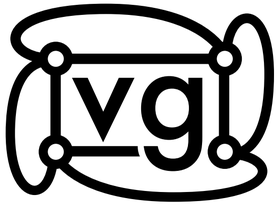
Giraffe: A Pangenomic Short Read Aligner

Jouni Sirén
UCSC Genomics Institute

Giraffe aligner

- Pangenomic short read aligner.
- Combines the speed of linear aligners with the accuracy of graph-based aligners.
- Designed for (paired-end 150 bp) Illumina reads, where most sequencing errors are substitutions.
- Restricts its attention to paths that are locally consistent with observed haplotypes.
- Part of the VG toolkit.



https://github.com/vgteam/vg

Pangenomics enables genotyping of known structural variants in 5202 diverse genomes

Jouni Sirén[1]†, Jean Monlong[1]†, Xian Chang[1]†, Adam M. Novak[1]†, Jordan M. Eizenga[1]†, Charles Markello[1], Jonas A. Sibbesen[1], Glenn Hickey[1], Pi-Chuan Chang[2], Andrew Carroll[2], Namrata Gupta[3], Stacey Gabriel[4], Thomas W. Blackwell[5], Aakrosh Ratan[6], Kent D. Taylor[7], Stephen S. Rich[6], Jerome I. Rotter[7], David Haussler[1,8], Erik Garrison[9], Benedict Paten[1]*

To appear in Science, 2021.

Preprint on bioRxiv:
https://doi.org/10.1101/2020.12.04.412486

Read alignment: the easy part

Mapping: Approximate location of the read in the reference.

Alignment: The best base-to-base alignment between the read and the reference near a particular mapping.

Typical considerations: mapping speed, fraction of mapped reads, fraction of correct mappings.

With these goals, read aligner design has equal parts of science, engineering, and art to it.

Mostly a principled process with some heuristics and computational parameters.

Seed and extend approach:

1. Find **seeds** (partial alignments) using a text index.

2. **Cluster** the seeds that correspond to the same mapping (or **chain** them into rough alignments with long reads).

3. **Extend** the seeds into full alignments.

We continue exploring promising mappings until we are confident that we have found the best alignment.

The ugly part

Mapping quality: Our confidence that we have chosen the correct mapping.

mapq = -10 $\log_{10}$ p, where p is (the estimated) error probability.

Usually capped at 60 (one-in-million) and rarely calibrated that well in practice.

We model the probability to get the read from a particular alignment due to sequencing errors and unknown variants.

In principle, we can compute the mapping quality from the probabilities over all possible alignments (or mappings).

In practice, we use heuristics to determine when unexplored mappings should no longer have a significant effect on mapq.

We also estimate the probability that the heuristics went wrong and there was a good alignment we did not find.

It seems unavoidable that we spend months investigating why we map a handful of reads incorrectly with mapq 60 and how to deal with them without making the aligner too slow.

From VG map to Giraffe

Garrison et al.: **Variation graph toolkit improves read mapping by representing genetic variation in the reference**.
Nature Biotechnology, 2018.
DOI: 10.1038/nbt.4227

The original VG aligner can be understood as an adaptation of BWA-MEM for graphs.

It is more **accurate** than aligners mapping short reads to linear reference sequences but also ~10x **slower** than them.

Index construction requires graph pruning and takes 1–2 days for a human graph.

Complex graph regions are computationally expensive, because the number of possible paths grows exponentially with the number of variants.

Giraffe started as an attempt to deal with the weaknesses of VG map by:

- relying less on sophisticated text indexes;
- using haplotype information to deal with complex regions; and
- avoiding expensive dynamic programming.

Giraffe data model

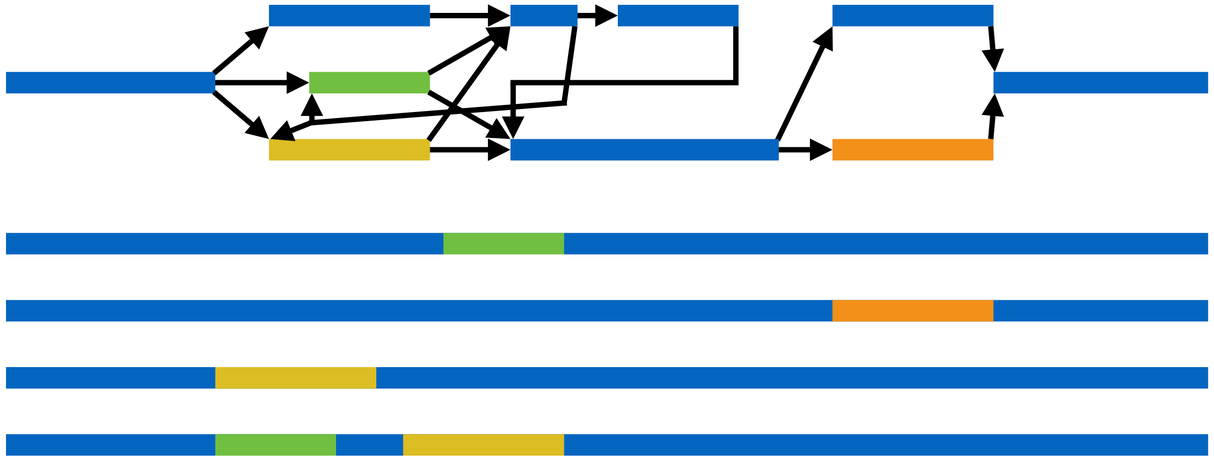
The graph represents a collection of **aligned haplotypes**.

Positions in the haplotypes that map to the same node are considered equivalent.

Each **traversal** of the graph is a potential haplotype.

Traversals that are locally consistent with the original haplotypes are more likely to be biologically plausible.

The original haplotypes are stored as **paths**.



path ~ walk
path ~ stored traversal
traversal ~ emergent path

GBWT

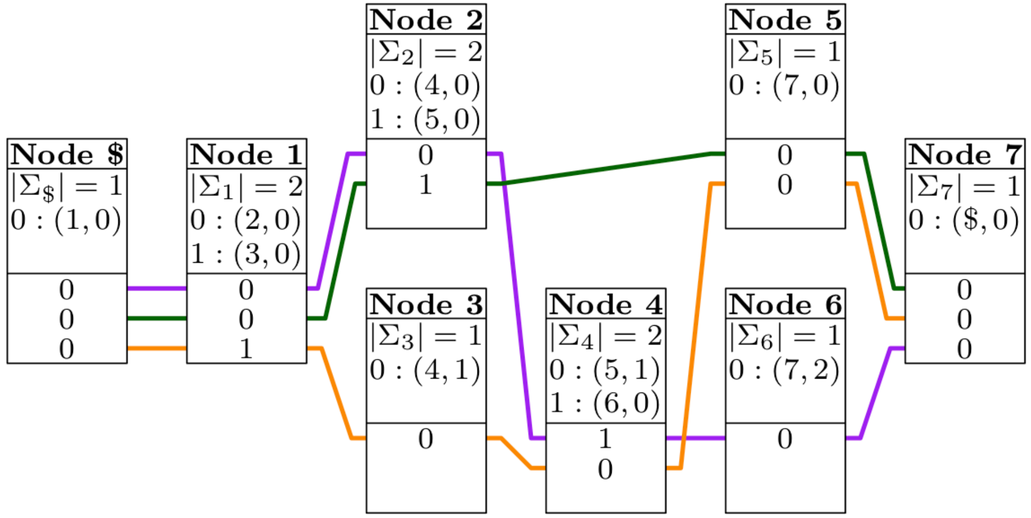
GBWT is effectively RLBWT for integer sequences. (There should be a talk on RLBWT on Friday.)

We choose to interpret the integers as nodes and the sequences as haplotype paths.

If the paths are similar enough, the GBWT can store them space-efficiently.

We partition the BWT and the rank structure into nodes, which improves memory locality.

For any graph traversal, we can easily determine how many indexed paths contain the traversal as a subpath.



Sirén et al.: **Haplotype-aware graph indexes**. Bioinformatics, 2020.
DOI: 10.1093/bioinformatics/btz575

https://github.com/jltsiren/gbwt
https://github.com/jltsiren/gbwtgraph
https://github.com/jltsiren/gbwt-rs

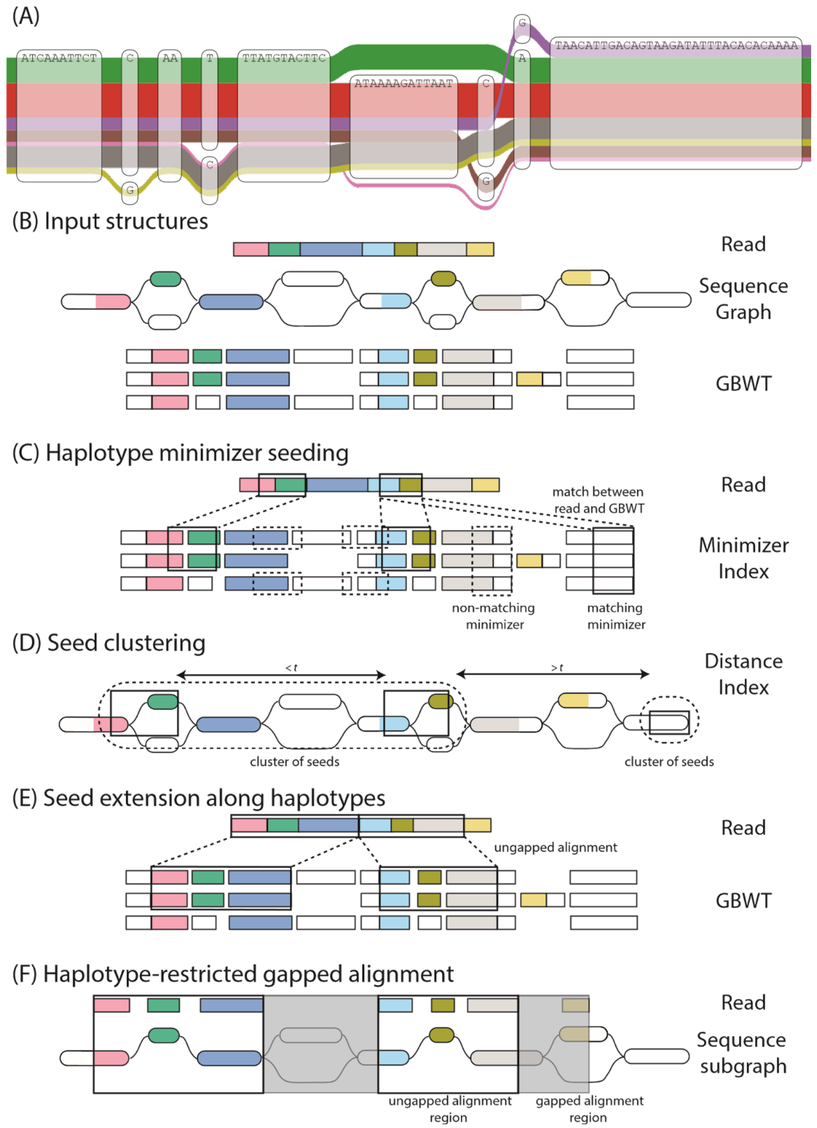
Giraffe algorithm

1. Find seeds using a minimizer index of the haplotypes.

2. Cluster the seeds using a distance index based on a hierarchical decomposition of the graph.

3. Extend the seeds over the haplotypes, allowing for a limited number of mismatches.

4. If we did not get enough full-length alignments, align the tails of best partial extensions using dynamic programming over the haplotypes.



1. Minimizer seeds

Due to my background, I used to think that text indexes are a big deal in read aligners.

For Giraffe, we chose to use a **minimizer index**: a simple hash table that maps k-mers to graph positions.

A (w, k)-minimizer is the k-mer with the smallest hash value among all k-mers and their reverse complements in a w + k – 1 bp window.

We index all minimizers in the haplotypes. Index construction takes 5–10 minutes for a human genome graph, which may be faster than loading the index from a network drive.

Most minimizers are unique (in the graph), while minimizers occurring in repetitive regions may have too many hits to be useful.

For each read, we choose all minimizers below the **soft hit cap** (10 hits) and some minimizers below the **hard hit cap** (500 hits), using a scoring heuristic.

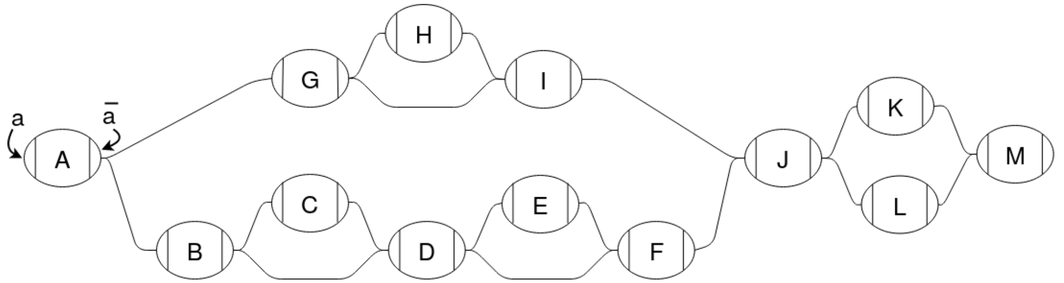
We try to avoid using minimizers with multiple occurrences in the read, because each (read position, graph position) pair is a separate seed.

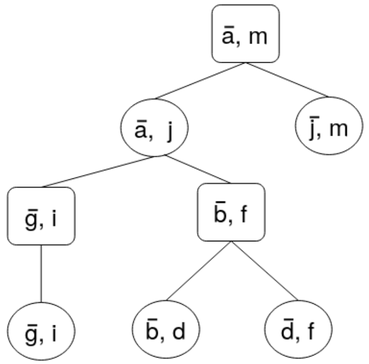
2. Seed clustering

Clustering requires measuring distances between graph positions, which can be slow.

We use a **distance index** based on the **snarl decomposition** of the graph, which reduces distances in the graph to distances in a shallow tree.

Each graph component is a chain of snarls, and each snarl is primitive or consists of parallel chains.





Chang et al.: **Distance indexing and seed clustering in sequence graphs**. Bioinformatics, 2020.
DOI: 10.1093/bioinformatics/btaa446

3. Seed extension

We **extend** the seeds in most promising clusters, chosen primarily by the fraction of the read covered by the minimizers.

Before extending, we merge seeds corresponding to the same alignment between the read and a node.

For each seed, we find the extension with the highest **alignment score** among all extensions over local haplotypes.

Seed extension is based on traversing a bidirectional GBWT (starting from the seed node) and counting the number of mismatches between read/node substrings.

We allow any number of mismatches in the seed node, 4 mismatches in total, and 2 mismatches in each flank (even if that would exceed the overall mismatch bound).

If there are full-length extensions, we return those that do not overlap too much.

Otherwise we trim the partial extensions to maximize the alignment scores and return all distinct extensions.

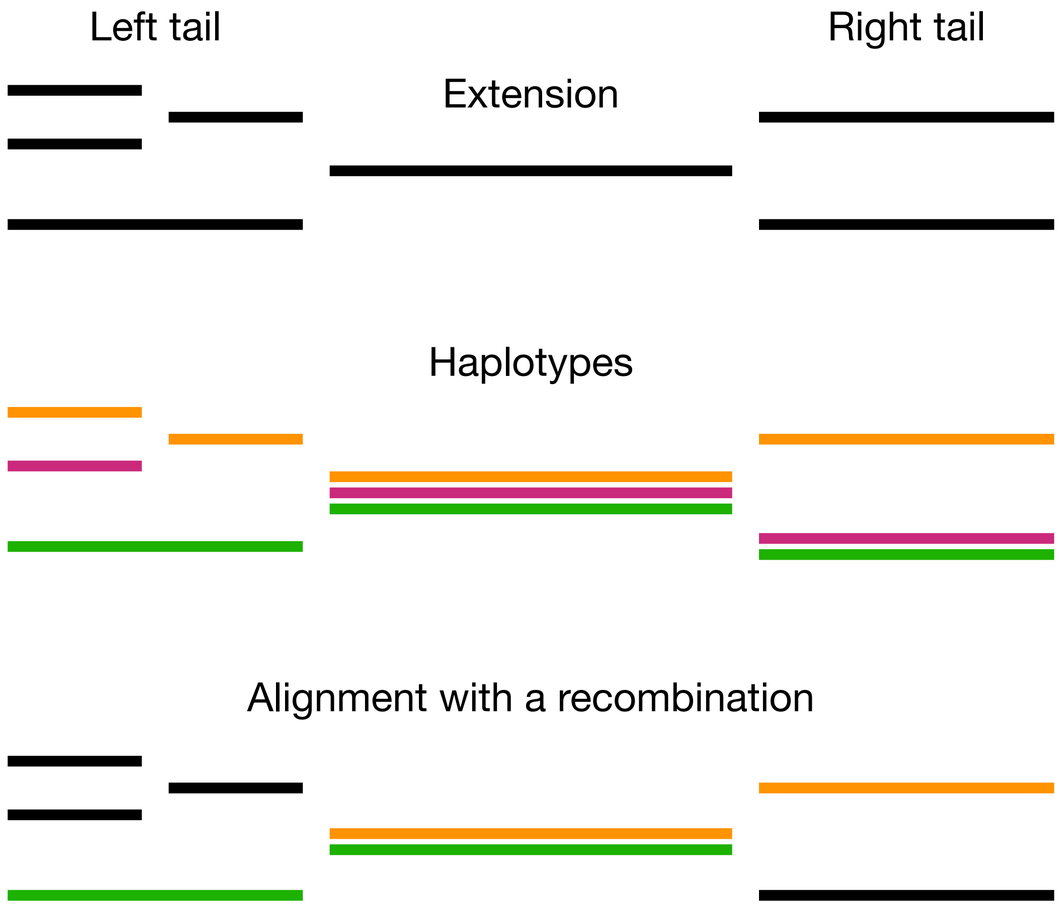
4. Gapped alignment

If we did not find enough full-length alignments, the read may contain **indels** that are not present in the haplotypes.

We then resort to **dynamic programming**, aligning the tails of most promising extensions over all local haplotypes.

The decisions which extensions to align involve some of the ugliest and most complicated heuristics in Giraffe.

Because we handle the tails independently, we may also find alignments corresponding to **recombinations** of the haplotypes.

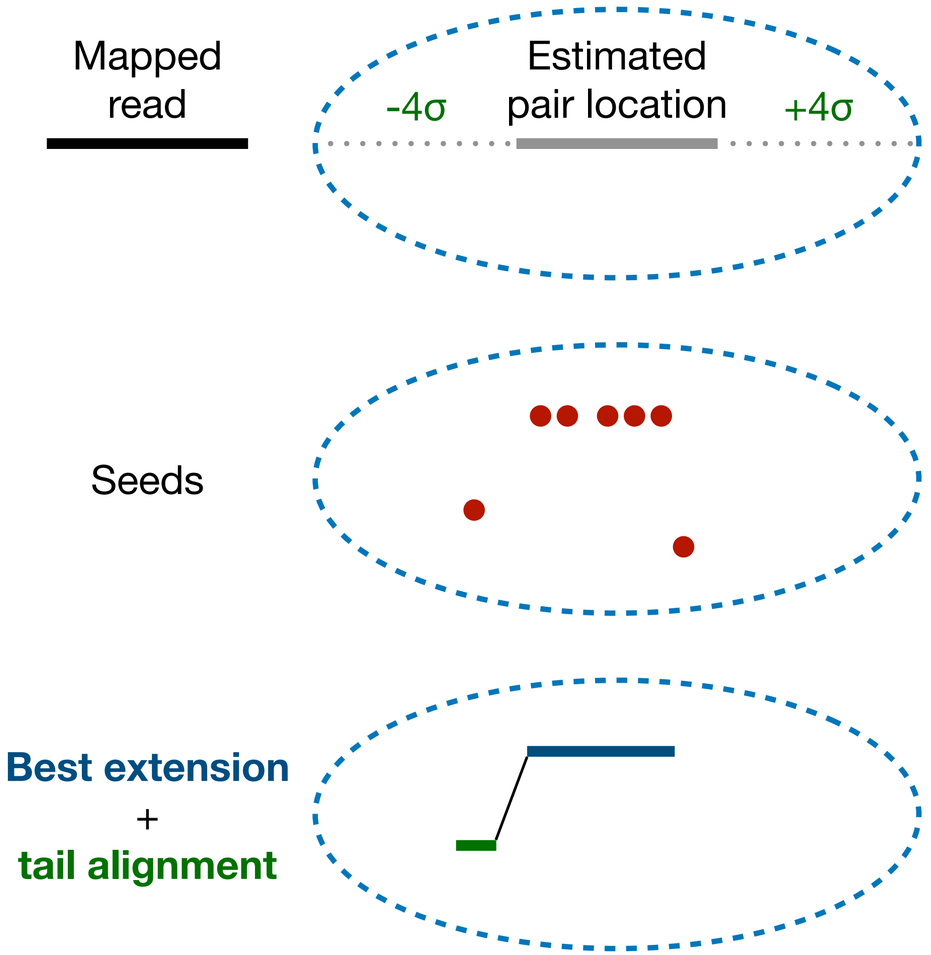


Paired-end mapping

If **fragment length distribution** is not provided, Giraffe estimates it with single-end mapping until there are enough confidently mapped pairs.

We cluster the seeds for both reads at the same time and pair clusters that are approximately at the right distance from each other.

If we have a confidently mapped read with an unmapped pair, we extract a subgraph and try to **rescue** the pair by aligning it using a simplified version of the Giraffe algorithm.



Artificial haplotypes

1. The graph we use as a mapping target may contain components (e.g. decoy sequences, unlocalized contigs) without haplotype information.

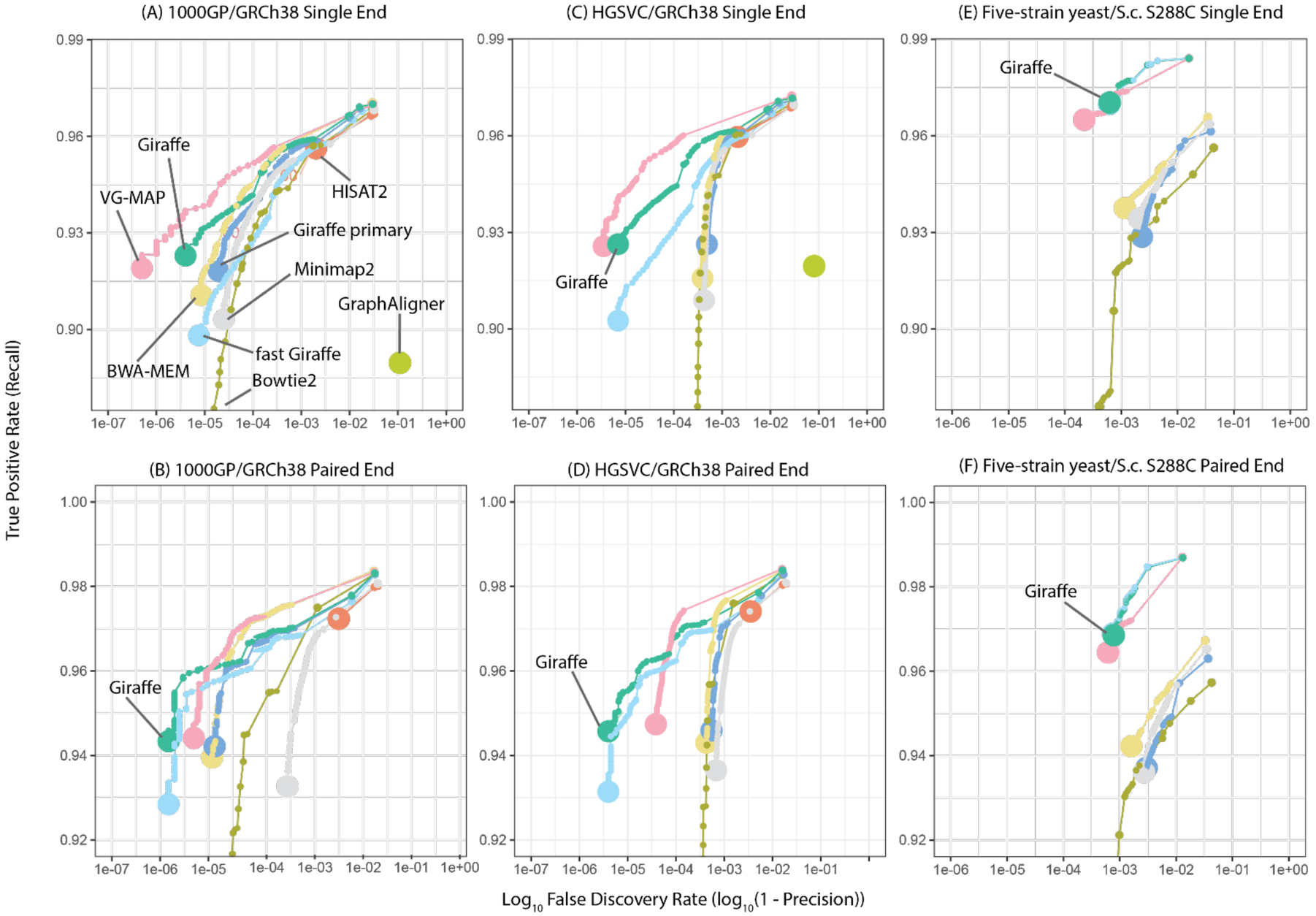
2. When we add a variant to the graph, it increases mapping accuracy for samples that contain the variant and reduces the accuracy for samples that do not contain it. For rare variants, the trade-off may not be worth it.

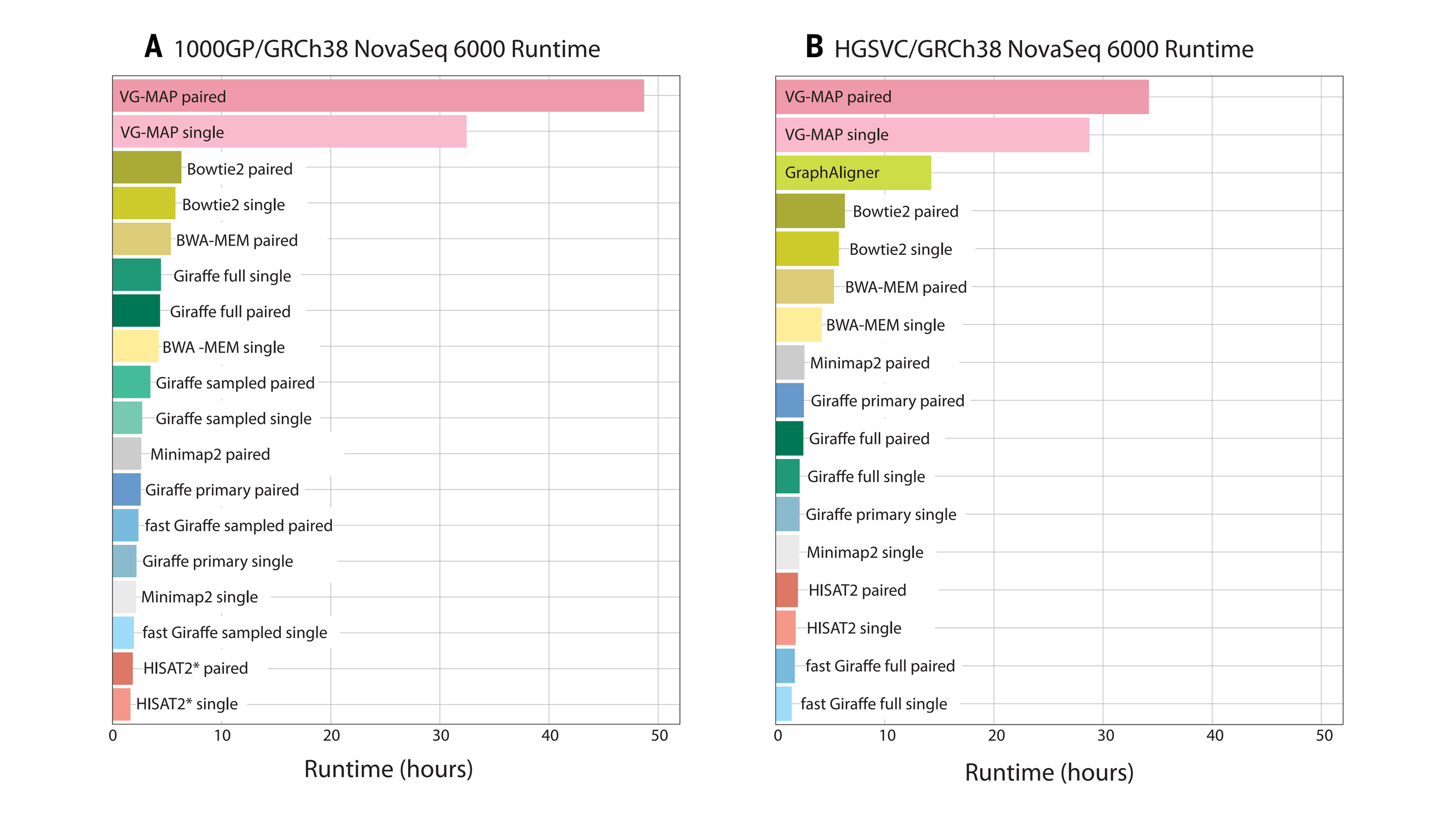
3. Mapping speed depends on the complexity of the graph and number of distinct local haplotypes.

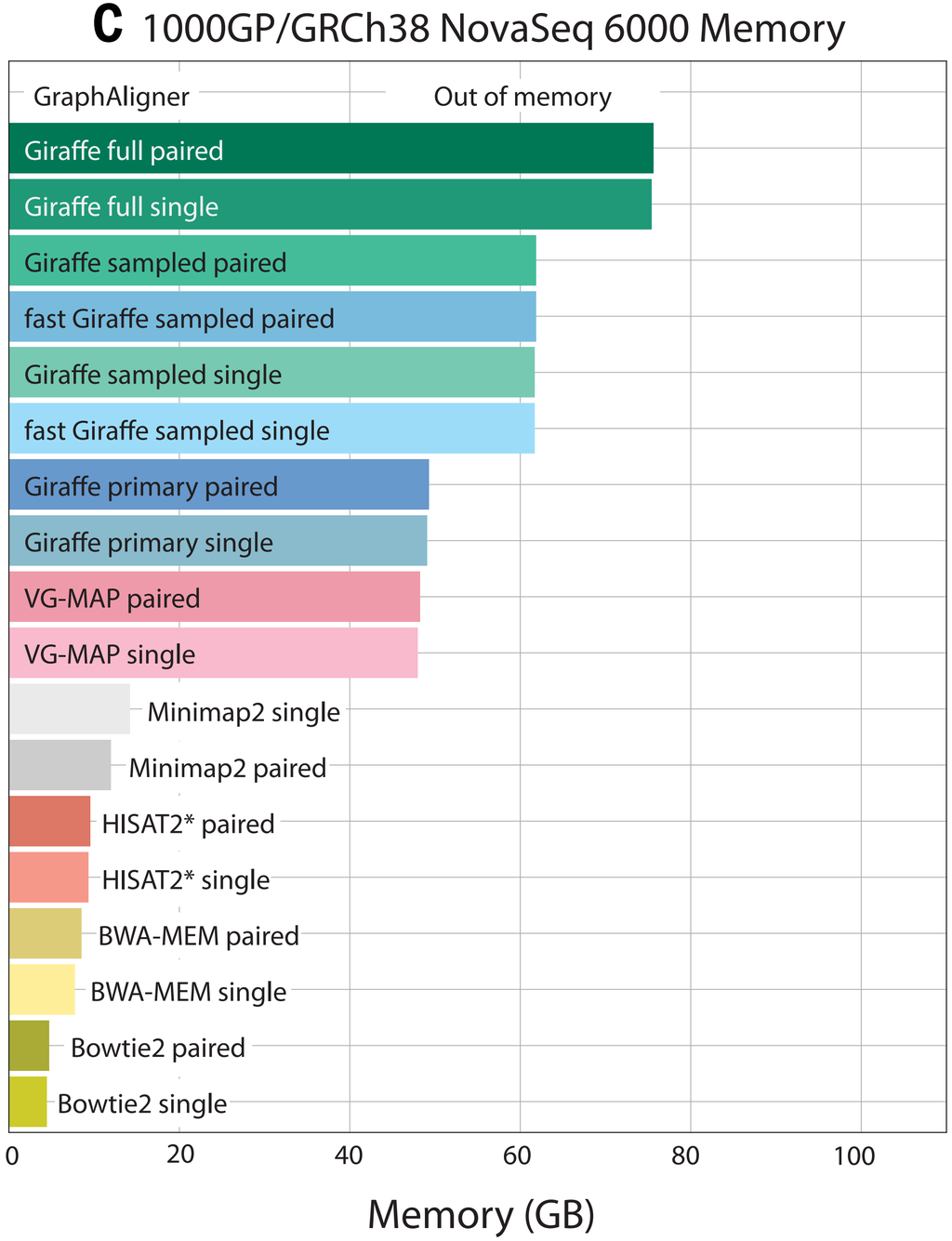
We can deal with these issues by generating artificial haplotypes with a greedy algorithm that **samples** k-node (default k = 4) local haplotypes proportionally.

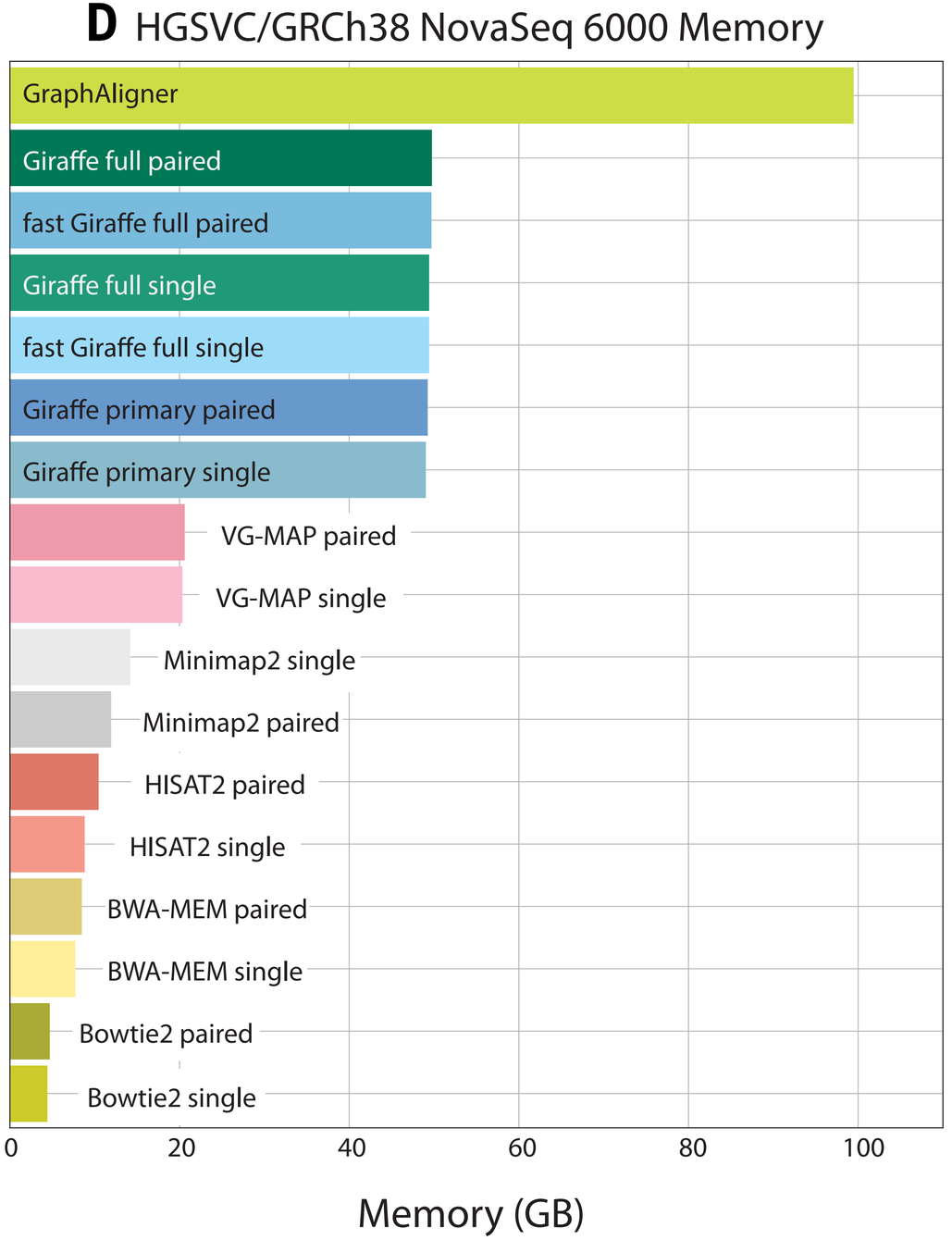
Inspired by:
Ghaffaari, Marschall: **Fully-sensitive seed finding in sequence graphs using a hybrid index**. Bioinformatics, 2019.

If there are more than ~200 haplotypes in the graph, the current best practice is downsampling them to 64 artificial haplotypes.


(A) 1000GP/GRCh38 Single End
(C) HGSVC/GRCh38 Single End
(E) Five-strain yeast/S.c. S288C Single End
(B) 1000GP/GRCh38 Paired End
(D) HGSVC/GRCh38 Paired End
(F) Five-strain yeast/S.c. S288C Paired End
True Positive Rate (Recall)
Log10 False Discovery Rate (log10(1 - Precision))
Giraffe
VG-MAP
HISAT2
Giraffe primary
Minimap2
GraphAligner
BWA-MEM
fast Giraffe
Bowtie2
250000
500000
750000


A 1000GP/GRCh38 NovaSeq 6000 Runtime
VG-MAP paired
VG-MAP single
Bowtie2 paired
Bowtie2 single
BWA-MEM paired
Giraffe full single
Giraffe full paired
BWA -MEM single
Giraffe sampled paired
Giraffe sampled single
Minimap2 paired
Giraffe primary paired
fast Giraffe sampled paired
Giraffe primary single
Minimap2 single
fast Giraffe sampled single
HISAT2* paired
HISAT2* single
0
10
20
30
40
50
Runtime (hours)
B HGSVC/GRCh38 NovaSeq 6000 Runtime
VG-MAP paired
VG-MAP single
GraphAligner
Bowtie2 paired
Bowtie2 single
BWA-MEM paired
BWA-MEM single
Minimap2 paired
Giraffe primary paired
Giraffe full paired
Giraffe full single
Giraffe primary single
Minimap2 single
HISAT2 paired
HISAT2 single
fast Giraffe full paired
fast Giraffe full single
0
10
20
30
40
50
Runtime (hours)


C 1000GP/GRCh38 NovaSeq 6000 Memory
GraphAligner
Out of memory
Giraffe full paired
Giraffe full single
Giraffe sampled paired
fast Giraffe sampled paired
Giraffe sampled single
fast Giraffe sampled single
Giraffe primary paired
Giraffe primary single
VG-MAP paired
VG-MAP single
Minimap2 single
Minimap2 paired
HISAT2* paired
HISAT2* single
BWA-MEM paired
BWA-MEM single
Bowtie2 paired
Bowtie2 single
0
20
40
60
80
100
Memory (GB)


D HGSVC/GRCh38 NovaSeq 6000 Memory
GraphAligner
Giraffe full paired
fast Giraffe full paired
Giraffe full single
fast Giraffe full single
Giraffe primary paired
Giraffe primary single
VG-MAP paired
VG-MAP single
Minimap2 single
Minimap2 paired
HISAT2 paired
HISAT2 single
BWA-MEM paired
BWA-MEM single
Bowtie2 paired
Bowtie2 single
0
20
40
60
80
100
Memory (GB)

Thank you!