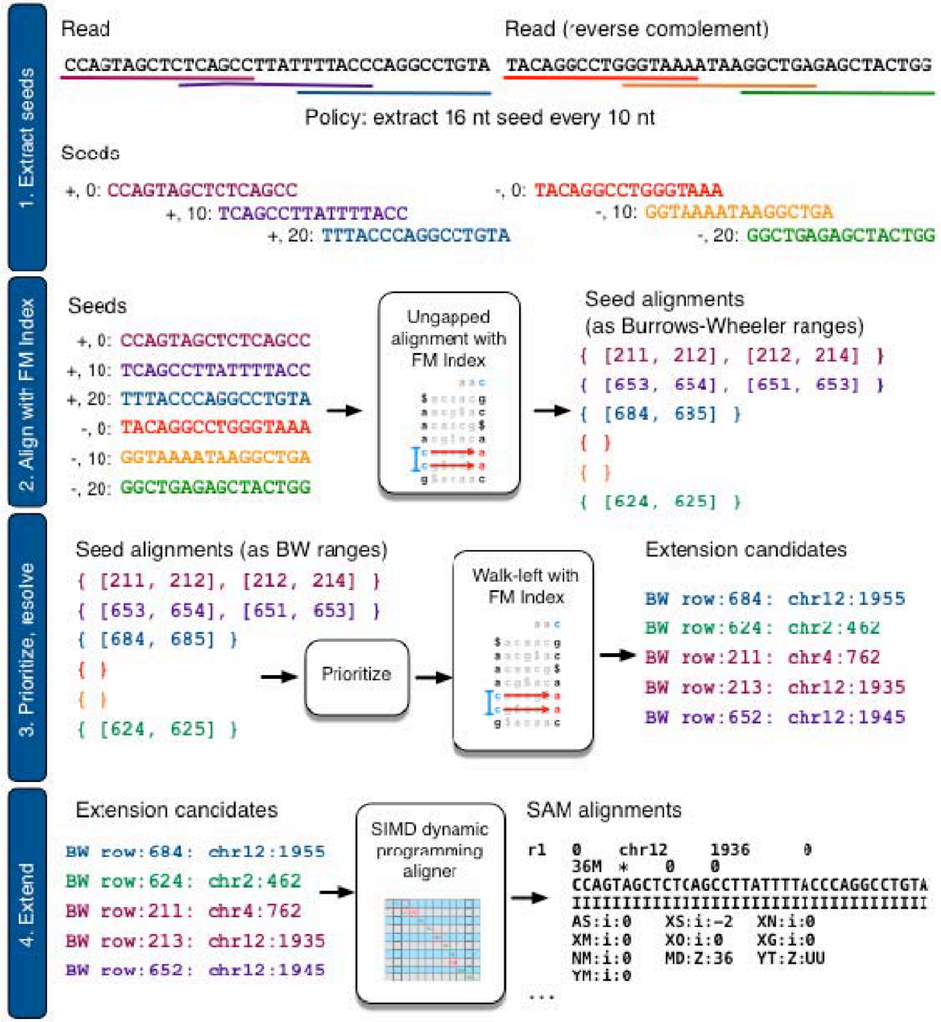
HASH-BASED INDEXES, K-MERS AND MINIMIZERS

K-mer based seeding

Fast gapped-read alignment with Bowtie 2

Ben Langmead[1,2] & Steven L Salzberg[1–3]

As the rate of sequencing increases, greater throughput is demanded from read aligners. The full-text minute index is often used to make alignment very fast and memory-efficient, but the approach is ill-suited to finding longer, gapped alignments. Bowtie 2 combines the strengths of the full-text minute index with the flexibility and speed of hardware-accelerated dynamic programming algorithms to achieve a combination of high speed, sensitivity and accuracy.



K-mer based seeding

- Bowtie 2 extracts seeds from evenly-spaced positions across the read
- The “stride” is a user-definable (runtime) parameter, as is the seed length
  - How is this the case?

K-mer based seeding

- Bowtie 2 extracts seeds from evenly-spaced positions across the read
- The “stride” is a user-definable (runtime) parameter, as is the seed length
  - How is this the case? (Because **all** substrings are indexed — Bowtie2 uses the FM-index)
- What other kinds of index might we use?

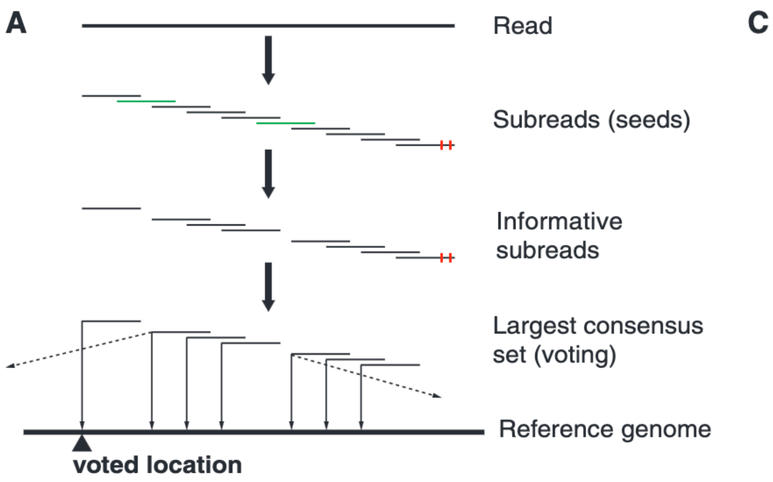
K-mer based seeding

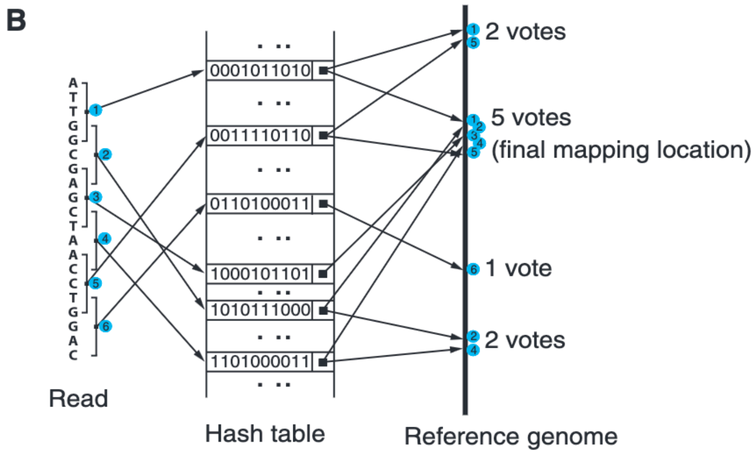
The Subread aligner: fast, accurate and scalable read mapping by seed-and-vote

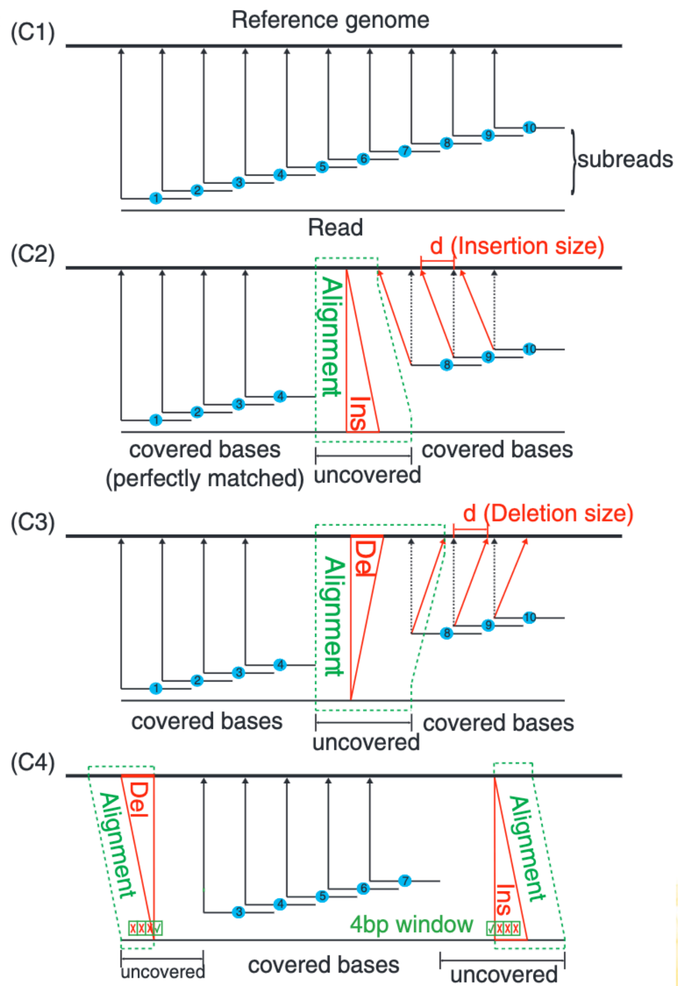
Yang Liao[1,2], Gordon K. Smyth[1,3] and Wei Shi[1,2,*]

[1]Division of Bioinformatics, The Walter and Eliza Hall Institute of Medical Research, 1G Royal Parade, Parkville, Victoria 3052, Australia, [2]Department of Computing and Information Systems, The University of Melbourne, Parkville, Victoria 3010, Australia and [3]Department of Mathematics and Statistics, The University of Melbourne, Parkville, Victoria 3010, Australia

Received October 14, 2012; Revised March 7, 2013; Accepted March 8, 2013

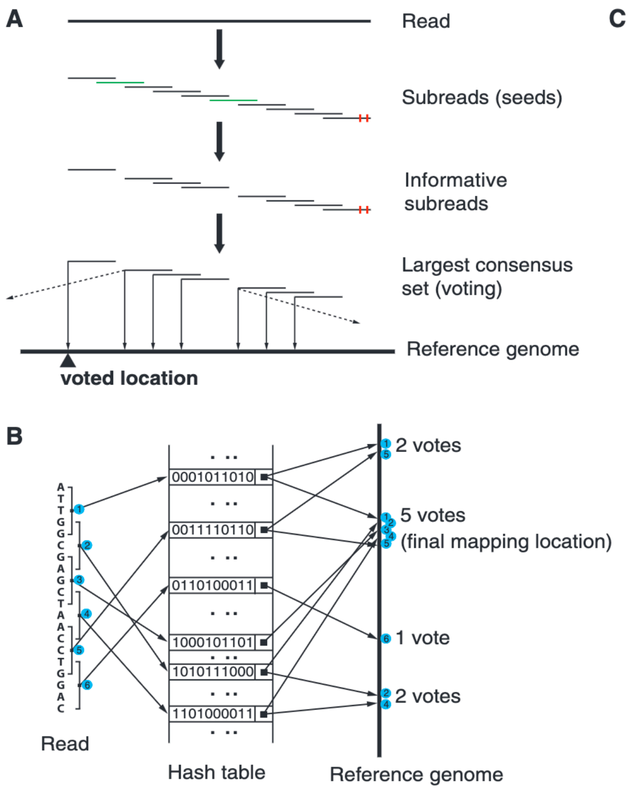






K-mer based seeding

- Subhead aligner extracts subreads from the read (seeds)
- Informative subheads are ones that don't occur too frequently <=24 times.
- Each sub-read "votes" on the implied location of the read (based on the list of associated positions and the subhead offset in the read)
- The location with the most votes wins (is the subject of further alignment)
- Must be careful to allow "wiggle room" in case there are insertions /deletions in the read



K-mer based seeding

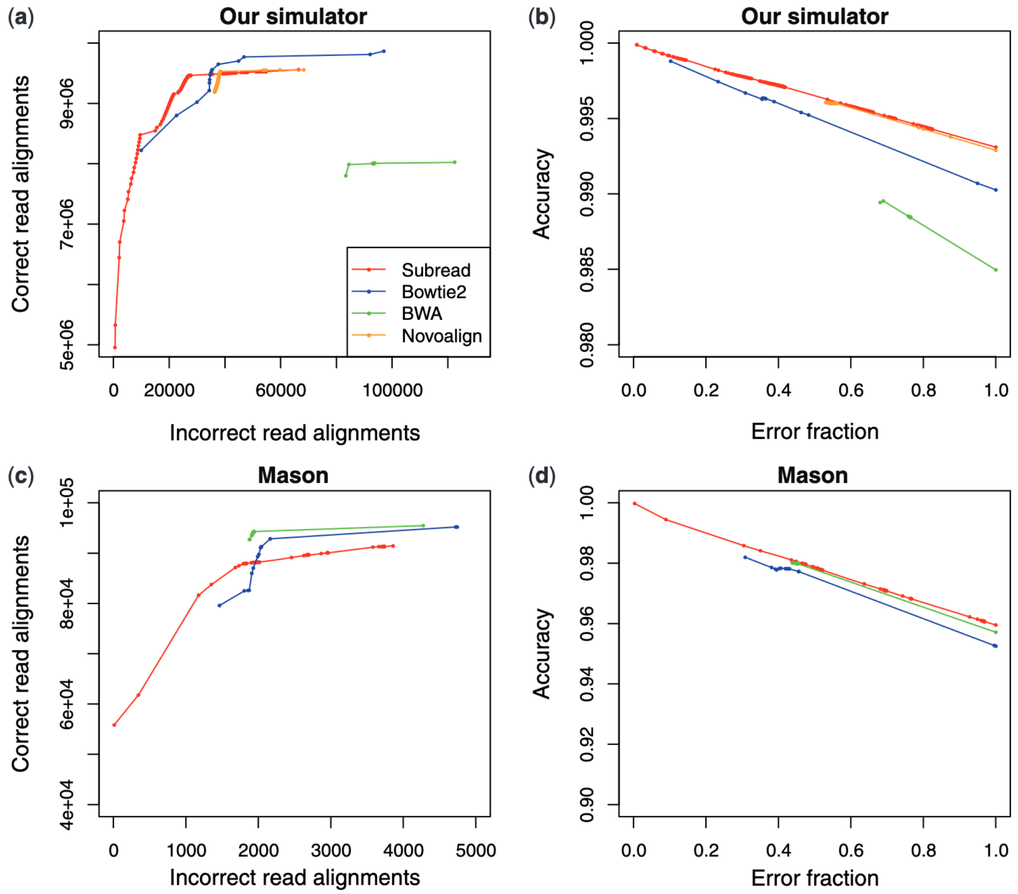


Figure 4. Recall and accuracy of aligners with respect to MQSs. (**a**) and (**c**) give the cumulative number of correctly mapped reads and incorrectly mapped reads from high to low mapping quality. (**b**) and (**d**) show the cumulative accuracy and error fractions from high to low mapping quality. (a) and (b) use the indel data set included in Table 4, and (c) and (d) use the Mason data set included in Table 5. Each point in each plot corresponds to an MQS given by an aligner. Subread was run with default setting in (a) and (b) and with -f 100 in (c) and (d).

Table 1. Performance of aligners in mapping genomic DNA reads from the 1000 Genomes project

Aligner	Mapped (%)	Rabema intervals (%)	Time (h)	Memory (Gb)
Subread (default)	97.7	86.7	1.6	7.6
Subread (low memory)	97.7	86.7	2.9	4.3
Bowtie2	99.1	87.2	6.0	3.3
BWA	95.6	82.6	15.2	3.3
Maq	98.1	86.3	48.3	19.1
Novoalign	93.9	68.9	18.7	8.2
MrsFast	70.3	73.8	48.2	25.8

Columns give the percentage of reads that are successfully mapped, the percentage of normalized found intervals given by the Rabema program (in 'all' category, maximal error rate of 8%), the time taken and the peak memory usage. Results are given for Subread with default settings and when set to use less memory.

Table 2. Performance of aligners in mapping RNA-seq reads from the SEQC project

Aligner	Mapped (%)	Time (min)	Memory (Gb)
Subread (default)	96.9	23	7.6
Subread (low memory)	96.9	40	4.3
Bowtie2	85.7	90	3.3
BWA	78.6	284	3.3
Maq	66.4	685	5.2
Novoalign	78.4	361	8.1
MrsFast	46.2	398	7.4

Columns give the percentage of reads that are successfully mapped, the time taken and the peak memory usage. Results are given for Subread with default settings and when set to use less memory.

The sub-read aligner has "comparable" accuracy to e.g. Bowtie2, but it is *substantially* faster.

Why?

K-mer based seeding

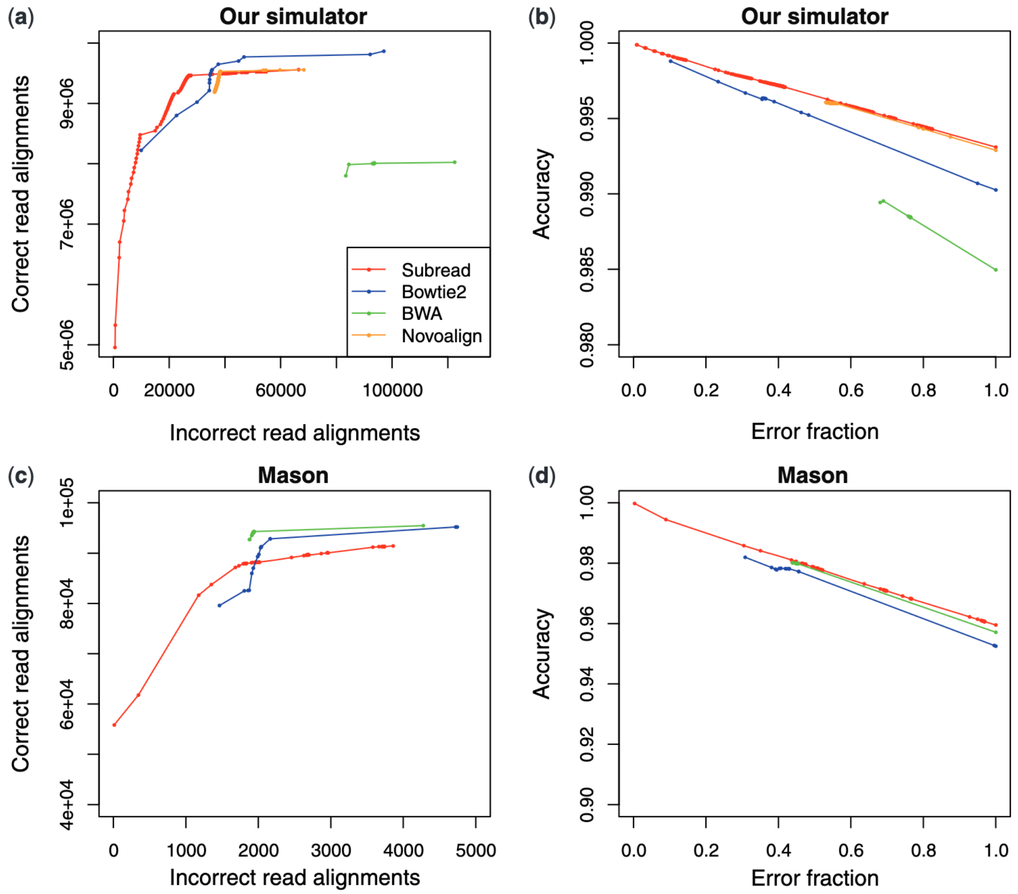


Figure 4. Recall and accuracy of aligners with respect to MQSs. (**a**) and (**c**) give the cumulative number of correctly mapped reads and incorrectly mapped reads from high to low mapping quality. (**b**) and (**d**) show the cumulative accuracy and error fractions from high to low mapping quality. (a) and (b) use the indel data set included in Table 4, and (c) and (d) use the Mason data set included in Table 5. Each point in each plot corresponds to an MQS given by an aligner. Subread was run with default setting in (a) and (b) and with -f 100 in (c) and (d).

Table 1. Performance of aligners in mapping genomic DNA reads from the 1000 Genomes project

Aligner	Mapped (%)	Rabema intervals (%)	Time (h)	Memory (Gb)
Subread (default)	97.7	86.7	1.6	7.6
Subread (low memory)	97.7	86.7	2.9	4.3
Bowtie2	99.1	87.2	6.0	3.3
BWA	95.6	82.6	15.2	3.3
Maq	98.1	86.3	48.3	19.1
Novoalign	93.9	68.9	18.7	8.2
MrsFast	70.3	73.8	48.2	25.8

Columns give the percentage of reads that are successfully mapped, the percentage of normalized found intervals given by the Rabema program (in 'all' category, maximal error rate of 8%), the time taken and the peak memory usage. Results are given for Subread with default settings and when set to use less memory.

Table 2. Performance of aligners in mapping RNA-seq reads from the SEQC project

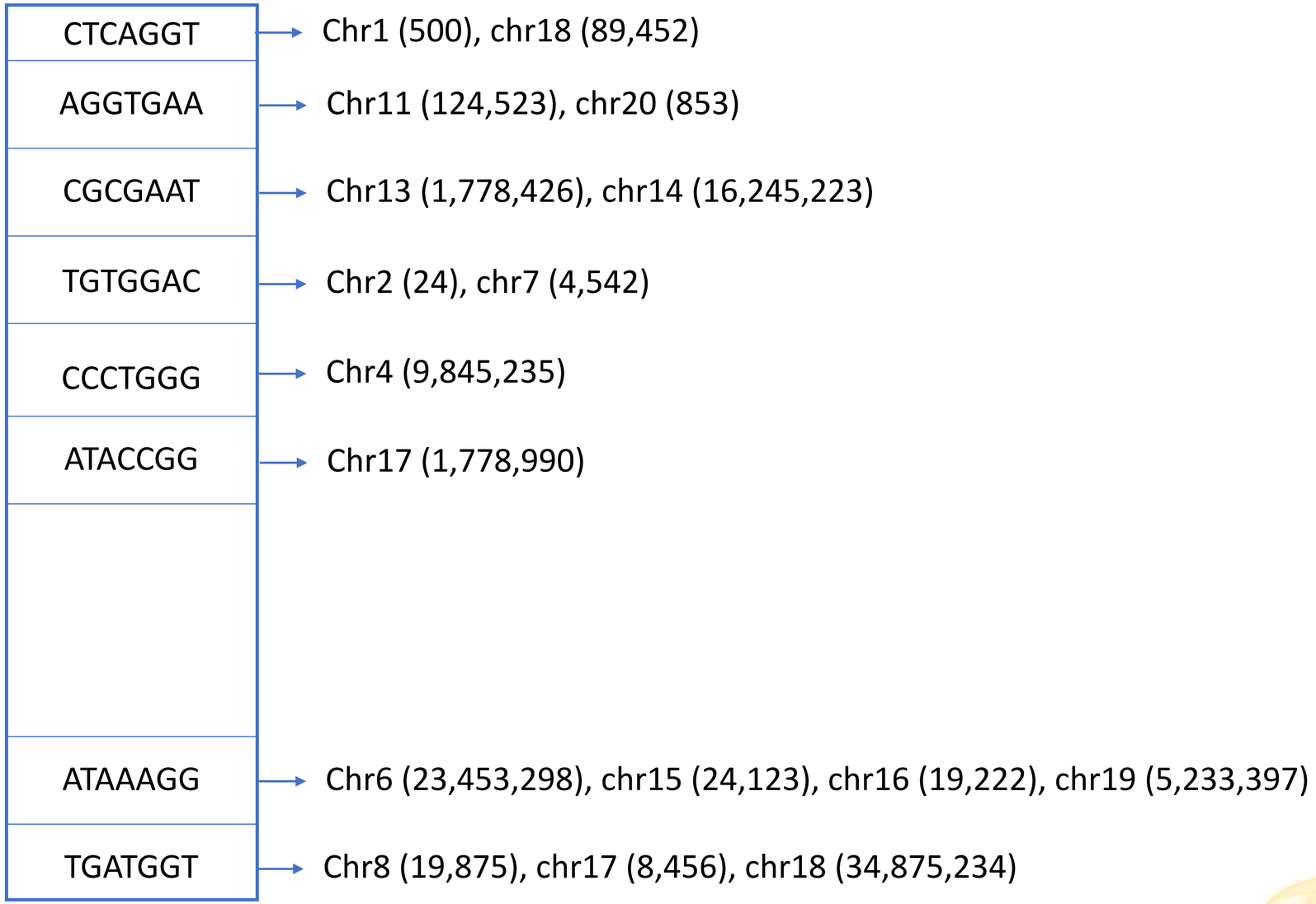
Aligner	Mapped (%)	Time (min)	Memory (Gb)
Subread (default)	96.9	23	7.6
Subread (low memory)	96.9	40	4.3
Bowtie2	85.7	90	3.3
BWA	78.6	284	3.3
Maq	66.4	685	5.2
Novoalign	78.4	361	8.1
MrsFast	46.2	398	7.4

Columns give the percentage of reads that are successfully mapped, the time taken and the peak memory usage. Results are given for Subread with default settings and when set to use less memory.

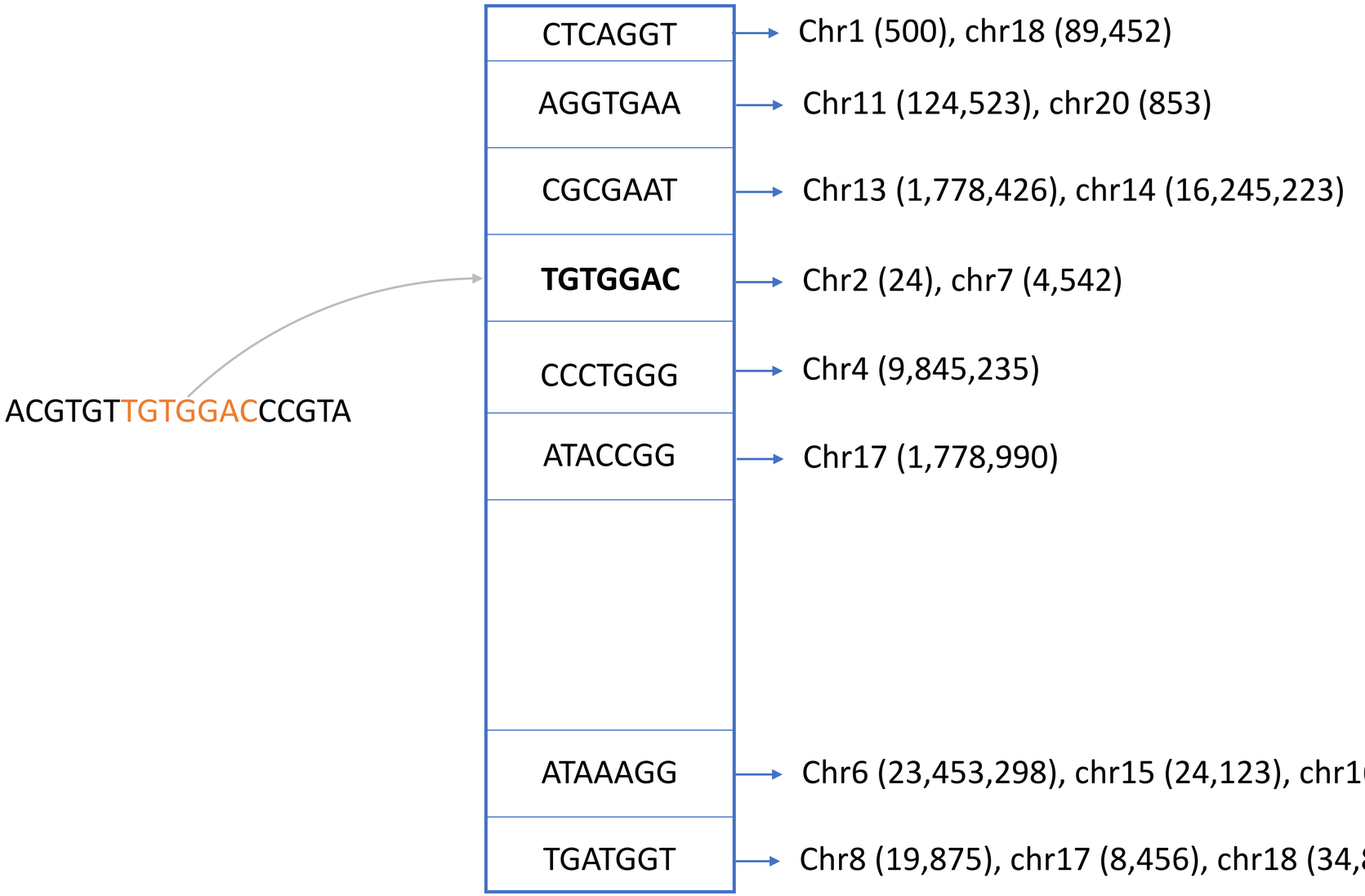
The sub-read aligner has "comparable" accuracy to e.g. Bowtie2, but it is *substantially* faster.

Why? Hash-table lookup is *fast* and *cache efficient*.

K-mer based seeding

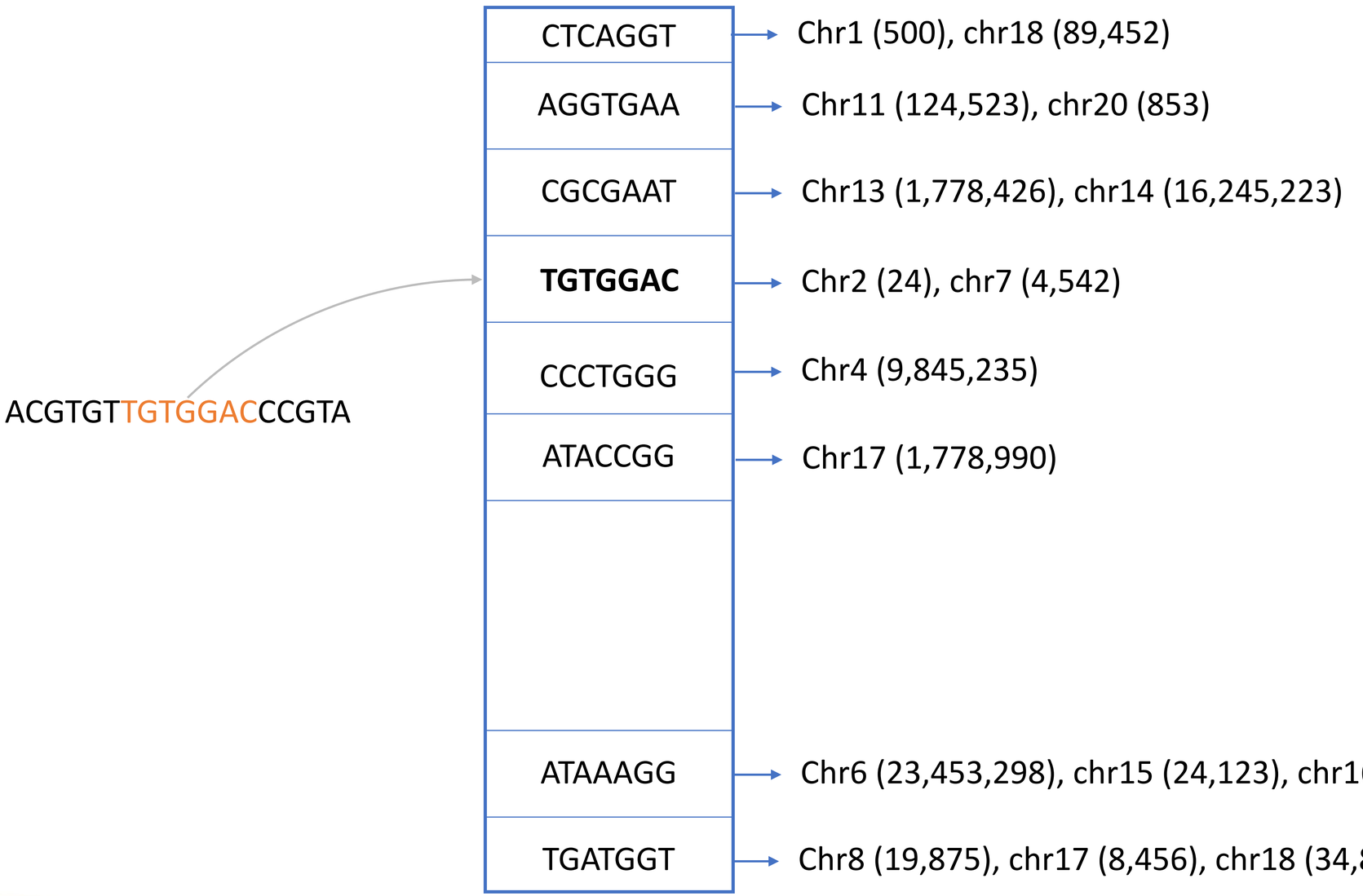


K-mer based seeding



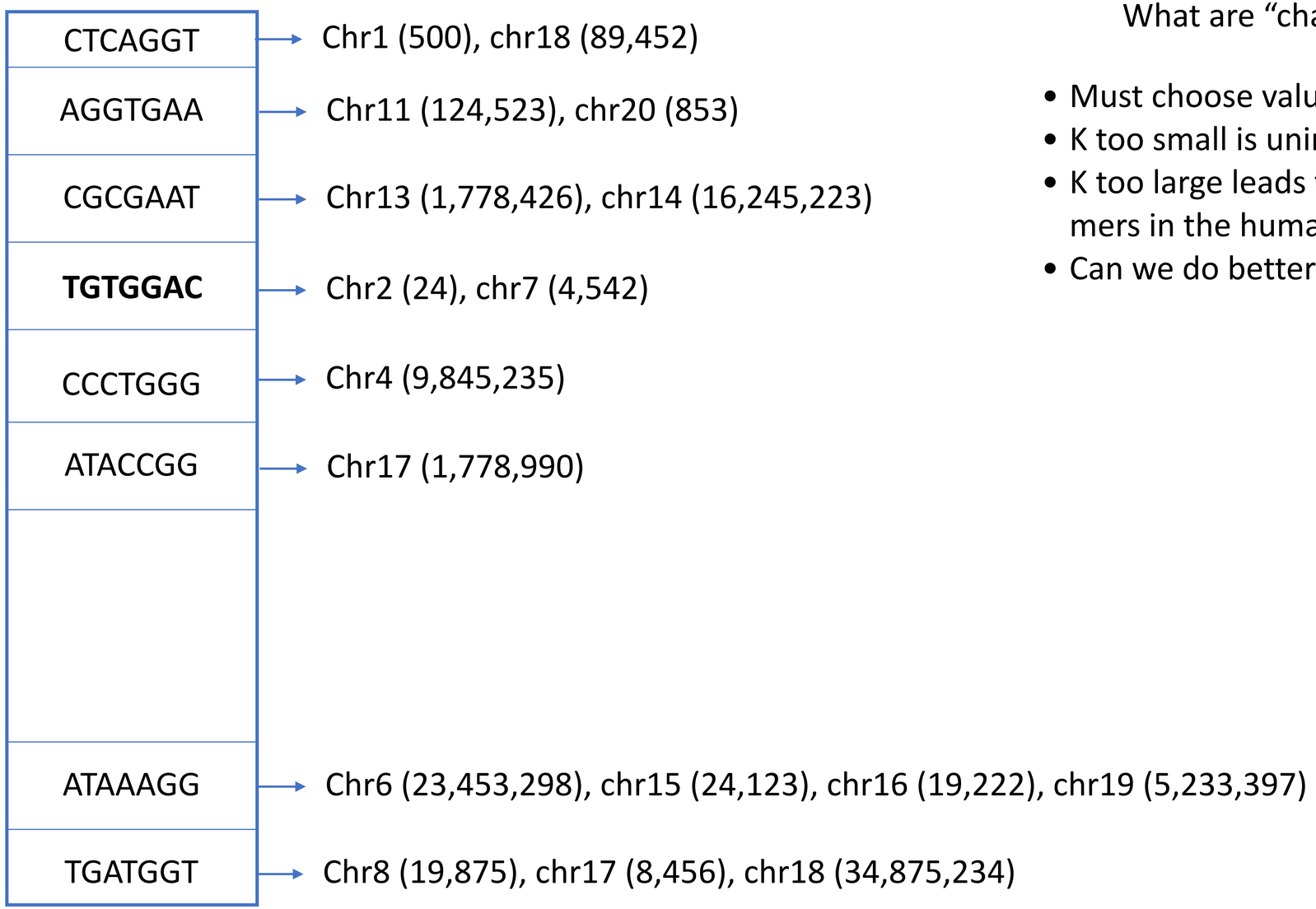
- Basic “inverted index”
- Keys are k-mers, values are *lists* of positions
- You look up potential “keys” from your query in your index
- The associated values immediately tell you where the seeds occur in your reference.

K-mer based seeding



- "Intersect" occurrences of different keys to determine a consensus or putative mapping position

K-mer based seeding



What are “challenges” in k-mer based indexes?

- Must choose value of k beforehand (built into index)
- K too small is uninformative
- K too large leads to very big indices! At k=20 most k-mers in the human genome are unique
- Can we do better?

Minimizers: Sparsifying k-mers

The **minimizer** of a k-mer is the smallest length m sub-sequence of the k-mer under some ordering σ

ACTGACCCGTAGC **k-mer X (k=13)**

ACTGACCCGTAGC **minimizer of x (for m=3,** σ = alphabetical ordering**)**

> Bioinformatics. 2004 Dec 12;20(18):3363-9. doi: 10.1093/bioinformatics/bth408. Epub 2004 Jul 15.

Reducing storage requirements for biological sequence comparison

Michael Roberts [1], Wayne Hayes, Brian R Hunt, Stephen M Mount, James A Yorke

ARTICLE

Winnowing: local algorithms for document fingerprinting

Authors: Saul Schleimer, Daniel S. Wilkerson, Alex Aiken Authors Info & Claims

SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data • June 2003 • Pages 76–85 • https://doi.org/10.1145/872757.872770

This can be useful for partitioning / grouping k-mers

ACTGACCCGTAGCGCTAGATAAC

ACTGACCCGTAGC**GCTAGA**TAAC

ACTGACCCGTAGCGCTAGATAAC

All k-mers in this window of length 19 share the *same* minimizer; they are called a **super k-mer**

A super k-mer can have length between k and 2k-m; provides a way to group k-mers looking only at it's actual sequence!

Minimizers: Sparsifying k-mers

- Minimizers provide a way of *deterministically* selecting an *informative* subset of "mers"
- This is useful in choosing which k-mers to index
  - We could index every "i-th" k-mer, what if we choose a cadence to tile our query that's at odds with this sampling?
  - Instead of indexing **all** k-mers, or every i-th k-mer, what if select the *minimizers of the sequence*?
  - Because minimizers are selected *consistently* according to the same rule, if the same k-mer appears in both the query and the reference, it will have the same minimizer.
  - This leads to a *sparser* index that is *smaller* and therefore faster to build and often quicker to query.

Minimizers: Sparsifying k-mers

- The specific manner in which one chooses minimizers matters!
- The "original" (alphabetical) definition of minimizers is problematic — it tends to highly concentrate on "small" (alphabetically early) k-mers… obviously
- Unfortunately, this can lead to the selection of highly repetitive or low complexity k-mers.
- The minimizer is defined by 3 parameters:
  - w (window length), k (k-mer length), O (the ordering imposed on k-mers). For example, (20,13,alpha) minimizers would be minimizers defined for each window of length 20, with k-mers of length 13, using the alphabetical ordering.
  - One way to help resolve the problem with alphabetical ordering is to instead choose a "random" ordering — these are usually good. Where might we get a random order?

Minimizers: Sparsifying k-mers

- The specific manner in which one chooses minimizers matters!
- The "original" (alphabetical) definition of minimizers is problematic — it tends to highly concentrate on "small" (alphabetically early) k-mers… obviously
- Unfortunately, this can lead to the selection of highly repetitive or low complexity k-mers.
- The minimizer is defined by 3 parameters:
  - w (window length), k (k-mer length), O (the ordering imposed on k-mers). For example, (20,13,alpha) minimizers would be minimizers defined for each window of length 20, with k-mers of length 13, using the alphabetical ordering.
  - One way to help resolve the problem with alphabetical ordering is to instead choose a "random" ordering — these are usually good. Where might we get a random order?
    - Consider hashing! Let h(.) be a "good" random hash function. We define the order of 2 k-mers as $k_1 < k_2$ iff $h(k_1) < h(k_2)$

Minimizers: Sparsifying k-mers

Sequence analysis

Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences

Heng Li

Medical Population Genetics, Broad Institute, Cambridge, MA 02142, USA

Associate Editor: Inanc Birol

Received on December 6, 2015; revised on March 14, 2016; accepted on March 14, 2016

Sequence analysis

Minimap2: pairwise alignment for nucleotide sequences

Heng Li*

Department of Medical Population Genetics Program, Broad Institute, Cambridge, MA 02142, USA

*To whom correspondence should be addressed.
Associate Editor: Inanc Birol

Received on January 1, 2018; revised on March 16, 2018; editorial decision on March 22, 2018; accepted on May 4, 2018

The minimizer idea has been hugely influential, and there are many specific definitions of minimizers we can use, and other places minimizers come up. Yet, we can imagine even more types of seeds!

Strobemers: Robust seeds

Method

Effective sequence similarity detection with strobemers

Kristoffer Sahlin

Department of Mathematics, Science for Life Laboratory, Stockholm University, 10691 Stockholm, Sweden

k-mer-based methods are widely used in bioinformatics for various types of sequence comparisons. However, a single mutation will mutate *k* consecutive *k*-mers and make most *k*-mer-based applications for sequence comparison sensitive to variable mutation rates. Many techniques have been studied to overcome this sensitivity, for example, spaced *k*-mers and *k*-mer permutation techniques, but these techniques do not handle indels well. For indels, pairs or groups of small *k*-mers are commonly used, but these methods first produce *k*-mer matches, and only in a second step, a pairing or grouping of *k*-mers is performed. Such techniques produce many redundant *k*-mer matches owing to the size of *k*. Here, we propose *strobemers* as an alternative to *k*-mers for sequence comparison. Intuitively, strobemers consist of two or more linked shorter *k*-mers, where the combination of linked *k*-mers is decided by a hash function. We use simulated data to show that strobemers provide more evenly distributed sequence matches and are less sensitive to different mutation rates than *k*-mers and spaced *k*-mers. Strobemers also produce higher match coverage across sequences. We further implement a proof-of-concept sequence-matching tool StrobeMap and use synthetic and biological Oxford Nanopore sequencing data to show the utility of using strobemers for sequence comparison in different contexts such as sequence clustering and alignment scenarios.

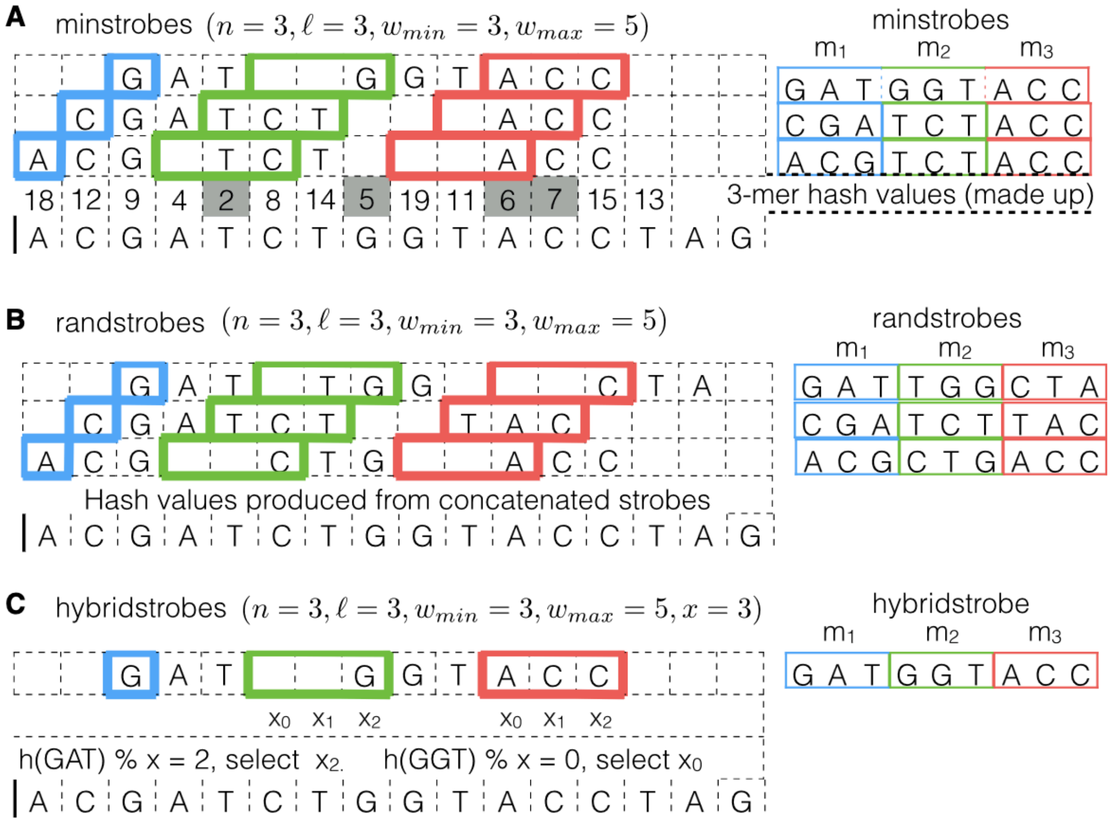
[Supplemental material is available for this article.]

Method | Open access | Published: 15 December 2022

Strobealign: flexible seed size enables ultra-fast and accurate read alignment

Kristoffer Sahlin

Genome Biology **23**, Article number: 260 (2022) | Cite this article



Strobemers: Robust seeds

- Basic idea (minstrobe) : strobemer determined by parameters (n, l, w_{min}, w_{max})
- n is the number of strobes in the strobemer
- l is the strobe length
- w_{min} and w_{max} define the windows from which strobes are extracted
- Let j be position from which the first strobe is chosen, consider the downstream window from position j + w_{min} to j + w_{max} define the l minimizer of this window to be the next strobe
- Now, look at the window j + w_{min} + w_{max} to j + 2*w_{max} and extract the minimizer from that window
- For the general case, the n^{th} minstrobemer comes from window j + w_{min} + (n-2)*w_{max} to j + (n-1)*w_{max}

Strobemers: Robust seeds

Fig. 2

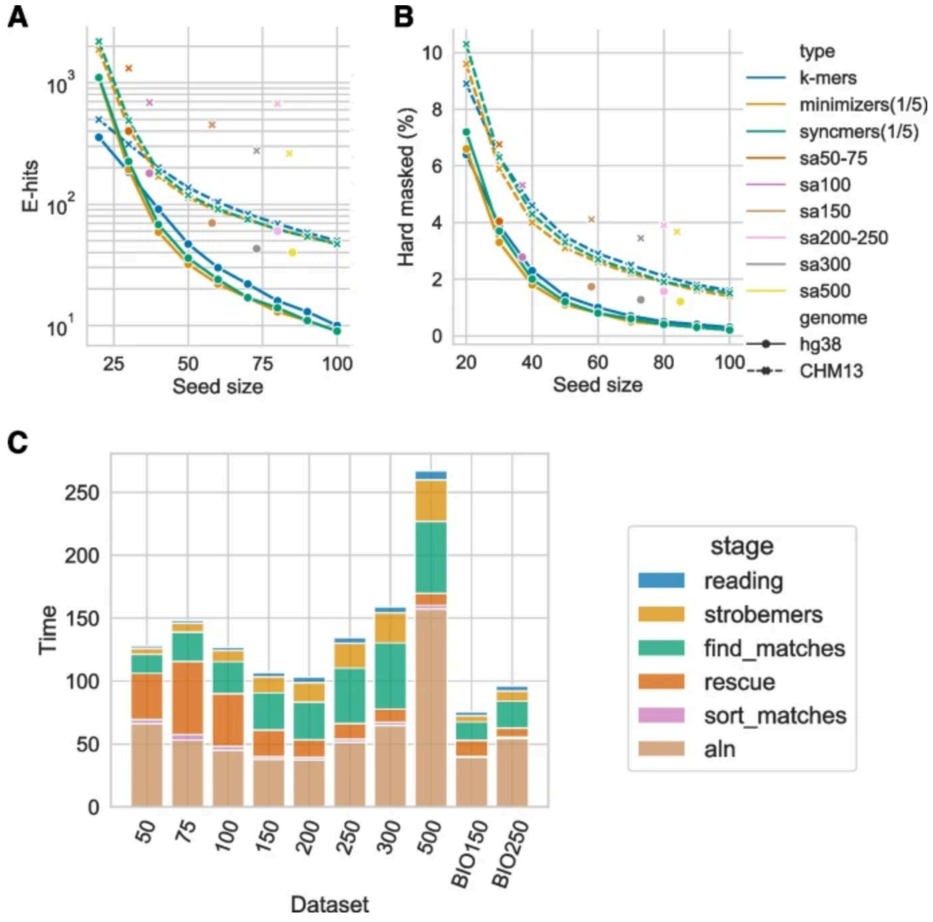
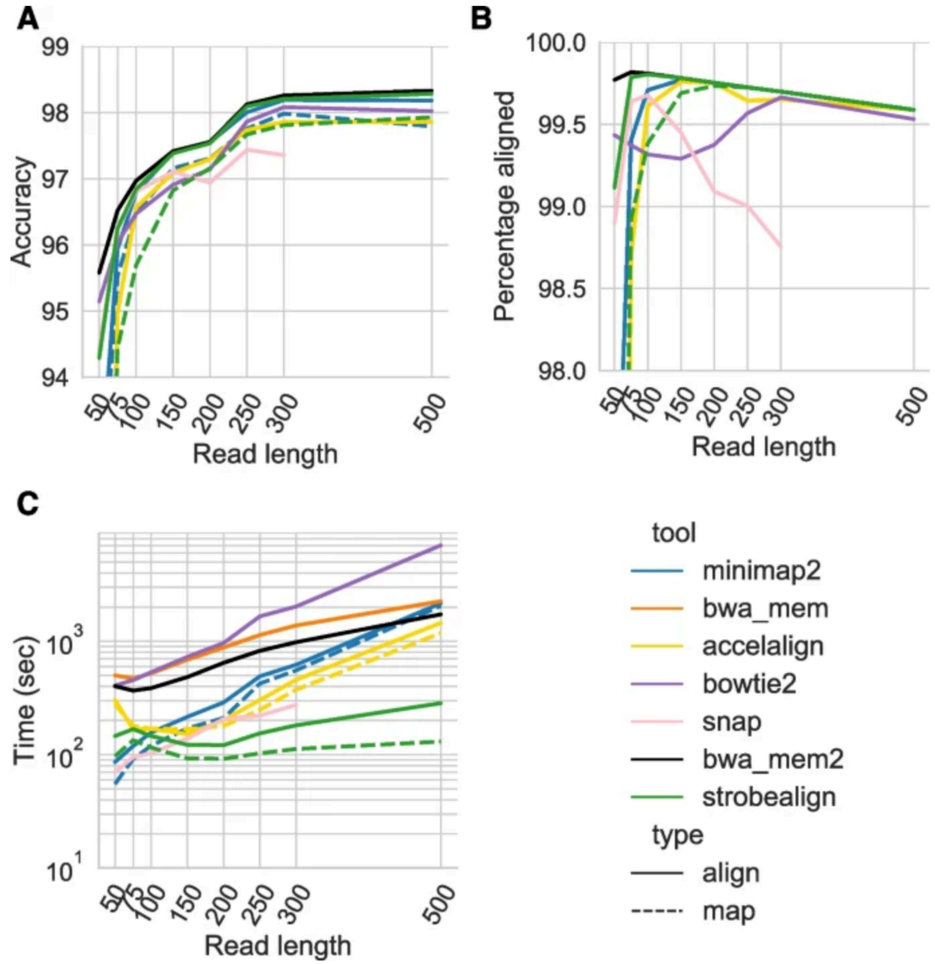


Fig. 3



Accuracy (**A**), percent aligned reads (**B**), and runtime (**C**) of aligning paired-end reads to the SIM3 dataset