Experiments on *de novo* assembly for nanopore sequencing data

Paolo Carnevali

June 2018

Chan Zuckerberg Initiative

Motivation

- Nanopore sequencing is emerging as a valuable DNA sequencing technology for some applications.
- Permits *de novo* assembly for human genomes, as shown in Jain *et al.*, Nanopore sequencing and assembly of a human genome with ultra-long reads, *Nature Biotechnology* **36**, 338–345 (2018) doi:10.1038/nbt.4060.
- If done for many genomes, could yield information on clinical implications of structural variants, currently not well studied because of limitations of short reads.
- However, the *de novo* assembly process is computationally expensive and logistically complex and could become a bottleneck, even as the actual sequencing becomes simpler and less expensive.

Current state of *de novo* assembly for nanopore data

- Current approach uses Canu, a fork of the old Celera assembler, developed $\sim$ 20 years ago for reads with very different error characteristics.
- Computationally expensive ($\sim 10^5$ core hours for a human genome) and logistically complex.
- Relies on a preliminary step that corrects the reads and incurs loss of information.

Wish list

Be able to run *de novo* assembly for a human genome:

- In $<$ 1 day elapsed time.
- At a computational cost $\sim$ \$100.
- Easily, without laborious job submissions on clusters.
- Without using an error correction step.
- Enable routine *de novo* assembly for clinical purposes.

Proposed computational platform

- Single multicore Linux machine, large memory ($\sim$ 512 GB).
- All computations in memory, using memory mapped data structures on large pages (2MB or 1GB) for reduced system overhead and improved TLB performance.
- Multithreaded parallelism at 2 threads per core (1 thread per ”virtual processor”), 16 to 64 cores.
- Simplifies development of high performance parallel code.
- Price per virtual processor hour remains contained for large memory machines.

Price per virtual processor hour

- On demand prices for AWS EC2, US-East (Ohio) region.
- Spot prices are often $\sim 3\times$ lower.
- Price per vCPU hour does not increase dramatically for large memory instances.

Instance	GB	vCPUs	$/h	$ / (vCPU $\times$ h)
m5.xlarge	16	4	0.19	0.05
m5.4xlarge	64	16	0.77	0.05
m5.24xlarge	384	96	4.61	0.05
x1.16xlarge	976	64	6.67	0.10
x1.32xlarge	1952	128	13.34	0.10

Data I am using for experimenting

- Data from Jain *et al.*, Nanopore sequencing and assembly of a human genome with ultra-long reads, *Nature Biotechnology* **36**, 338–345 (2018) doi:10.1038/nbt.4060.
- GM12878, coverage 40×, including ultralong reads.
- Base calling redone using Guppy 0.3.0 (courtesy M. Jain), input file rel5-guppy-0.3.0-chunk10k.fastq.
- More recent PromethION data are also available, but I decided not to use them for now because:
  - Coverage is much lower (7×).
  - Quality is lower.
  - No *de novo* assembly available that can be used for comparison.

The computational challenge

- Nanopore reads are long, but have high error rates ($\sim$ 5% raw read error rate).
- Errors include frequent deletions, occasional insertions, and are more frequent in homopolymer runs.
- k-mers of length sufficient to guarantee uniqueness in the human genome are usually affected by errors.
- Assembly algorithms can only rely on very short k-mers.

The basic idea

- Randomly choose a fixed subset of all possible 4^k k-mers of length k.
- Occurrences of these ”special” k-mers define *markers* in reads. Nearby markers can overlap.
- The sequence of markers R_i that occur in a read can be used as a succinct representation of the read.
- Use a MinHash approach to find overlapping reads, using as features groups of m consecutive markers.
- k must be small enough that a reasonable portion of k-mers are not affected by errors.
- In these experiments, I use as markers 10% of all possible k-mers with $k = 8$. A feature is defined as the consecutive occurrence of $m = 4$ markers in a read. These choices are not necessarily optimal.

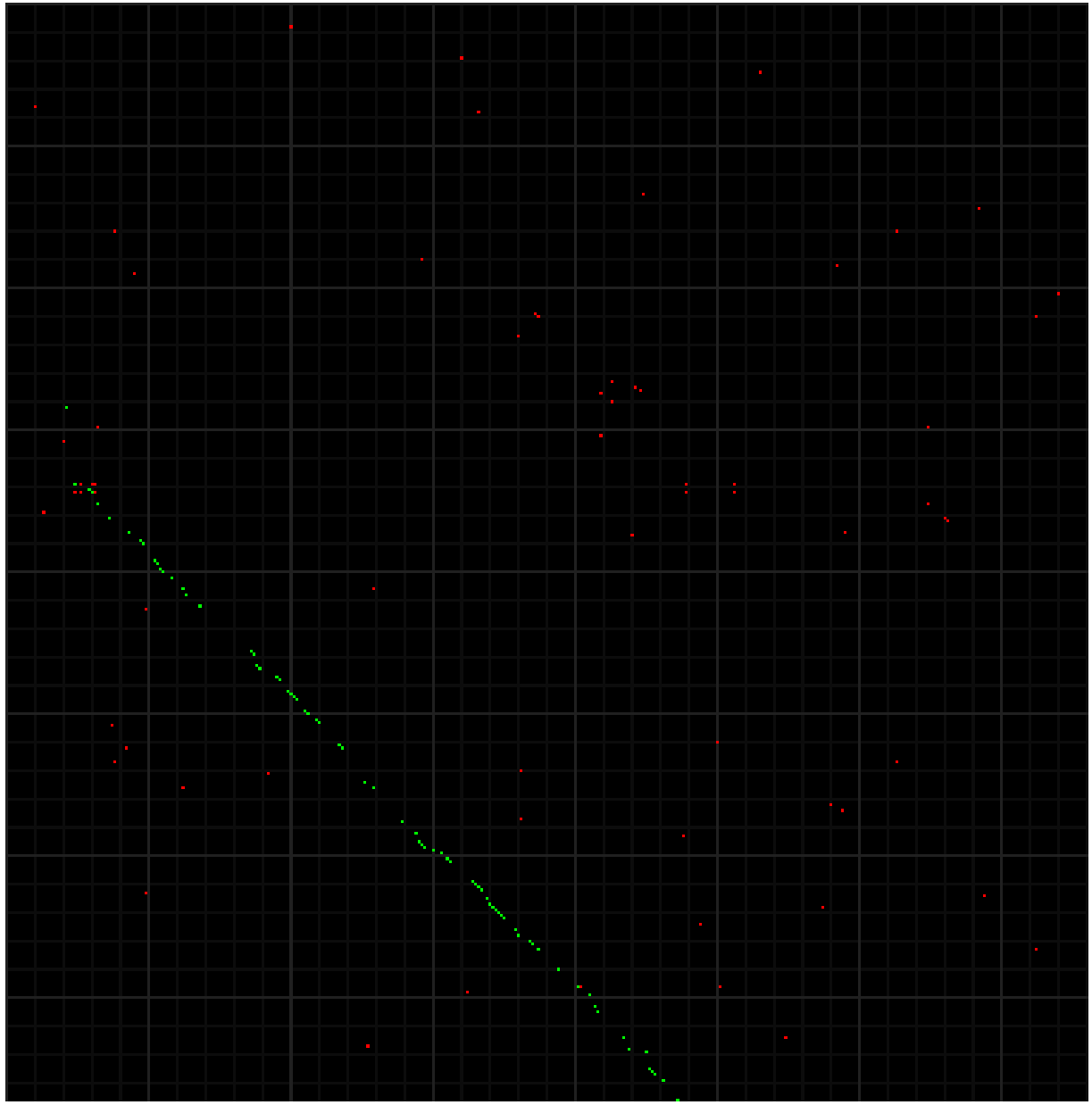
Aligning reads

- The MinHash algorithm finds pairs of reads that have features in common, but for each pair we still have to check that the two reads form a plausible alignment, rather than just sharing some repeated sequence.
- Aligning reads can be expensive,but we can make it faster by using the marker representation of each read.
- Instead of aligning the base sequence of the two reads, align the marker sequence.
- Much faster, runs at an average 0.5 ms per alignment while running 64 threads on a 64 vCPU machine (additional speed up possible).

Alignment matrix

- Given two reads A with n markers A_i and B with m markers B_j.
- Define an $n \times m$ alignment matrix $M_{ij} = \delta(A_i, B_j)$
- For illustration purposes only - construction of the matrix is not needed to compute the alignment.

Alignment example



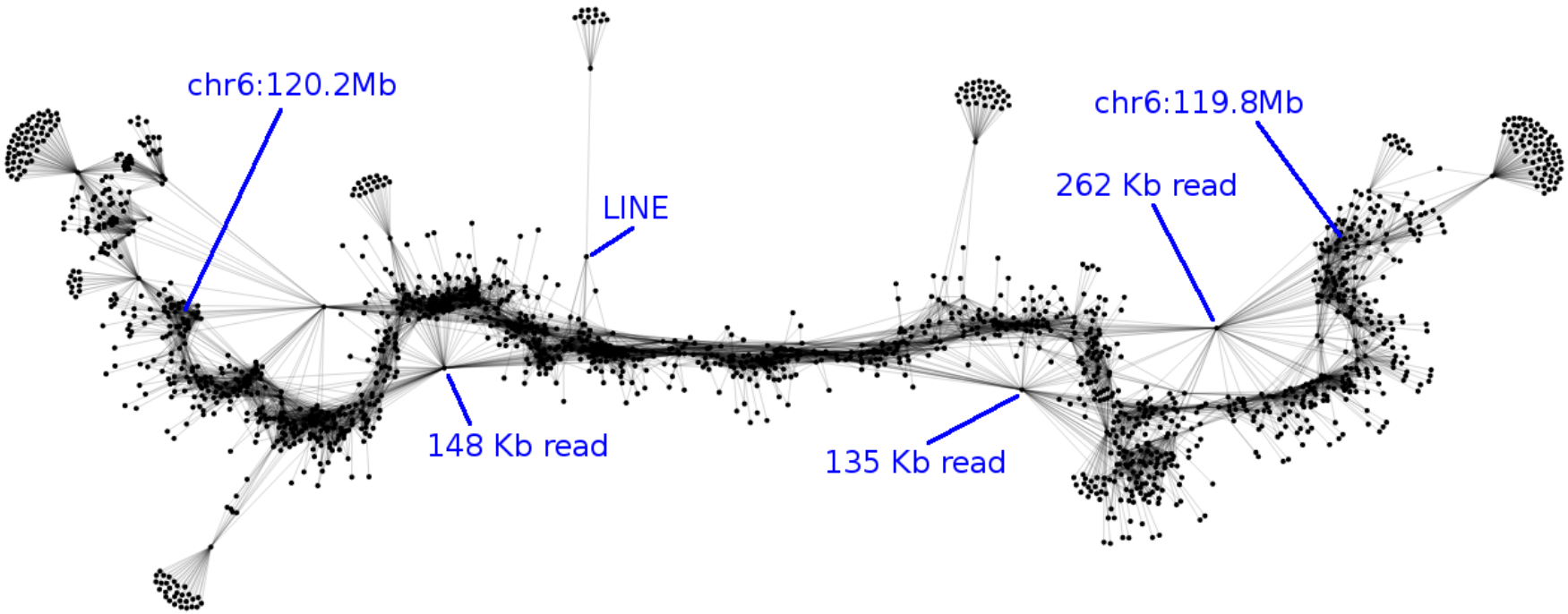
Finding pairs of overlapping reads

1. Run MinHash to find candidate pairs.
2. Compute alignments for all candidate pairs. Only keep pairs that have a sufficiently good alignment (number of aligned markers, small enough trim).

- The process took about 5 hours on a m4.16xlarge AWS EC2 instance ($\sim$\$0.70/h spot price).
- Spot checking of the read overlap graph indicates that the graph has a mostly one-dimensional structure.
- Some ”bridges” between long repeat copies are present and will have to be handled in the subsequent assembly process, or by using restrictive thresholds on the alignments accepted as good.

A small section of the read overlap graph

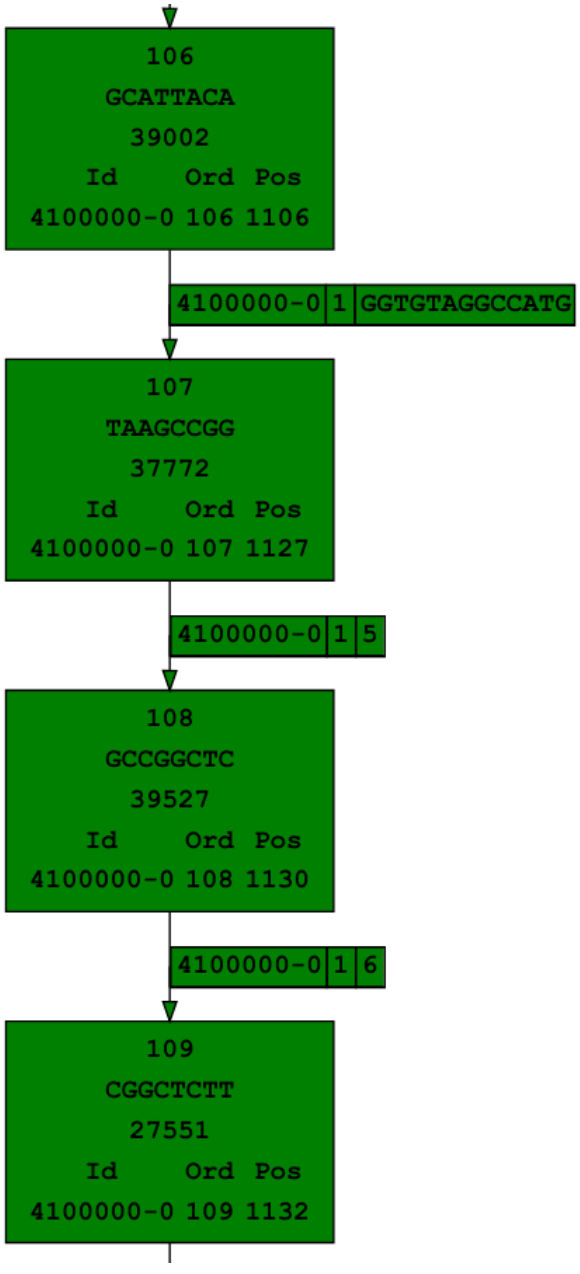
- Truncated at a maximum distance from a starting point.
- Covers $\sim$ 0.5 Mb of chr6.
- ”Bridges” to repeat regions visible.



More on the read representation using markers

- Represent a read as a marker graph - a linear chain in which vertices represent markers.
- Consecutive markers are joined by directed edges in the order in which they appear in the read.
- Each edge between two consecutive markers is labeled:
  - If the two markers do not overlap: with the read bases intervening between the two markers.
  - If the two markers do overlap: with the number of overlapping bases.

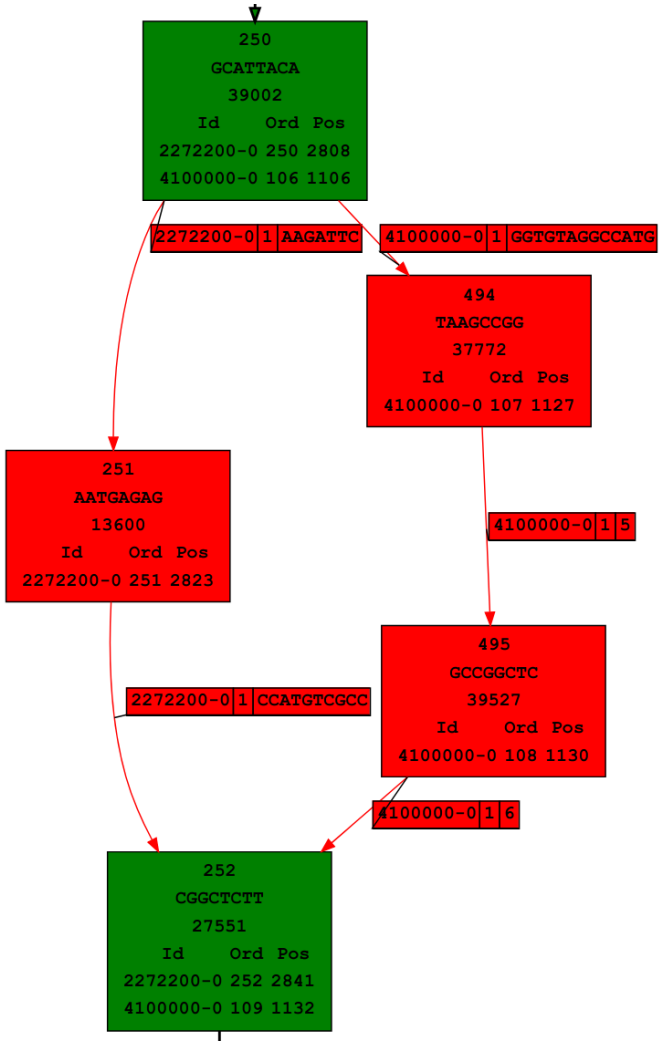
Portion of a marker graph for one read



We can reconstruct the read sequence from the graph:

```
GCATTACA
        GGTGTAGGCCATG
                     TAAGCCGG
                        GCCGGCTC
                          CGGGGCTCTT
GCATTACAGGTGTAGGCCATGTAAGCCGGGGCTCTT
```

Marker graph for two aligned reads

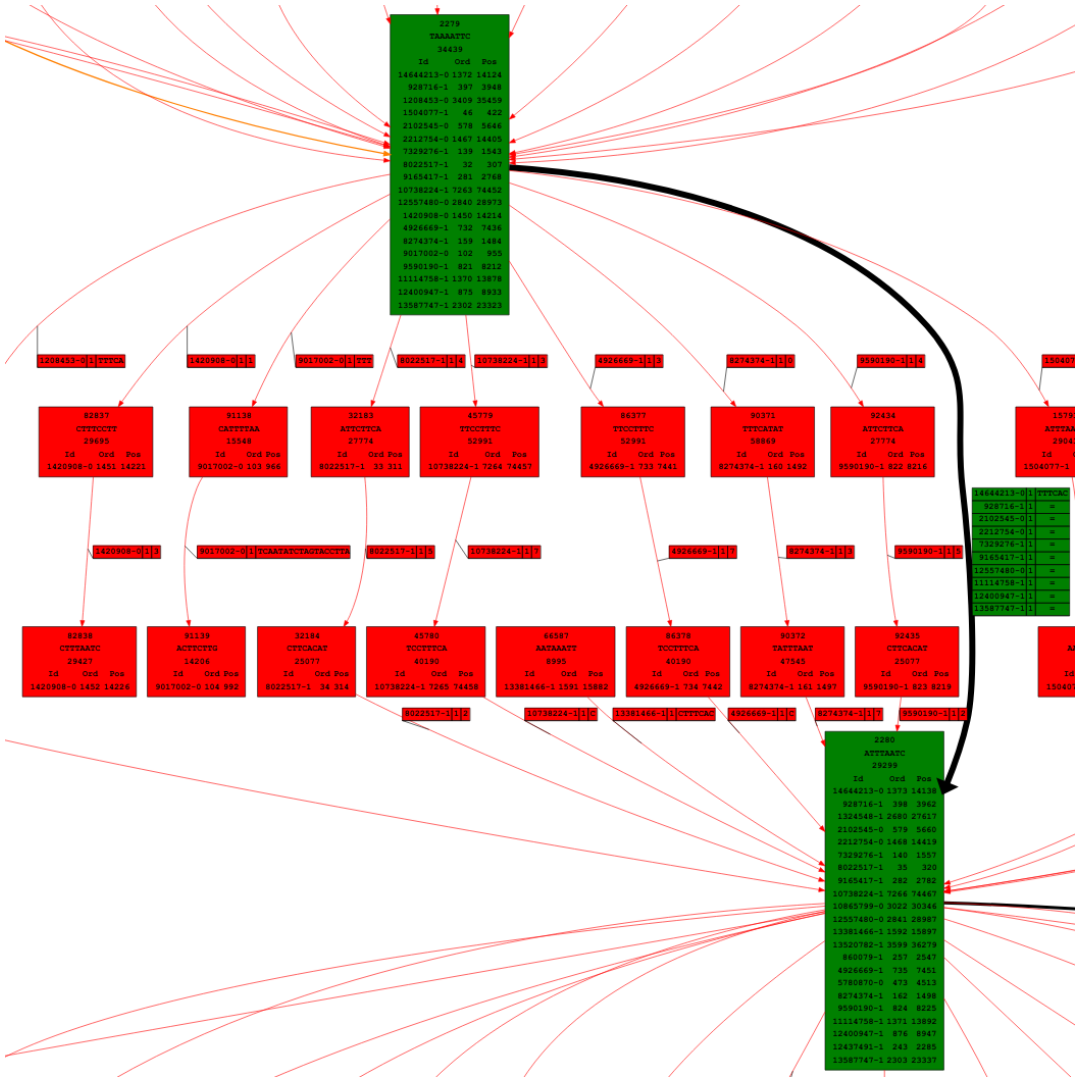


- If we have two reads for which a marker alignment was computed, we can merge the vertices corresponding to aligned markers.
- We can still reconstruct the read sequences by following in the graph the path corresponding to each read.

Marker graph for many aligned reads

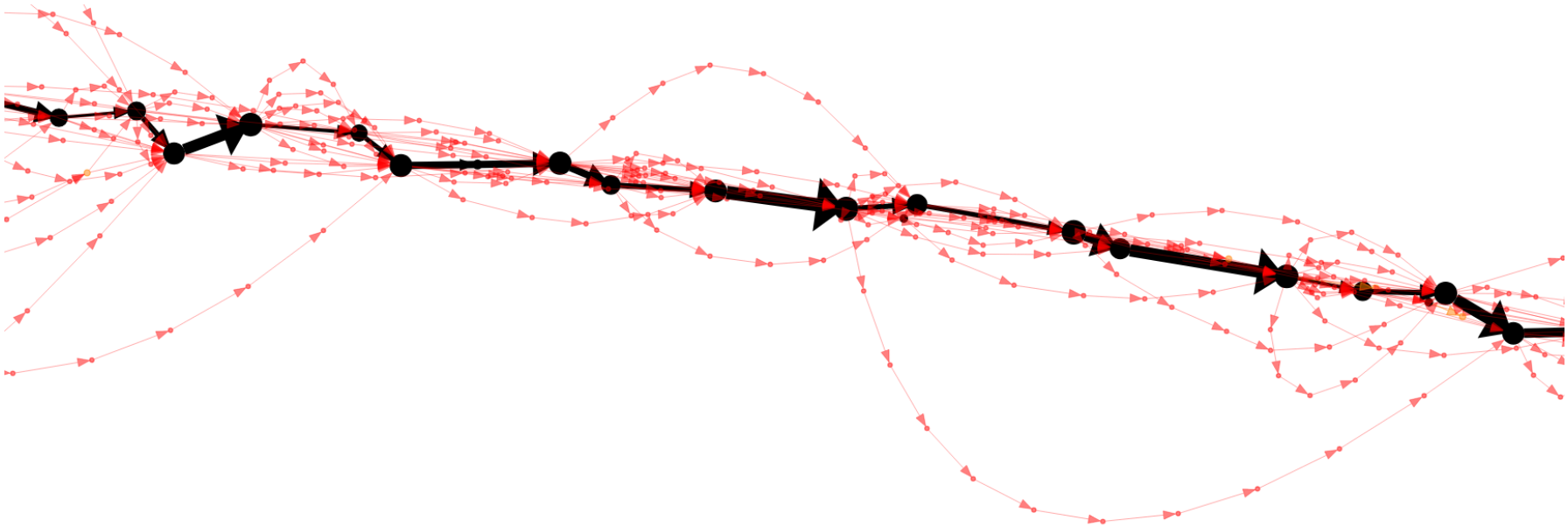
- We can similarly construct a marker graph for any number of aligned reads:
    1. Start with linear chains for each read.
    2. Merge pairs of vertices corresponding to aligned markers.
- The marker graph is a way to describe the multiple alignment between the reads.
- Get the alignments from the read overlap graph!
- It can be used for assembly:
    1. Follow paths that are supported by a sufficient number of reads.
    2. Extract sequence from each path.

Detail of marker graph for many aligned reads



- The top green marker is present in 19 reads.
- The bottomg green marker is present in 22 reads.
- 10 reads agree on the intervening sequence.
- Of the remaining reads, no two are in agreement, so they are probably all in error.

The marker graph on a larger scale



- Red vertices and edges are present in one read only.
- Well supported paths can be used to extract sequence for assembly.
- There are occasional breaks where there is no agreement among reads. A conservative assembler should no-call those locations.
- However these sequence breaks do not cause breaks in the scaffold.

Manual assembly

- Manual assembly of the well supported path in the previous slide gives 219 bases:

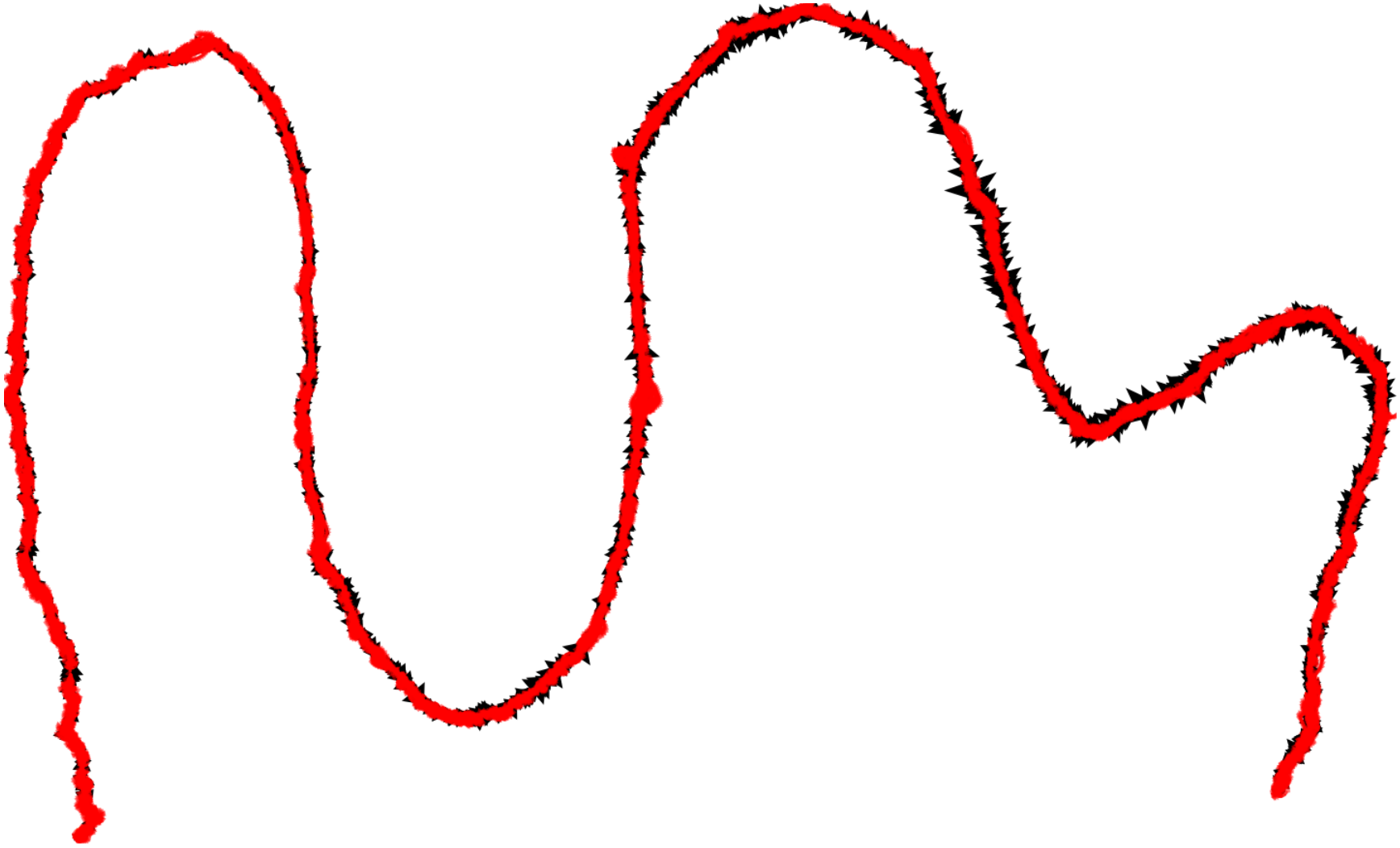
```
GCACTATGAGTTTCCCACAATGTCTTTATTTCTGTTAAAATTAATTCAATTATTTTCATT
TATTTGTATTCCTCCACATACACGATTTTATACAGCTTCCTCATCTAATTAATGGGACAC
AAACTCACCATATTAATGATGCCTTACATTTGTACAATGCTTTGCACAACATTTATTTCA
GAATCTCATTTGTGTTCACAAAAGTGAGAAACTTTGAAG
```

- They map to chrX, with a missing A from a 5-base homopolymer run:

SCORE	START	END	QSIZE	IDENTITY	CHRO	STRAND	START	END	SPAN
218	1	219	219	100.0%	X	+	142784211	142784430	220

- The marker graph shows that there are 5 reads with the expected number of A's, versus 10 reads with one missing A.

The marker graph on an even larger scale



Current prototype

- Already implemented efficiently on global scale:
  - MinHash.
  - Computation of alignments for all pairs found by MinHash.
  - Creation of global read graph.
- Implemented on local scale only:
  - Creation of local marker graph using alignments in the read graph.
  - Extraction of assembled sequence from the local marker graph (partial).
- Not yet implemented:
  - Creation of global marker graph.
  - Extraction of assembled sequence from the global marker graph.

Creation of the global marker graph

- The global marker graph has billions of vertices.
- Efficient implementation is a significant piece of work but possible.
- Use a disjoint-set data structure to merge aligned vertices.

Thank you for encouragement, support, insight, data

- Benedict Paten, UCSC
- Miten Jain, UCSC
- Bruce Martin, CZI