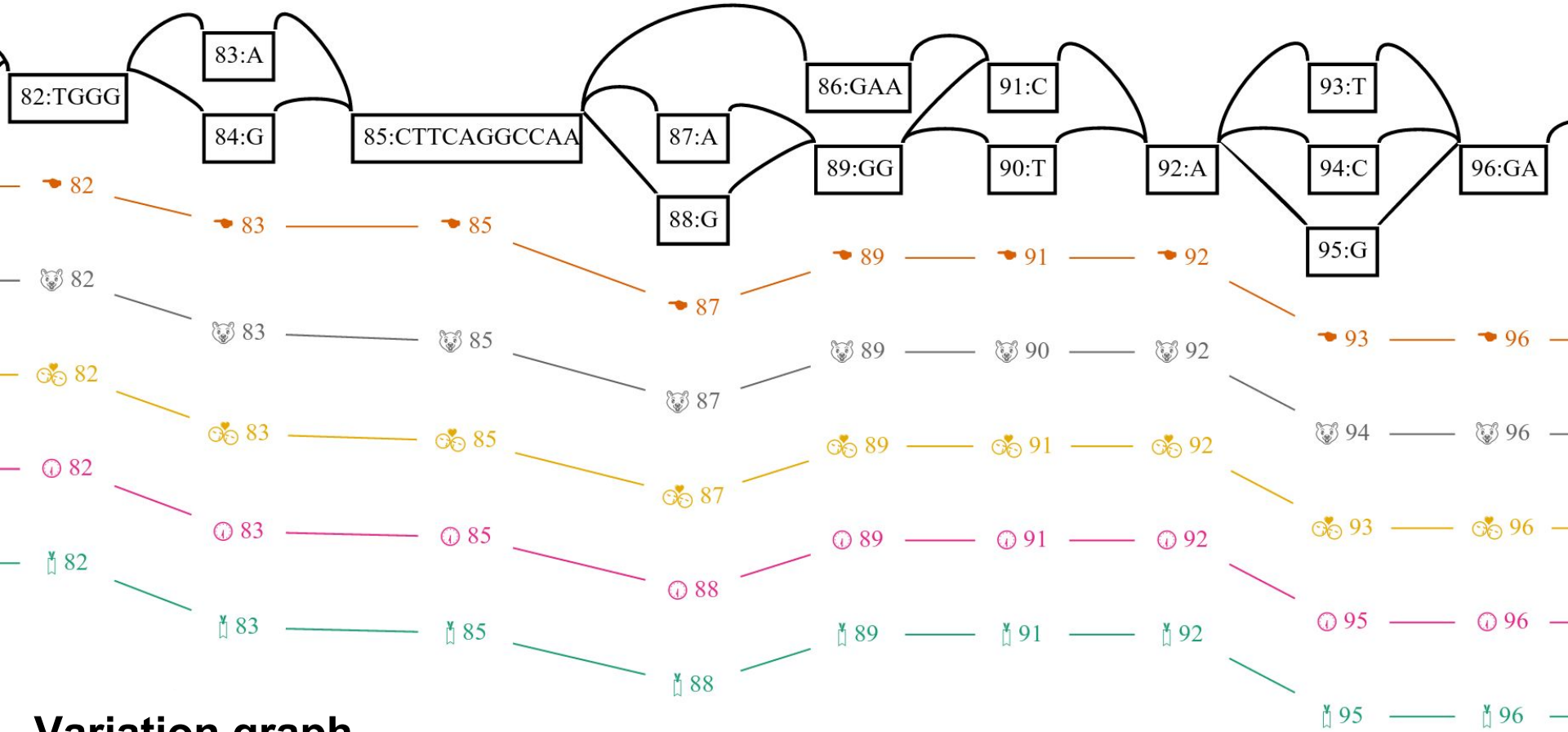
Computational Pangenomics #CPANG18

Day 2 (March 7, 2018)

Tobias Marschall, Erik Garrison, and Jordan Eizenga

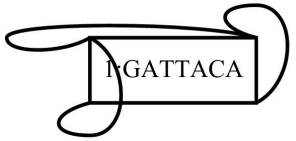
Wrap-up of day 1

- Variation graphs
- Data model
  - Graph, Node, Edge, Path, Mapping, Position, Edit, Alignment
- vg
  - construct, view, index, find, sim, map
- Practical discussion



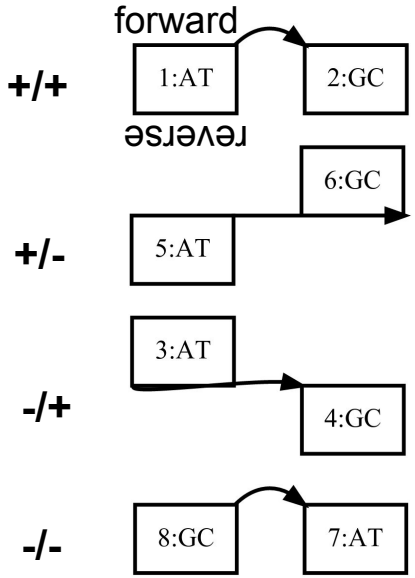
Variation graph

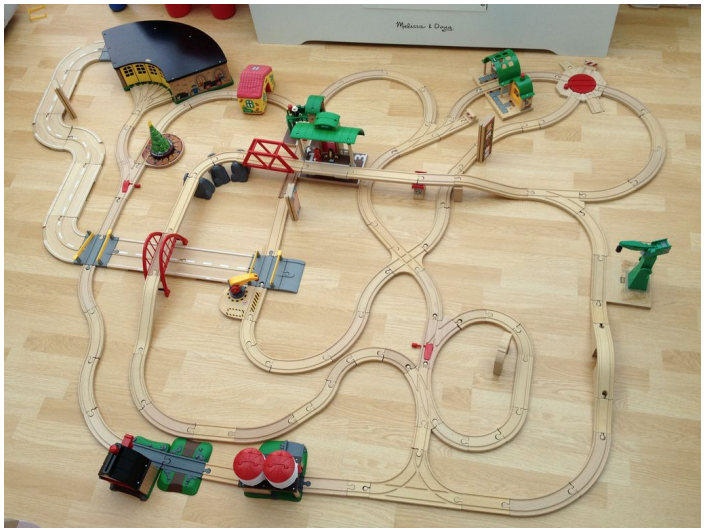
Train track graphs

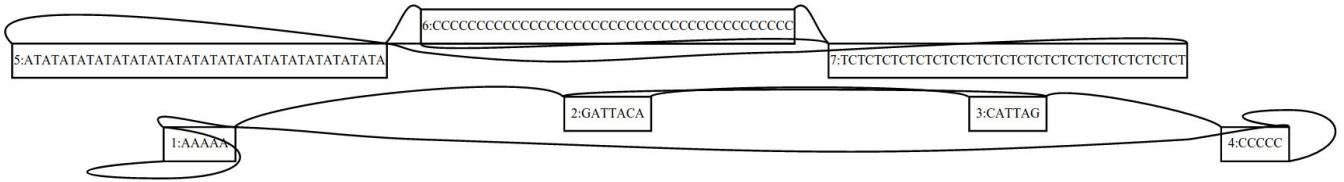


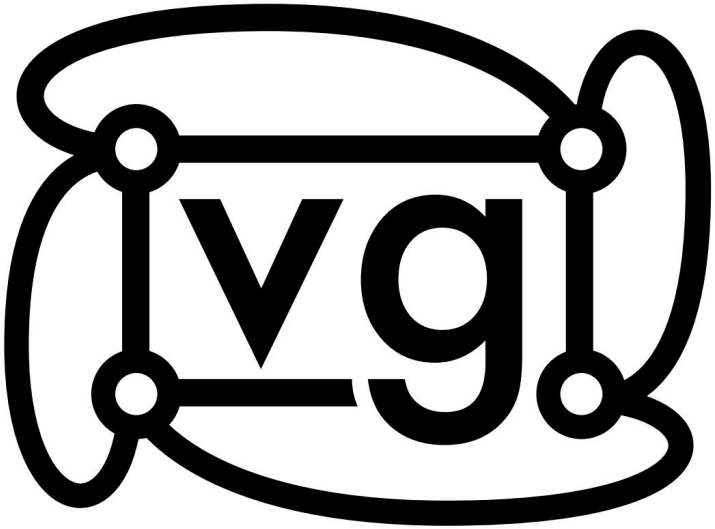
The graph is implicitly bidirectional, encoding both the forward and reverse complement.

Edges switching from the forward (+) to reverse (-) represent inversions.

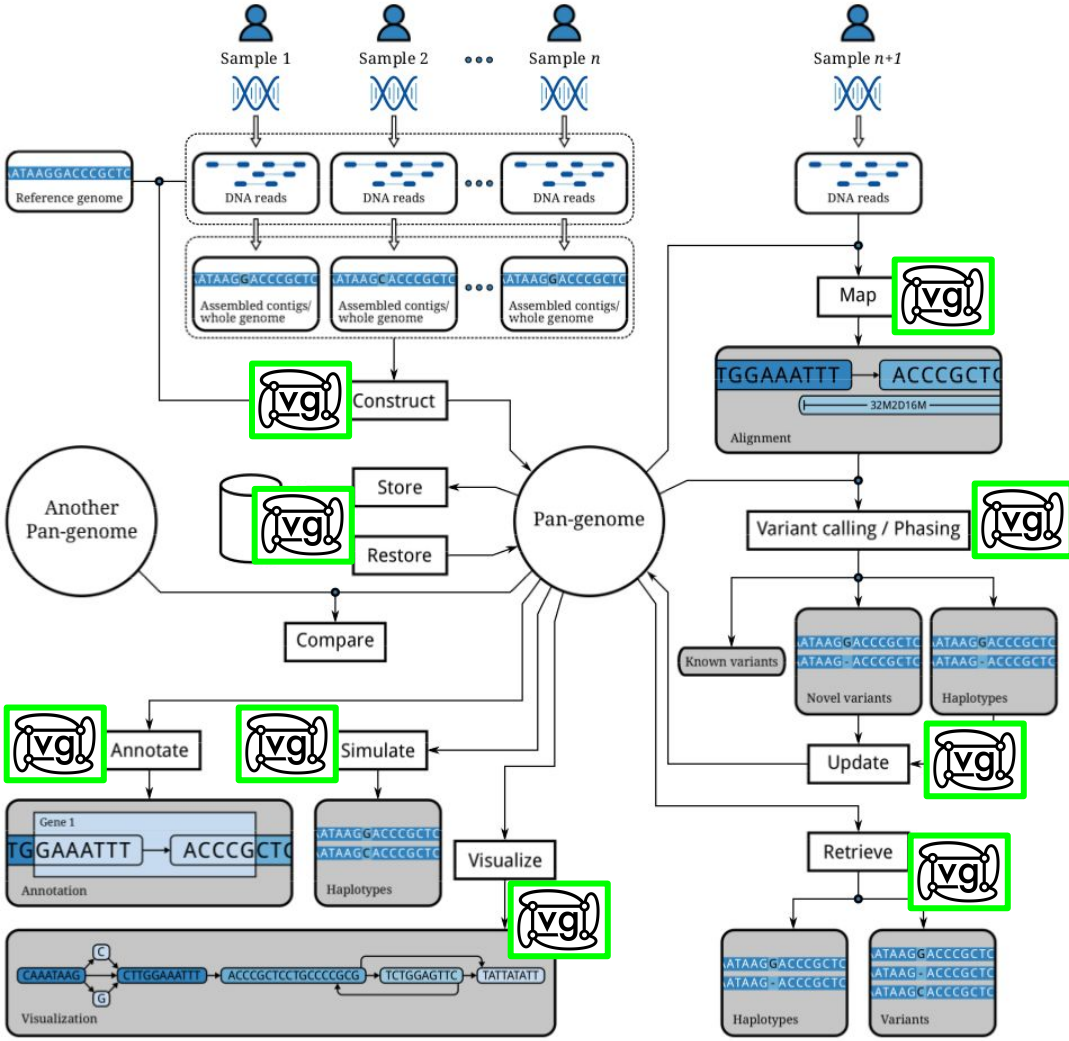






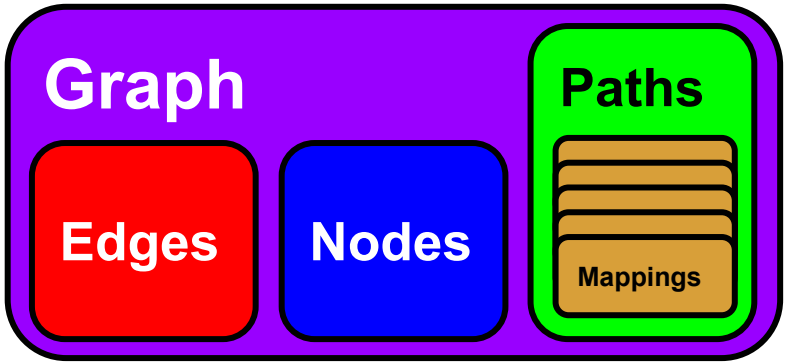


github.com/vgteam/vg



Data model

Basic entity is a *Graph*:



Implemented in vg using protobuf, JSON, RDF, and GFA

vg construct

tiny.fa

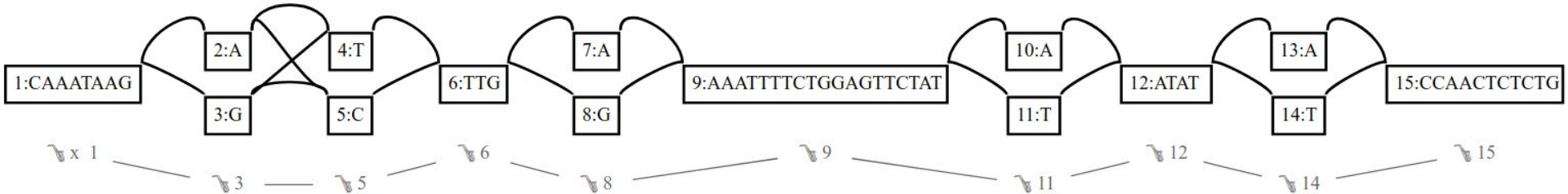
```
1:CAAATAAGGCTTGGAAATTTTCTGGAGTTCTATTATATTCCAACTCTCTG
```

tiny.vcf.gz

#CHROM	POS	REF	ALT
x	9	G	A
x	10	C	T
x	14	G	A
x	34	T	A
x	39	T	A

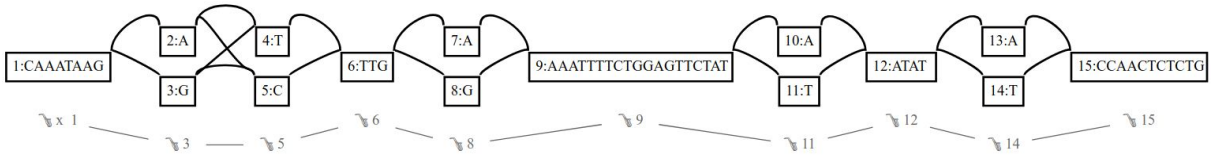
```
vg construct \
    -v tiny/tiny.vcf.gz \
    -r tiny/tiny.fa >tiny.vg
```

tiny.vg



vg index

tiny.vg



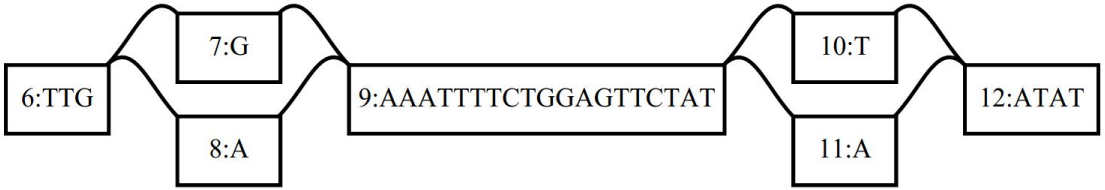

```
vg index tiny.vg \
    -x tiny.xg \
    -g tiny.gcsa -k 16
```

```
vg kmers -gk 16 tiny.vg | head -50
ATTTGGAAATTTTCTG        2:0     G       G       9:10
GTTTGGAAATTTTCTG        3:0     G       G       9:10
CAAATAAGATTTGAAA        1:0     #       A       9:2
GTTTGAAAATTTTCTG        3:0     G       G       9:10
ATTTGAAAATTTTCTG        2:0     G       G       9:10
GCTTGAAAATTTTCTG        3:0     G       G       9:10
TAAGATTTGAAAATTT        1:4     A       T       9:6
GCTTGGAAATTTTCTG        3:0     G       G       9:10
AATAAGATTTGAAAAT        1:2     A       T       9:4
CCTTATTTG$$$$$$$        3:-0    A,G     $       17:7
ACTTGAAAATTTTCTG        2:0     G       G       9:10
CTTGAAAATTTTCTGG        5:0     A,G     A       9:11
CAAATAAGATTTGGAA        1:0     #       A       9:2
CTTGGAAATTTTCTGG        5:0     A,G     A       9:11
AAATAAGATTTGAAAA        1:1     C       T       9:3
```

vg find

```
vg find -x tiny.xg \
    -p x:20-25 -c 1 \
   | vg view -d -
```

Query the nodes around x:20-25 in the reference path “x”.



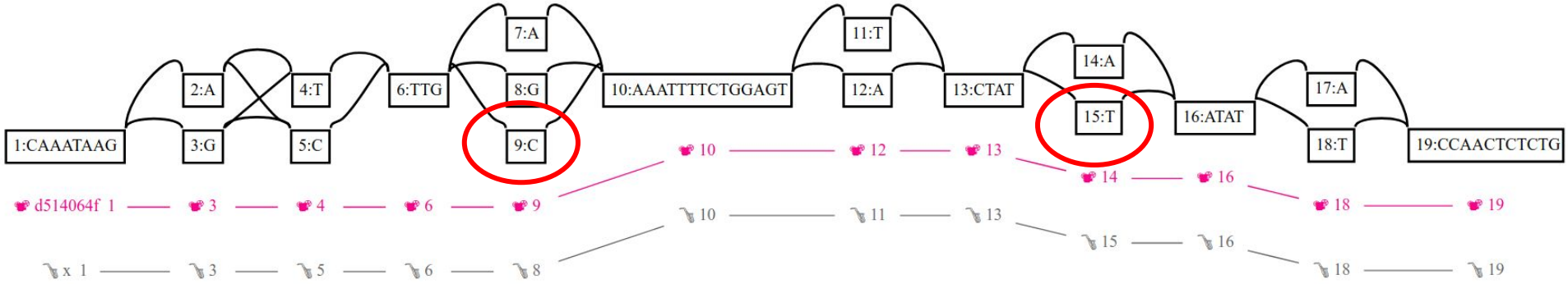

```
vg find -g tiny.gcsa \
    -S TCCAGAAAATTTTCAA
→ 9:-7
```

Query the position of a particular sequence in the GCSA2 index.

vg sim

Use a haplotype representing some variants relative to the tiny.vg to build a new graph:

```
vg msga -g tiny.vg -Nz \
    -s CAAATAAGGTTTGCAAATTTTCTGGAGTACTATAATATTCCAACTCTCTG \
    >truth.vg
```



We can then use it as a generative model and sample reads from it:

```
vg sim -l 50 -n 10 -s 1337 -x truth.xg >truth.reads
```

vg map

```
vg map -x tiny.xg -g tiny.gcsa -T truth.reads >aln.gam
```

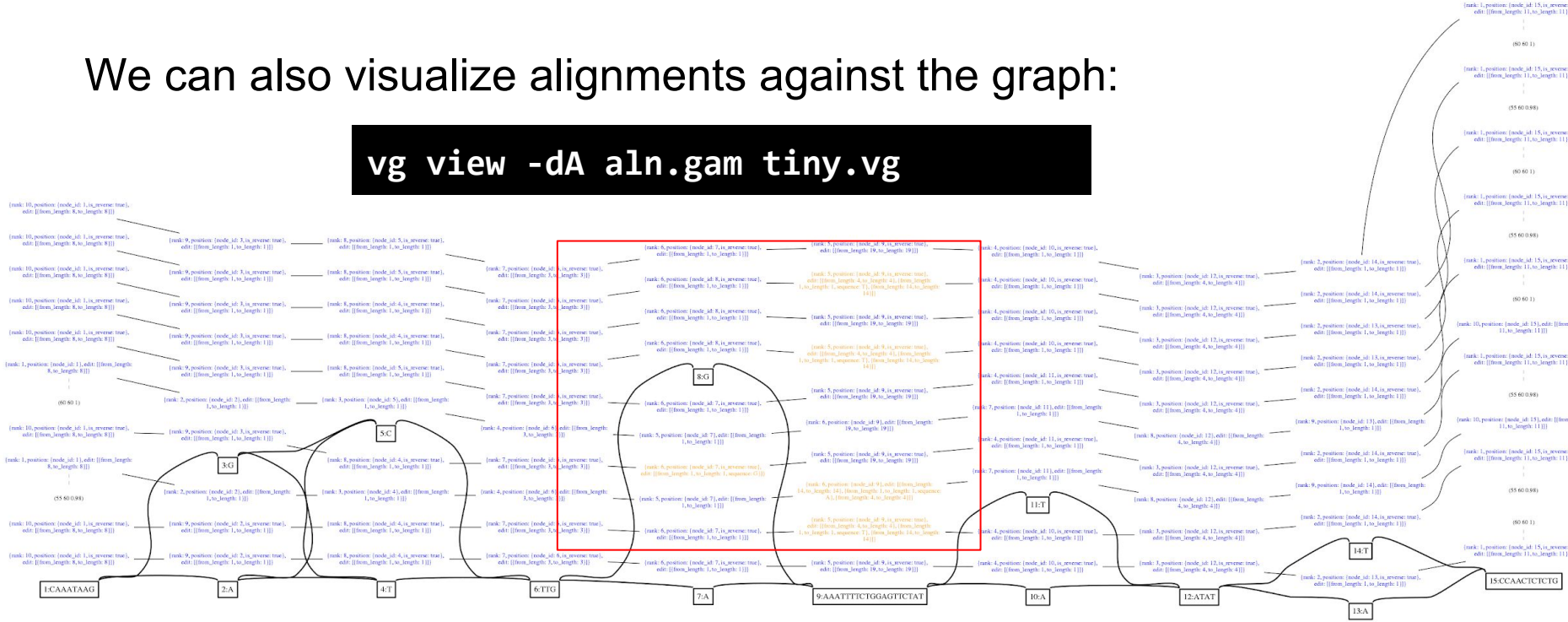

```
{
  "sequence": "CAGAGAGTTGGTATATTATAGAACTCCAGAAAATTTCCAAACCTTATTTG",
  "identity": 1,
  "path": {
    "mapping": [
      {
        "position": {
          "node_id": 15,
          "is_reverse": true
        },
        "edit": [
          {
            "from_length": 11,
            "to_length": 11
          }
        ],
        "rank": 1
      },
      {
        "position": {
          "node_id": 13,
          "is_reverse": true
        },
        "edit": [
          {
            "from_length": 1,
            "to_length": 1
          }
      .....
```


```
vg view -a aln.gam
```

alignment viz

We can also visualize alignments against the graph:

```
vg view -dA aln.gam tiny.vg
```



Blue represents perfect match.
Yellow represents a mismatch.

Questions

How comfortable are you with the concept of a pangenome?

How well do you understand vg construct?

How comfortable are you with the concept of indexing a pangenome?

Do you feel that you could set up and evaluate a read mapping pipeline in vg?

Today, how easy would it be for you to use the vg data model in your own project?

Day 1 practical results

What did we find?

Nano-presentations by various groups on their results.

Discussion.

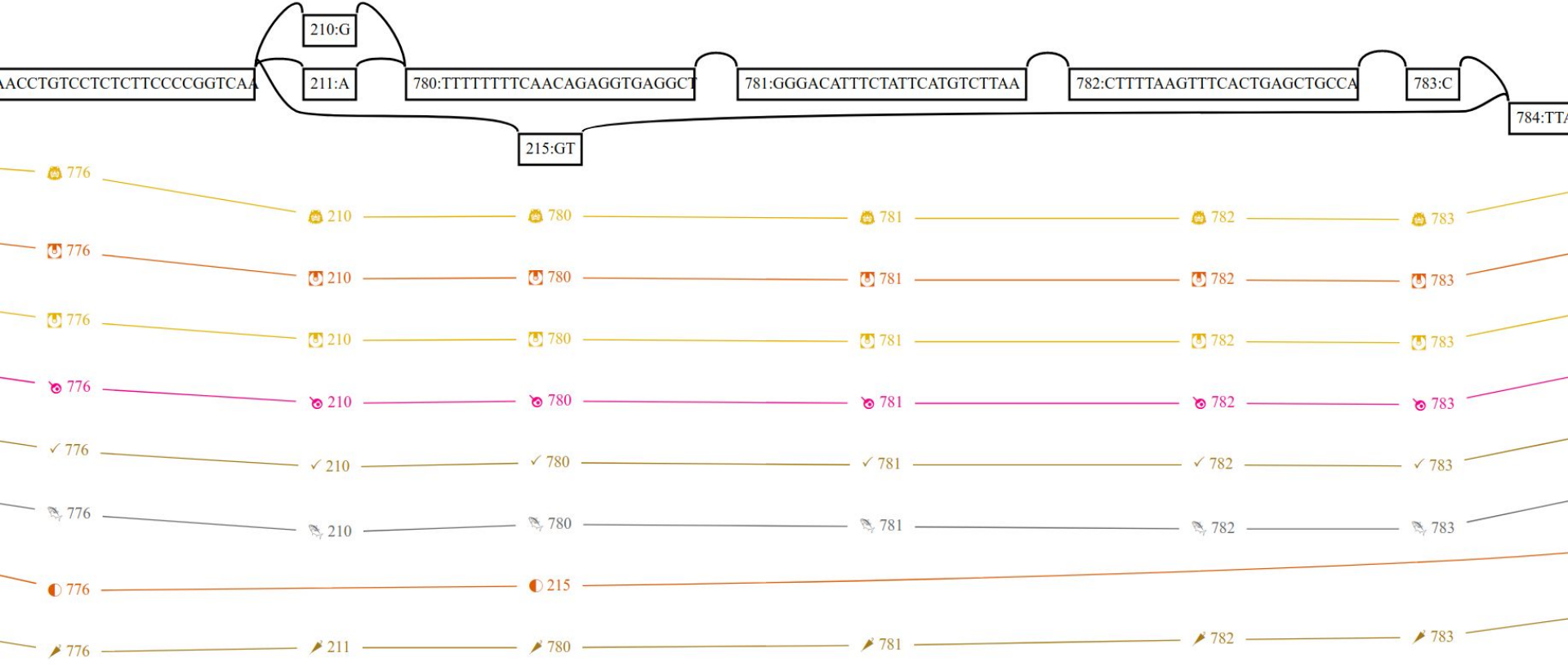
Day 2 proper

New commands: msga, surject, vectorize, mod, augment

vg msga

**vg msga -f GRCh38_alts/FASTA/HLA/L-3139.fa -D **
| vg mod -U 10 - | vg mod -X 32 -c - >L-3139.vg

vg view -dp L-3139.vg



vg surject

vg map -x z.xg -g z.gcsa -G z.sim | vg surject -d z.xg >aln.bam

or

vg map -d z -G z.sim --surject-to bam >aln.bam

Select the path to surject into with --into-path or --into-paths

vg vectorize

vg vectorize -f -x tiny.xg aln.gam

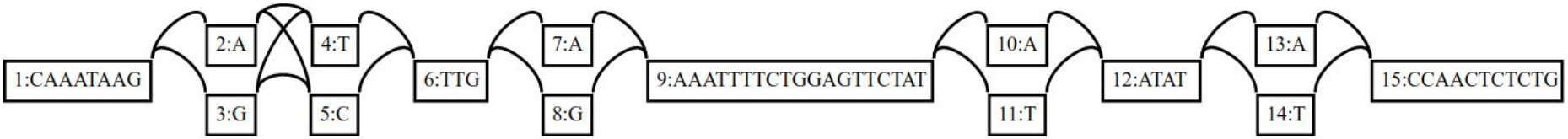
aln.name	node.1	node.2	node.3	node.4	node.5	node.6	node.7	node.8	node.9	node.10	node.11	node.12	node.13	node.14	node.15
d20030447889ddce	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1
617e3f3871de4388	1	1	0	1	0	1	0	1	1	0	1	1	0	1	1
47747b2abe90ed0c	1	0	1	1	0	1	0	1	1	0	1	1	0	1	1
37b9b60a8a5213ff	1	1	0	1	0	1	0	1	1	1	0	1	0	1	1
e5d31d6cd282cf8d	1	0	1	1	0	1	0	1	1	0	1	1	0	1	1
57dda702eaeb82c9	1	0	1	0	1	1	1	0	1	1	0	1	1	0	1
08343878ae5b90f3	1	0	1	0	1	1	1	0	1	0	1	1	0	1	1
757b525e41d48830	1	1	0	1	0	1	1	0	1	1	0	1	0	1	1
cd17bf40552fc5a2	1	0	1	1	0	1	0	1	1	0	1	1	0	1	1
1b8e295543bed0e8	1	1	0	1	0	1	0	1	1	0	1	1	1	0	1

vg mod

MANY graph modification tools in one command line utility.

- Sorting
- Chopping
- Simplification
- Augmentation
- Unfolding/unrolling
- Path manipulation (add, remove, keep)
- … etc, etc

vg mod -pl / vg prune

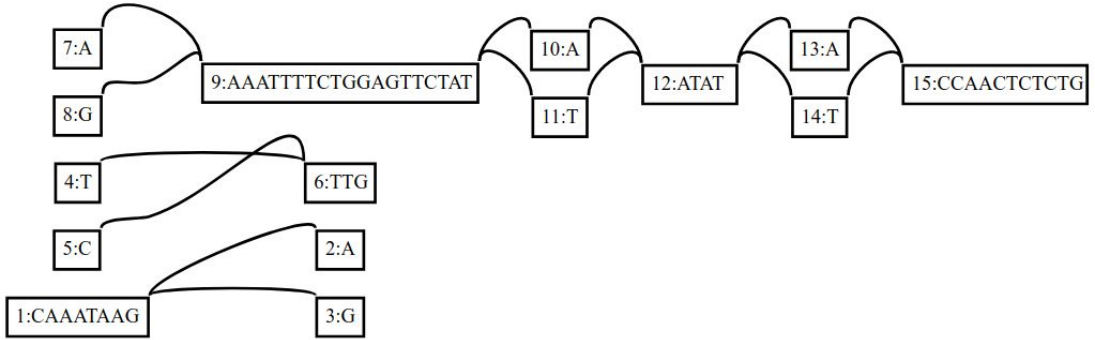


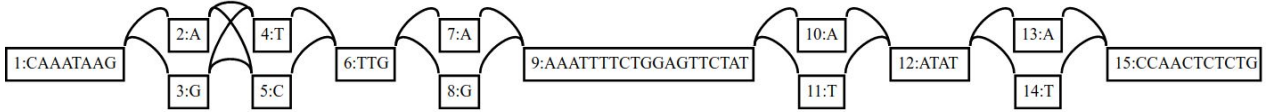
vg mod -pl 8 -e 2 tiny.vg

or

vg prune -k 8 -e 2 -s 0 tiny.vg

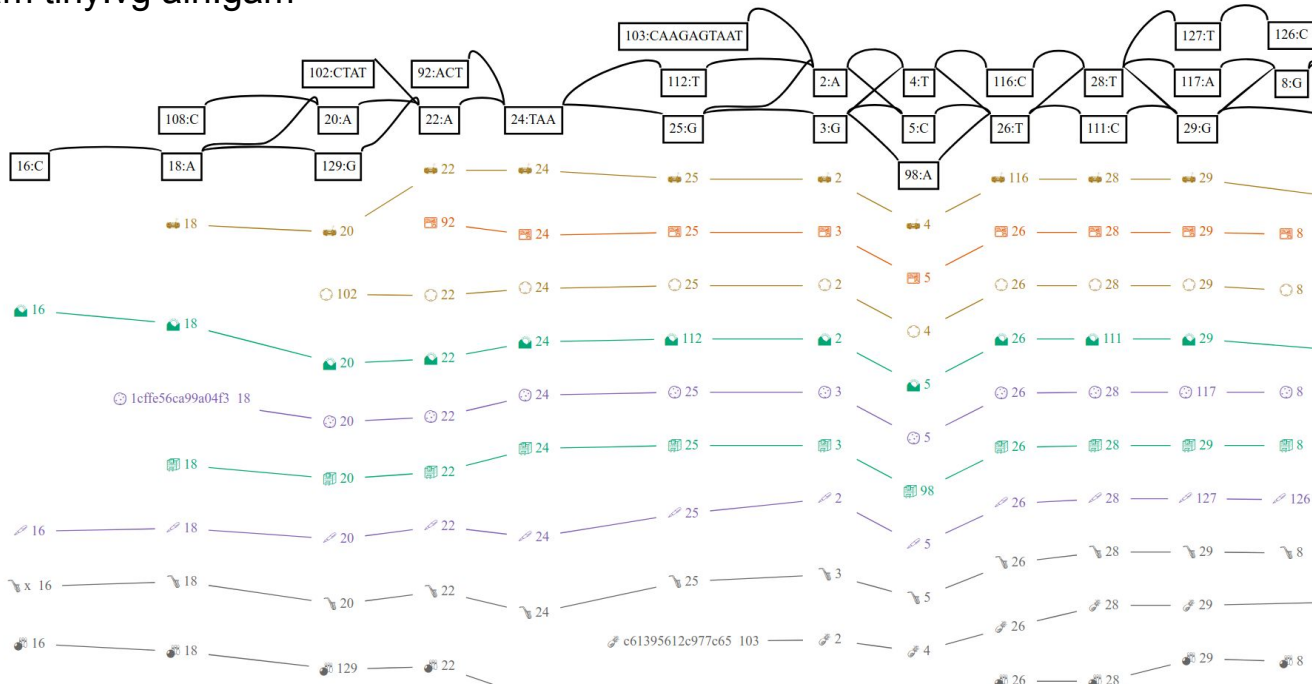
Removes edges for which we would have crossed 2 bifurcations in a path of 8 bases. (Used in indexing.)

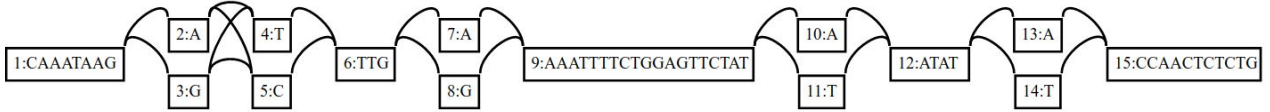




vg mod -i / vg augment

vg map -d tiny -G <(vg sim -n 10 -e 0.1 -i 0.05 -l 50 -a -x tiny.xg) >aln.gam
vg mod -i aln.gam tiny.vg >tiny+.vg
vg augment -g 1 -A aln+aug.gam tiny.vg aln.gam





vg mod -i / vg augment

vg map -d tiny -G <(vg sim -n 10 -e 0.1 -i 0.05 -l 50 -a -x tiny.xg) >aln.gam
vg mod -i aln.gam tiny.vg >tiny+.vg
vg augment -g 1 -A aln+aug.gam tiny.vg aln.gam