `paftools mapeval`.

S4 Analysis of divergence

Supplemental Figure S1 shows the total number of mapped reads, the number of mapped reads at mapping quality Q60, and the number of wrongly mapped reads at Q60 of `mapquik`, when mapping simulated CHM13v2.0 reads at 10× coverage with varying levels of sequencing errors (from perfect to 10% error rate) to the CHM13v2.0 reference. For all levels, `mapquik` was run with $k = 5$, $l = 31$, $\delta = 0.01$. Intuitively, lowering the value of k for higher divergences would further increase sensitivity. Hence, this analysis provides a pessimistic estimation of `mapquik` robustness.



Supplemental Figure S2. Robustness of `mapquik` and `minimap2` with shorter read lengths. The proportion of reads mapped at Q60 is reported for both mappers, in samples containing simulated human 10× coverage reads of lengths ranging from 2 kbp to 14 kbp. `minimap2` was run with default HiFi parameters, and `mapquik` was run with the same parameters as those used to map the 24 kbp reads.

	`minimap2`		`mm2-fast`		`Winnowmap2`		`BLEND`		**`mapquik`**	
Size (Gbp)	Gbp	%	Gbp	%	Gbp	%	Gbp	%	Gbp	%
CHM13 10× coverage **simulated** 24 kbp HiFi reads										
3.117	2.861	91.77	2.861	91.77	3.092	99.19	3.026	97.08	3.033	97.31
HG002 30× coverage **real** 24 kbp HiFi reads (DeepConsensus)										
3.117	2.856	91.62	2.856	91.62	3.057	98.08	2.976	95.48	2.989	95.89
Maize 30× coverage **simulated** 24 kbp HiFi reads										
2.182	2.131	97.64	2.131	97.64	2.172	99.55	2.166	99.23	2.167	99.26

Supplemental Table S1. Mapping coverage statistics of `mapquik` and other evaluated methods (`minimap2`, `mm2-fast`, `Winnowmap2`, `BLEND`) on simulated and real HiFi reads, and simulated maize HiFi reads. Only reads with mapping quality of 60 were included. Coverage information was obtained using `BEDTools coverage` on the mapping output. The "Size" column indicates the reference genome size (in Gbp). The "Gbp" columns indicate the number of base pairs covered (in Gbp) on the reference genome, and the "%" columns indicate the percentage of the reference genome covered.

S5 Hash functions and minimizers implementation details

We use the `ntHash1` (Mohamadi et al. 2016) method for hashing ℓ-mers, as implemented in `https://github.com/luizirber/nthash`. Minimizers are computed as universe minimizers (Ekim et al. 2021), in implicit homopolymer-compressed space, i.e., all sequences are homopolymer-compressed (HPC) during minimizer computation, but positions of the HPC minimizers are reported in the original unmodified sequence.

Previous works have examined SIMD acceleration of (1) `ntHash2` (Kazemi et al. 2022) using AVX2 and AVX512 SIMD instructions (see `ntHash` GitHub repository pull request #9) and (2) windowed minimizer computation (Snytsar and Turakhia 2020), where speed-ups of $3-5\times$ were reported. The `mapquik` codebase implements several flavors of `ntHash` that can be of independent interest: A scalar (`ntHash1`) using 32- or 64-bit hashes, as well as an AVX512 implementation of `ntHash2` using 32-bit hashes. The latter ended up not being used in our tests due to the lack of an efficient compatibility with homopolymer-compressed input.

S6 Additional remarks on Figure 3

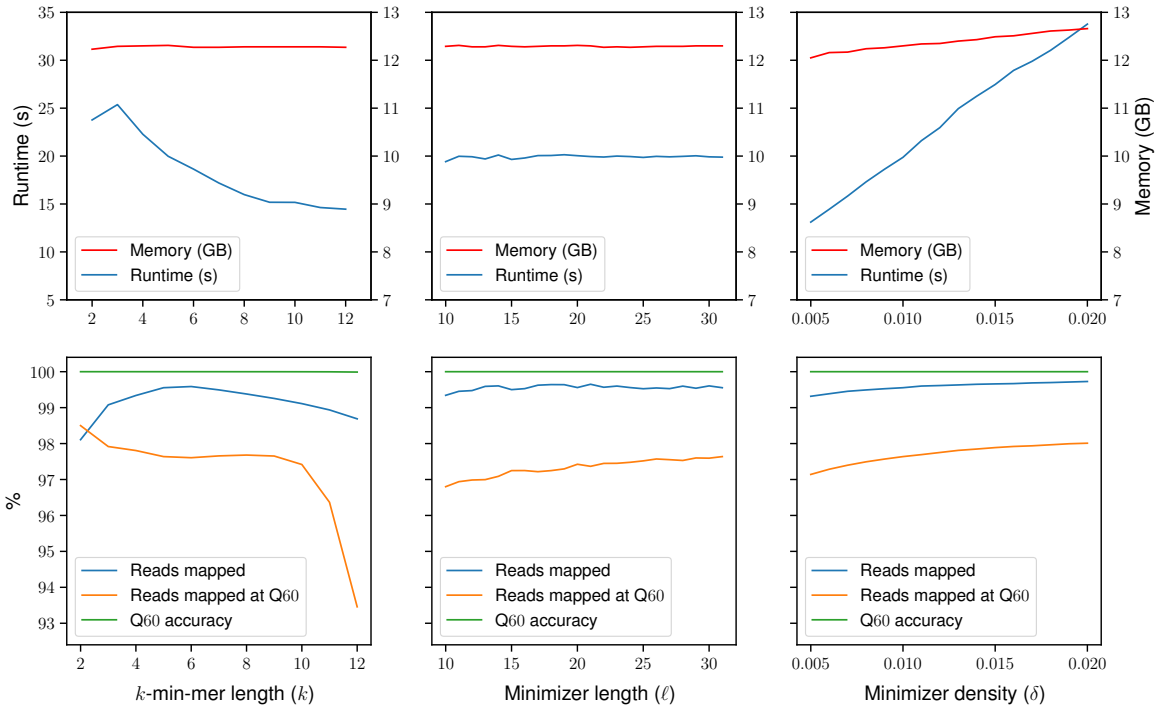
In the left panel of Figure 3, note that for all values of k shown, the distribution resembles that of a k-mer histogram: It follows a pseudo-normal distribution, except for the smaller peak on the left side of the curve that corresponds to erroneous pseudo-chains, which occur either due to sequencing errors in the reads, or due to hash collisions. Even though this peak is considerably smaller than that in a k-mer histogram (due to sampling minimizers, which may or may not have errors), we still observe a similar effect as the peak that corresponds to erroneous k-mers in a k-mer histogram: Similar to filtering out erroneous k-mers by imposing a threshold on the frequency of a k-mer (only selecting "solid" k-mers, for example, which have a count of > 1), we can filter out erroneous pseudo-chains by imposing a threshold on the score of each pseudo-chain.

S7 Additional remarks on maximality of k-min-mer matches

Here, we show that any output k-min-mer match is both left- and right-maximal. Since each k-min-mer match is extended maximally towards the right, each output match is right-maximal by definition. It's straightforward to show by induction that any output match is also a left-maximal match:

- For a match at position 0, left-maximality is trivial (no previous k-min-mer).
- Assume that the match at iteration i is right-maximal. Then, the k-min-mer after the match is not compatible with m, since m can't be extended further.
- If a match m' then starts from this k-min-mer in iteration $i+1$, it must be that m' is left-maximal, since otherwise m could have been extended with this k-min-mer in iteration i.

Then, any output match is also left-maximal, and hence maximal.



Supplemental Figure S3. Runtime and memory usage of `mapquik` with respect to different values of k, ℓ, and δ. All panels use the human reference genome CHM13v2.0 and the 10× simulated read set. The `mapquik` parameter ranges are $k = 2$ to 12, $\ell = 10$ to 31, and $\delta = 0.005$ to 0.02. The top row of panels show the runtime in seconds (red, y-axis, left) and memory usage (blue, y-axis, right), and the bottom row of panels show the percentage of reads mapped (blue), reads mapped at a MAPQ of 60 (orange), and Q60 accuracy (green) of `mapquik`, for different values of k (left), ℓ (center), and δ (right).

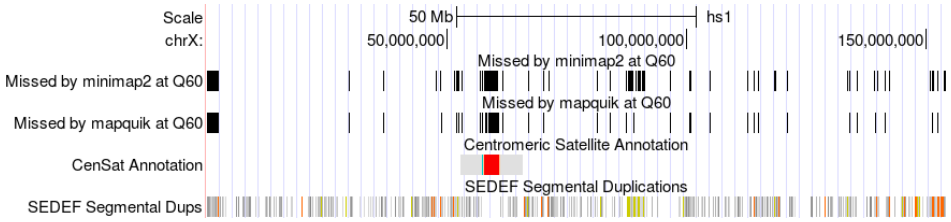
S8 Effect on `mapquik` parameters on runtime and accuracy

In this section, we explore the impact of different values of the input parameters k (k-min-mer length), ℓ (minimizer length), and δ (minimizer density) on `mapquik`'s runtime, memory use, and mapping accuracy using the CHM13v2.0 reference and the 10× simulated read set (evaluated by `paftools mapeval`).

As seen the top row of panels of Supplemental Figure S3, `mapquik` runtime is primarily impacted by the value of k (k-min-mer length) and δ (minimizer density). The runtime decreases as k increases, since there are fewer k-min-mer matches between the query and the reference. The runtime increases as δ increases, since more minimizers are selected, which increases the number of k-min-mers obtained. The maximum memory usage of `mapquik`, however, is not impacted by any

variation in the parameters, since the bottleneck in terms of memory usage is the parsing of the reference genome sequence, before any minimizers or k-min-mers are obtained.

The bottom row of panels of Supplemental Figure S3 shows that both the accuracy of `mapquik` at Q60 and the percentage of total reads mapped are generally not impacted by the varying parameters; the percentage of total reads mapped are slightly higher at $k = 5$ (the default value of k) than at any other value. The percentage of reads mapped at a MAPQ of 60, however, decreases rapidly at $k = 10$, due to fewer number of matches found between the query and the reference which satisfies the minimum chain score μ and chain length β, and are not assigned a MAPQ of 60. Apart from the effects of a large k-min-mer length on the percentage of reads mapped at Q60 and a large value of δ on runtime, `mapquik` appears to be robust for a wide set of ranges for all three input parameters.



Supplemental Figure S4. Missed genomic regions by `minimap2` and `mapquik` at Q60 on the CHM13 chromosome X. The missed regions were obtained by mapping the HG002 DeepConsensus HiFi reads to the human reference genome CHM13v2.0, and calling `BEDTools complement` and `BEDTools intersect` on the output. The "CenSat Annotation" track denotes the centromeric and pericentromeric satellite annotations on the CHM13 reference. The "`SEDEF` Segmental Dups" track denotes segmental duplications as generated by `SEDEF` (Numanagić et al. 2018) on the CHM13 reference. The visualization was generated using the UCSC Genome Browser.

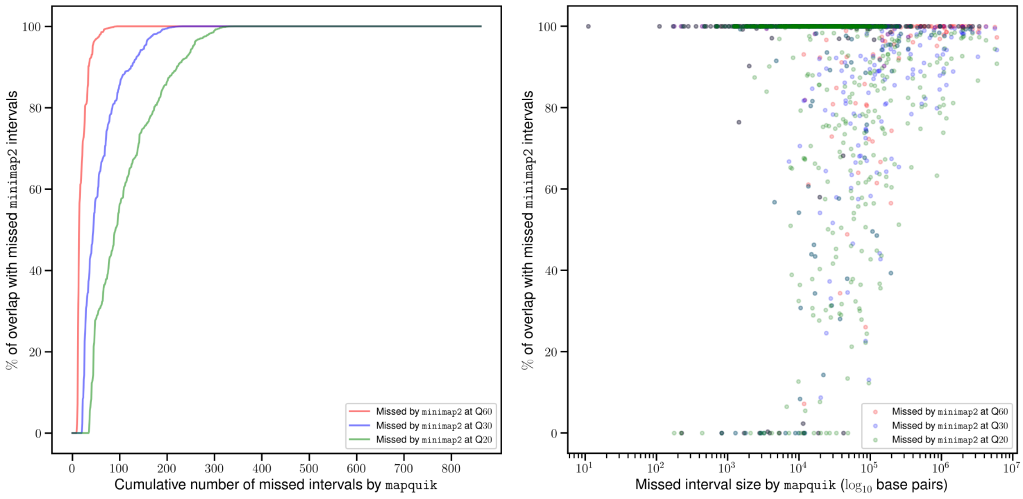
S9 Investigating genomic regions missed by `mapquik`

In order to check concordance between `minimap2` and `mapquik` on the real HG002 HiFi reads, we obtained the missed genomic regions for both mappers using `BEDTools complement` on the mapping output, and intersected the missed intervals using `BEDTools intersect`.

Supplemental Figure S4 shows a UCSC Genome Browser visualization of the genomic regions missed both by `minimap2` and `mapquik` a MAPQ of 60 on chromosome X when mapping the HG002 DeepConsensus HiFi reads to the CHM13 reference, along with the centromeric and pericentromeric satellite annotations and segmental duplications generated by `SEDEF` (Numanagić et al. 2018) on CHM13. Both `minimap2` and `mapquik` generally fail to map to the same regions across the chromosome, e.g., centromeric satellites and large segmental duplications; however, `mapquik` covers a larger fraction of the challenging regions (as shown in Table 1 and Supplemental Table S1).

Supplemental Figure S5 shows the percent overlap between the intervals missed by `minimap2` at MAPQ scores of 60 (red), 30 (blue), and 20 (green) and `mapquik`, for both the cumulative number of intervals missed by `mapquik` (left) and the respective sizes (in $\log_{10}$ base pairs) of the missed intervals by `mapquik` (right).

In the left panel in Supplemental Figure S5, the overlap percentage between the missed `mapquik` intervals and the missed `minimap2` intervals at Q60 converges to 100% at < 100, which shows that only less than 100 intervals missed by `mapquik` have an overlap of $< 100\%$ with the missed `minimap2`



Supplemental Figure S5. Percentage of overlap between the missed genomic regions by `minimap2` and `mapquik` by number and size of intervals. Both panels use the human reference genome CHM13v2.0 and the HG002 DeepConsensus HiFi reads. The percentage of overlap was computed by obtaining the missed intervals using `BEDTools complement` on the mapping output, and `BEDTools intersect` on the missed regions. Left panel: Percent overlap (y-axis) between the missed genomic intervals by `mapquik` and `minimap2` (at Q60 in red, Q30 in blue, and Q20 in green) for the cumulative number of missed intervals by `mapquik` (x-axis). Right panel: Percent overlap (y-axis) between the missed genomic intervals by `mapquik` and `minimap2` (at Q60 in red, Q30 in blue, and Q20 in green) per interval size in $\log_{10}$ base pairs (x-axis).

intervals at a MAPQ of 60. For MAPQ values of 30 and 20, the number of missed intervals without a 100% overlap between `minimap2` and `mapquik` increases (to $\sim$ 200 and $\sim$ 300, respectively), since `minimap2` additionally maps more locations at lower MAPQ scores. However, for all MAPQ thresholds, the number of `mapquik` intervals with less than 25% overlap between `minimap2` intervals are less than 50 (out of a total number of $\sim$ 900 intervals), which shows that a large fraction of missed intervals by `mapquik` corresponds to the missed intervals by `minimap2`.

In the right panel, Supplemental Figure S5 shows the overlap percentage between the missed `mapquik` intervals and the missed `minimap2` intervals per interval size, at the same MAPQ values of 60 (red), 30 (blue), and 20 (green). For the missed `mapquik` intervals without a 100% overlap percentage with the missed `minimap2` intervals (across all MAPQ values), the median size is around 1 to 10 kbp. Additionally, intervals with sizes of $>$ 1 Mbp across all MAPQ values have at least a 60% overlap with the missed `minimap2` intervals (with $>$ 80% for Q30 and $>$ 90% for Q60), which suggests that the intervals that `mapquik` additionally misses with respect to `minimap2` are small.