Sequence Assembly Graphs and their construction

(an introduction)

Camille Scott
Lab for Data Intensive Biology, UC Davis
presented at GraphXD, BIDS

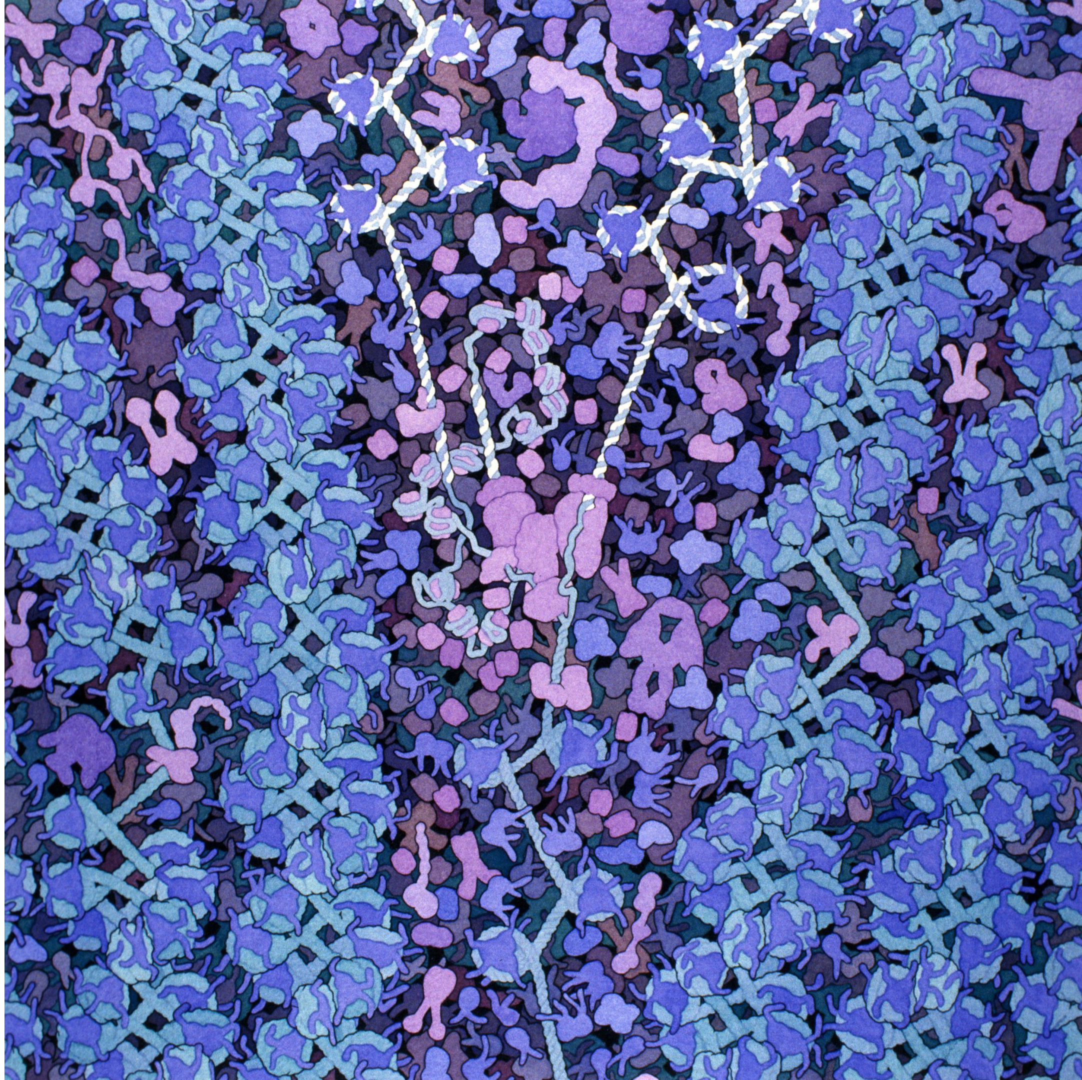
“Who are you and why are you here?”

- PhD student Titus’s Brown’s lab at UC Davis
- Studying streaming methods for constructing and analyzing assembly graphs
- Our lab has a general interest in assembly graphs, open science, teaching computational skills, scientific software sustainability

(because i was invited, or because my PI is currently frolicking in the redwoods with his family)

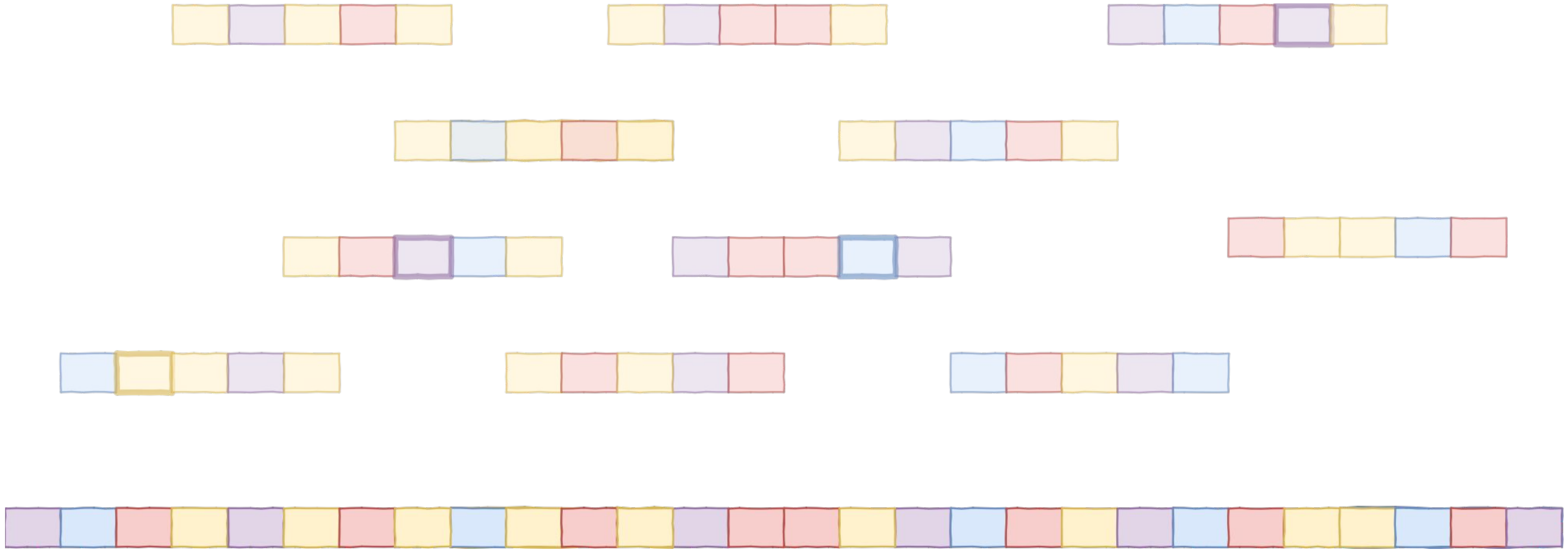
Motivating problem: how do we infer the sequence of symbols representing a molecule of DNA?

Illustration by David S. Goodsell, the Scripps Research Institute.



Generally, observing things damages them in some way: this limits how many nucleotides of sequence we can observe at a time before the molecule breaks down.

On most commonly used platforms, this means 100-250 bases; on some advanced/prototype platforms, up to 100Kb.

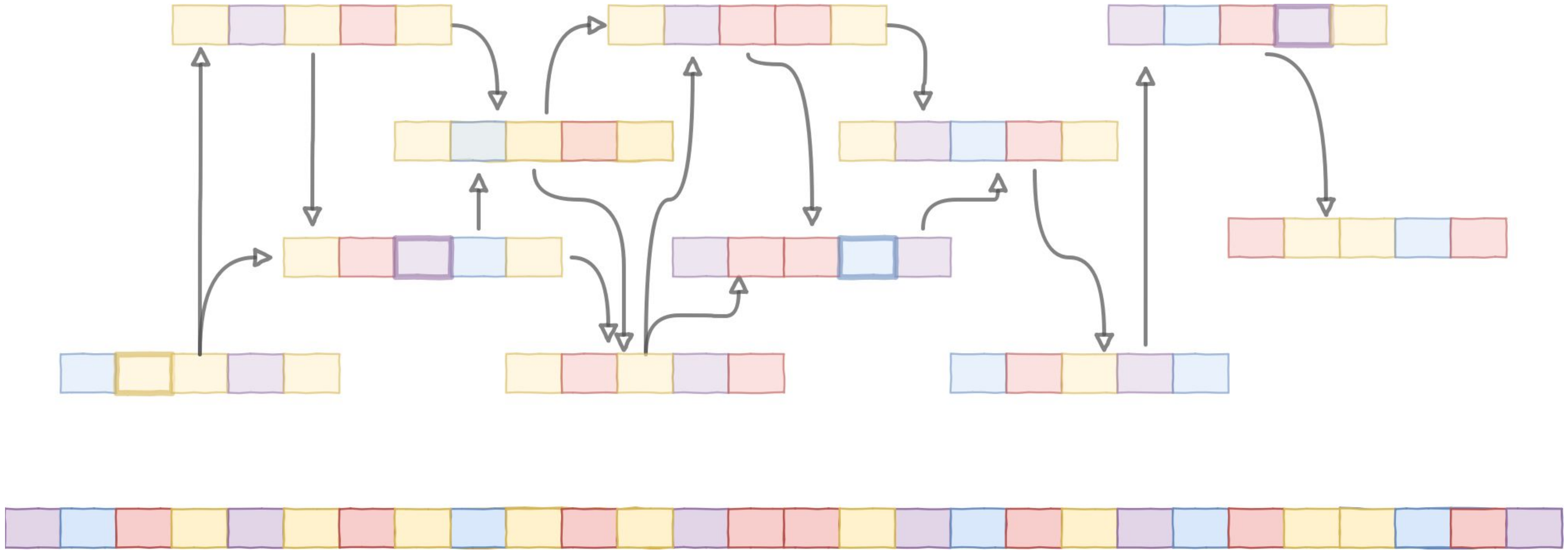


shotgun sequencing: randomly sample “reads” from underlying molecule

- Random sampling leads to high redundancy to guarantee coverage of most bases - 50-100x is common
- Sequencing process introduce some error epsilon
- For a single human-sized genome (3Gb) at 100x coverage, we’re dealing with 300Gb of raw string data.

“Sequence assembly” is the process of simplifying the random sampling down to some approximation of the underlying sequence it was sampled from.

- Gives rise to the “overlap-layout-consensus” approach.
    - Find overlaps between reads, represented as a graph (overlap)
    - Simplify graph as much as possible (layout)
    - Choose best option in conflicting regions (consensus)
- OLC approaches gave rise to the first **assembly graphs**

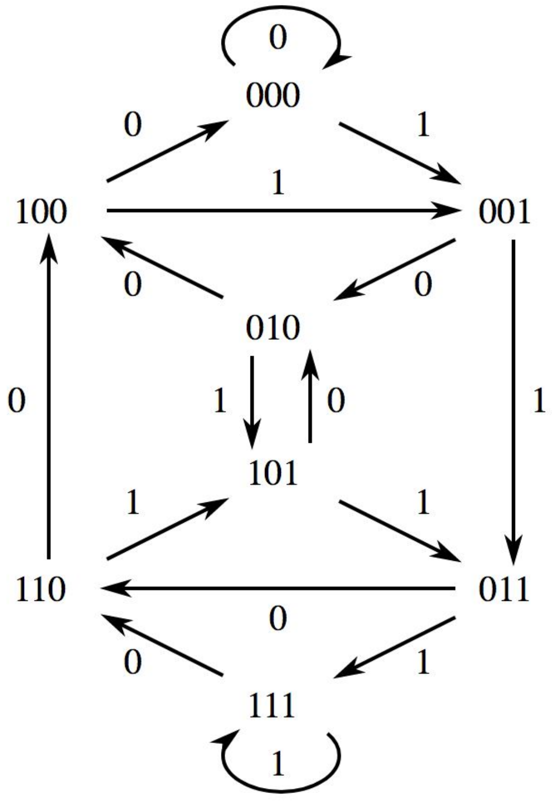


Overlap graph of sampled reads

- The naive approach is to do all-by-all sequence alignment between the reads...
- ...but on modern sequencers, producing 100 million to billions of reads, and alignment being **$O(n^2)$** for sequences of length **n**, this is absurdely impractical.
- Naturally, another graph comes to the rescue!

de Bruijn Graphs

- The complete dBG of order ***K*** and alphabet Σ is the dBG over all possible sequences of length ***K*** over Σ
- Two vertices $\mathbf{v_0}$ and $\mathbf{v_1}$ have a directed edge between them if the length ***K*-1** suffix of $\mathbf{v_0}$ matches the length ***K*-1** prefix of $\mathbf{v_1}$.
- These vertices are also called ***K*-mers**.
- Complete dBG is is Eulerian (has a path hitting every edge exactly once) and Hamiltonian (has a path hitting every node exactly once)



The complete dBG of order **K**=3 over Σ={0,1},

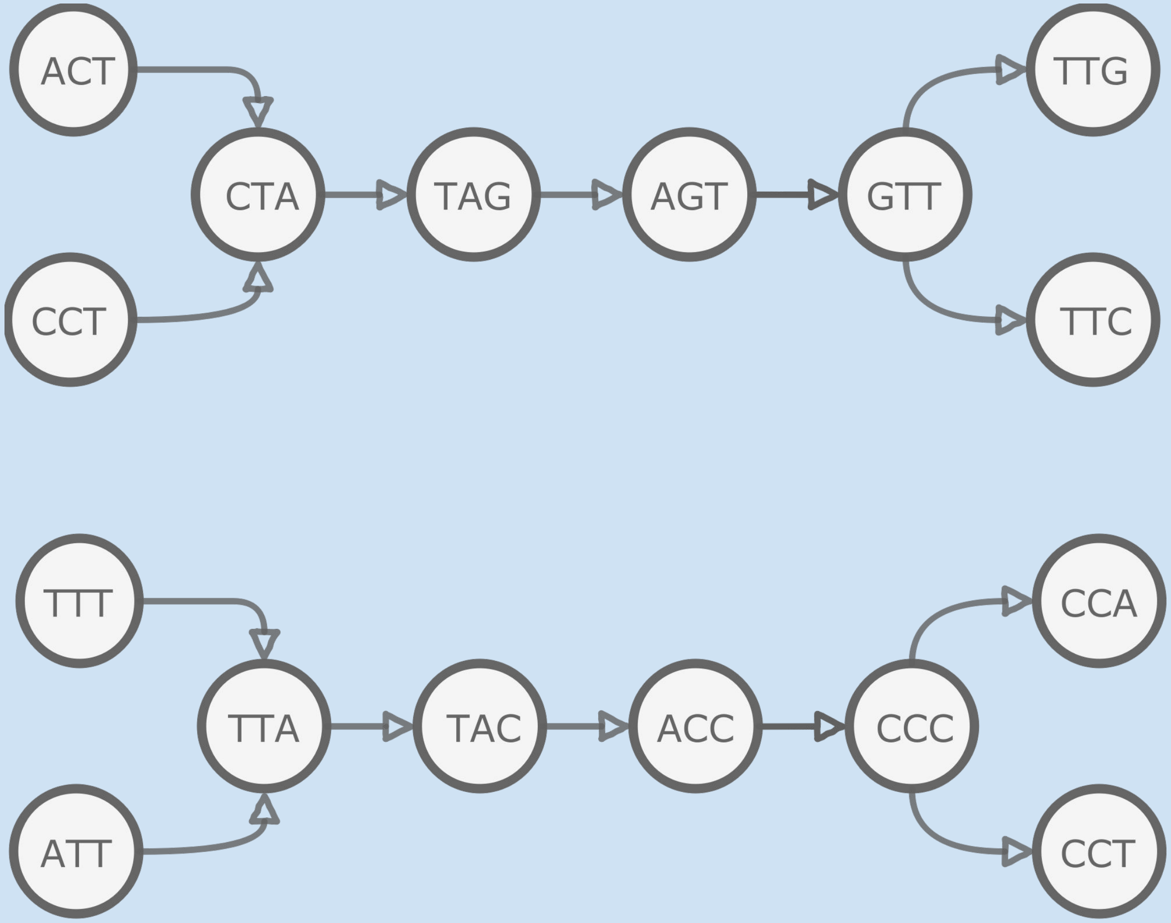
courtesy Wikipedia

An aside: this is one reason you usually have to enter a sentinal character after typing in a code for keypad locks.

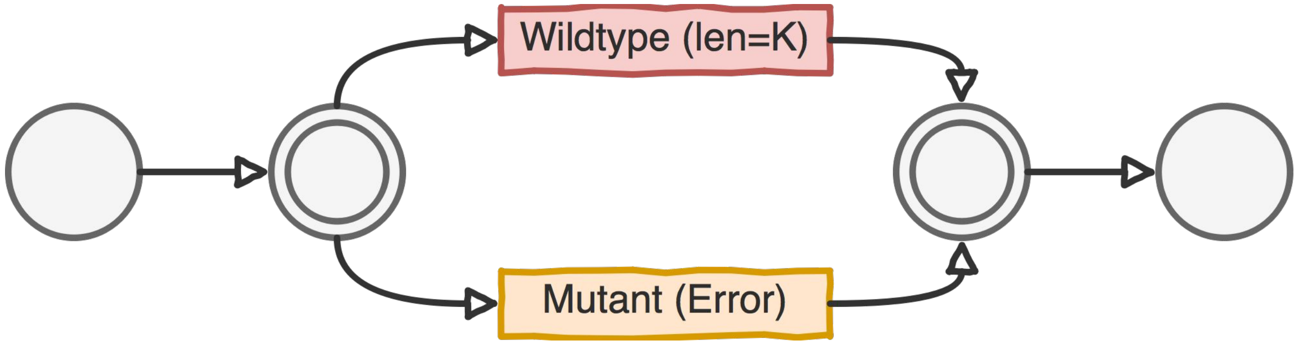
de Bruijn graphs have the convenient property of *implicitly defined edges*: given some vertex, I can find its neighbors by querying the set of vertices for the concatenation of its ***K*-1** prefix/suffix with each symbol from Σ

```
CGATTATCAGCAGTTTTTAAGAGCTTTGTTAGTCGCCCTGTCATCCGAGA
CGATTATCAGCAGTTTTTAA
 GATTATCAGCAGTTTTTAAG
  ATTATCAGCAGTTTTTAAGA
   TTATCAGCAGTTTTTAAGAG
    TATCAGCAGTTTTTAAGAGC
     ATCAGCAGTTTTTAAGAGCT
      TCAGCAGTTTTTAAGAGCTT
       CAGCAGTTTTTAAGAGCTTT
        AGCAGTTTTTAAGAGCTTTG
         GCAGTTTTTAAGAGCTTTGT
          CAGTTTTTAAGAGCTTTGTT
           AGTTTTTAAGAGCTTTGTTA
            GTTTTTAAGAGCTTTGTTAG
             TTTTTAAGAGCTTTGTTAGT
              TTTTAAGAGCTTTGTTAGTC
               TTTAAGAGCTTTGTTAGTCG
                TTAAGAGCTTTGTTAGTCGC
                 TAAGAGCTTTGTTAGTCGCC
                  AAGAGCTTTGTTAGTCGCCC
                   AGAGCTTTGTTAGTCGCCCT
                    GAGCTTTGTTAGTCGCCCTG
                     AGCTTTGTTAGTCGCCCTGT
                      GCTTTGTTAGTCGCCCTGTC
                       CTTTGTTAGTCGCCCTGTCA
                        TTTGTTAGTCGCCCTGTCAT
                         TTGTTAGTCGCCCTGTCATC
                          TGTTAGTCGCCCTGTCATCC
                           GTTAGTCGCCCTGTCATCCG
                            TTAGTCGCCCTGTCATCCGA
                             TAGTCGCCCTGTCATCCGAG
                              AGTCGCCCTGTCATCCGAGA
```

21-mers from sequence of length 50

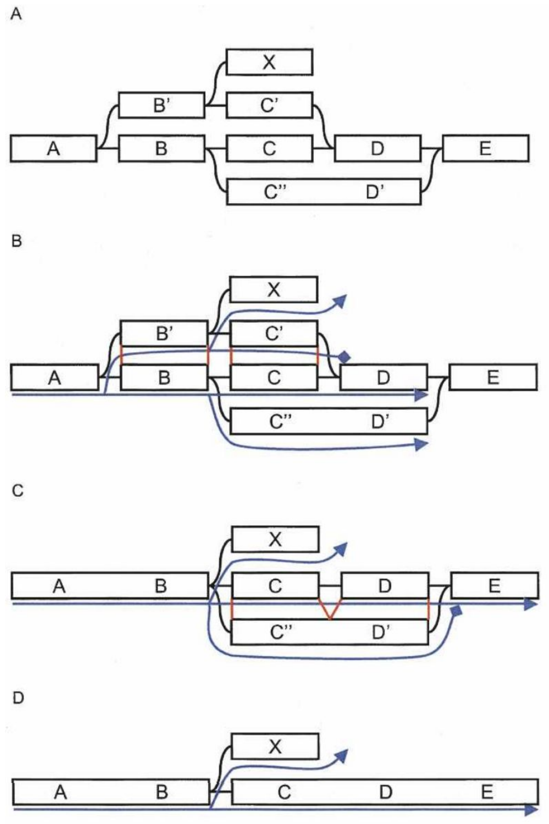

ACT
CCT
CTA
TAG
AGT
GTT
TTG
TTC
TTT
ATT
TTA
TAC
ACC
CCC
CCA
CCT

- The implict edge property means we solve a major problem for free: we also are implicitly finding all overlaps of length ***K*-1**.
- ***K*** is thus the dominating parameter for sequence de Bruijn graphs
    - too small, and we fail to uniquely localize subsequences
    - too large, and point errors remove most overlaps

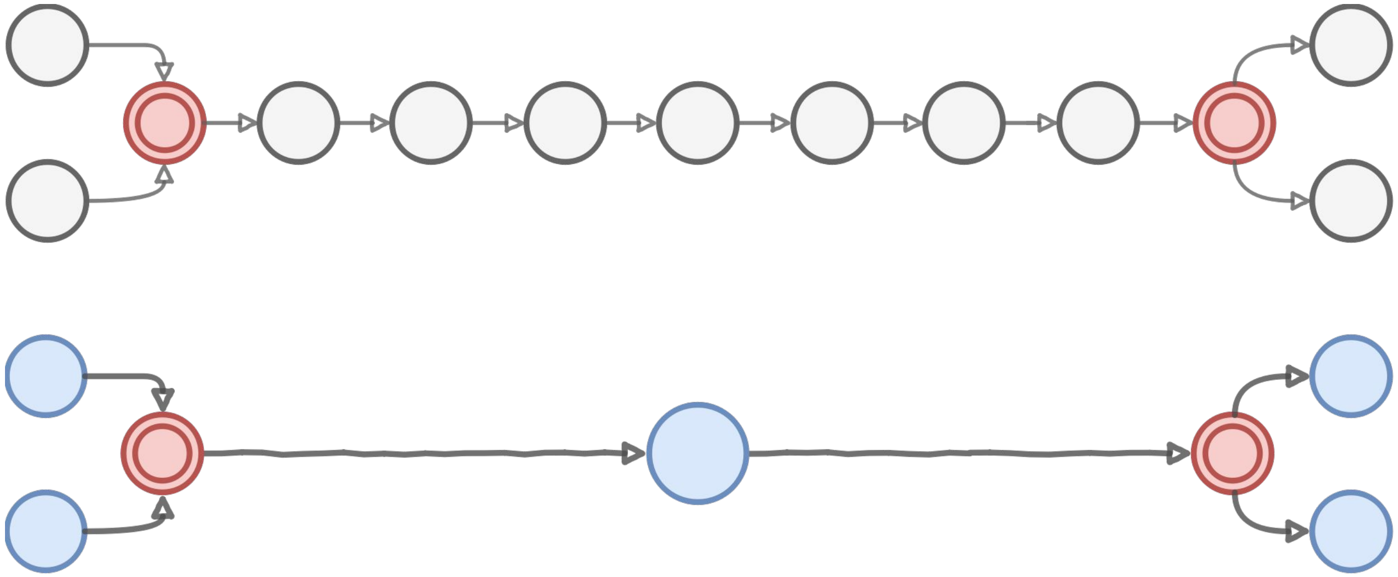


A point mutation (or error) introduces (up to) ***K*** new vertices to the dBG. We call this a bubble.

- After building the raw dBG, perform simplifying operations to reduce its size
- Bubble popping (based on coverage model)
- Tip removal
- Contraction of linear paths

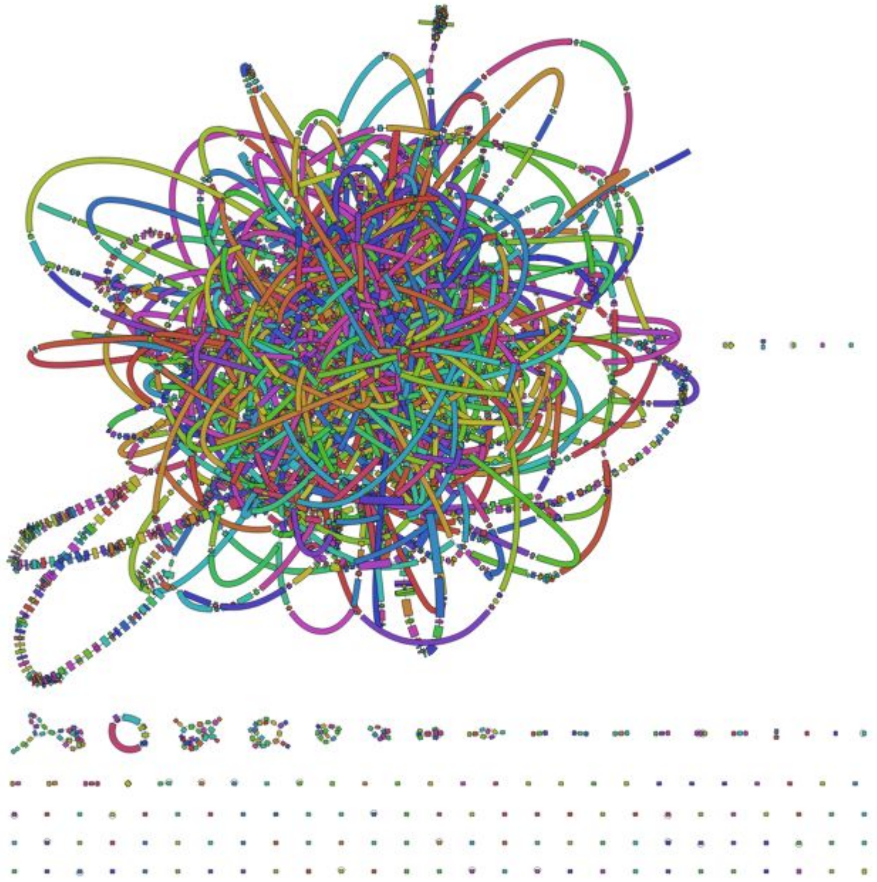


Zerbino, Daniel R., and Ewan Birney. "Velvet: algorithms for de novo short read assembly using de Bruijn graphs." Genome research 18.5 (2008): 821-829.

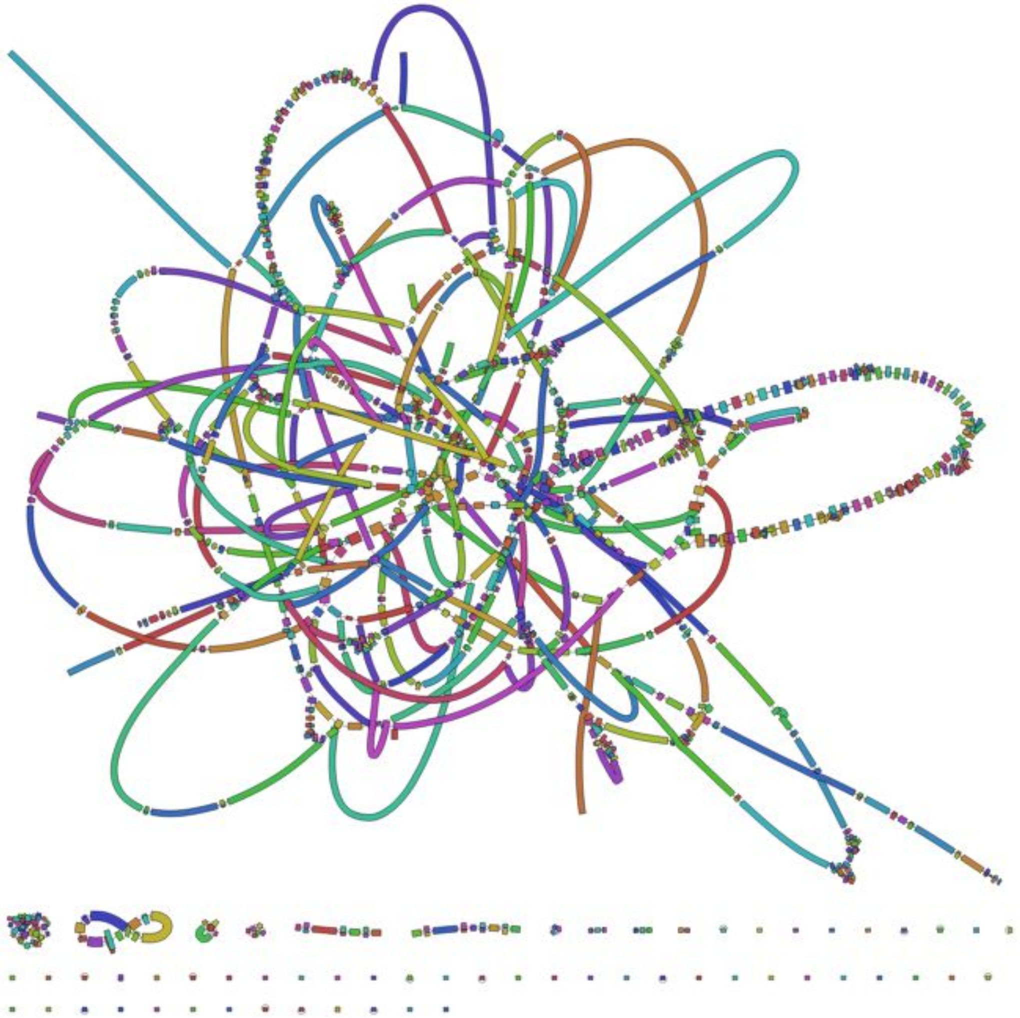


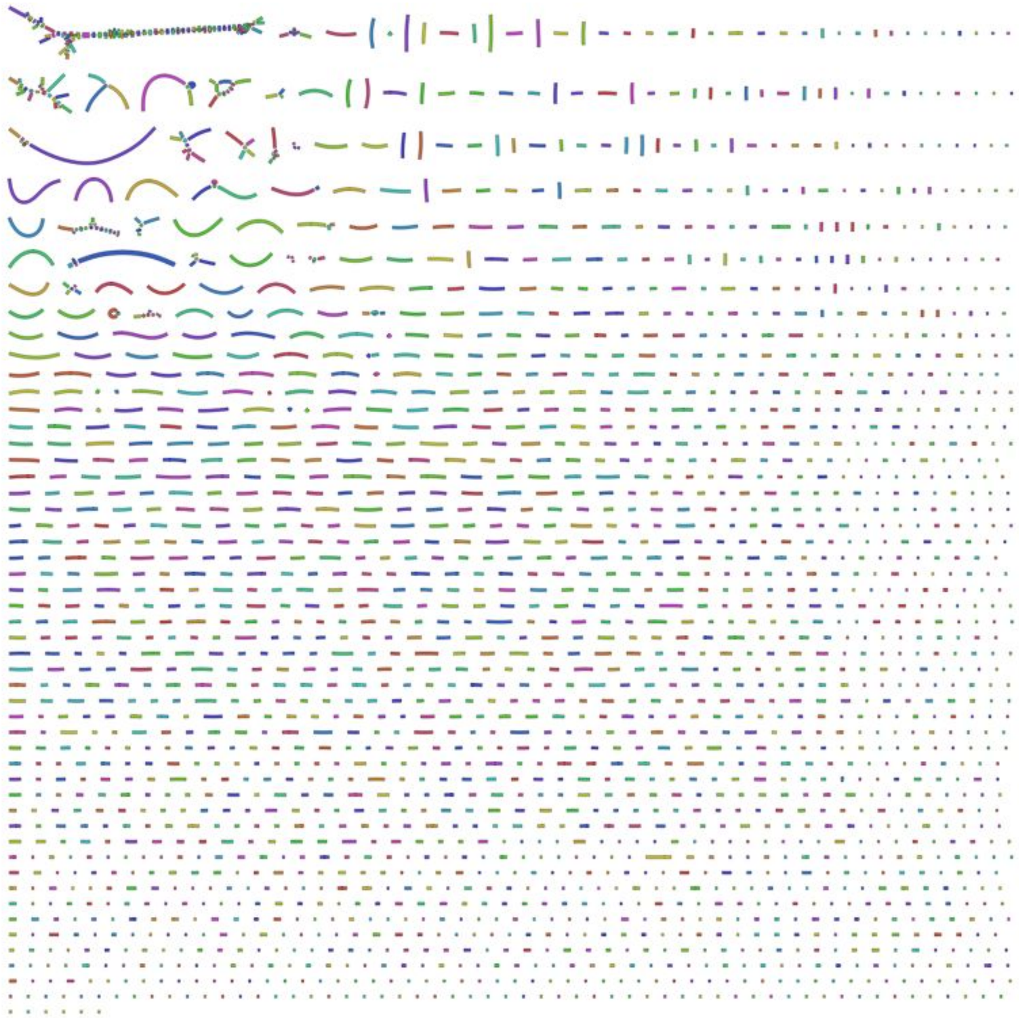
In a compact de Bruijn Graph (cDBG), all linear paths are reduced to single nodes storing their concatenated sequence.

- These linear paths are called contigs or unitigs (depending on who you ask), and directly represent portions of the underlying sequence
- The most basic assembler would output these unitigs, but actual assemblers perform additional heuristics to perform traversals across nodes of indegree or outdegree > 2

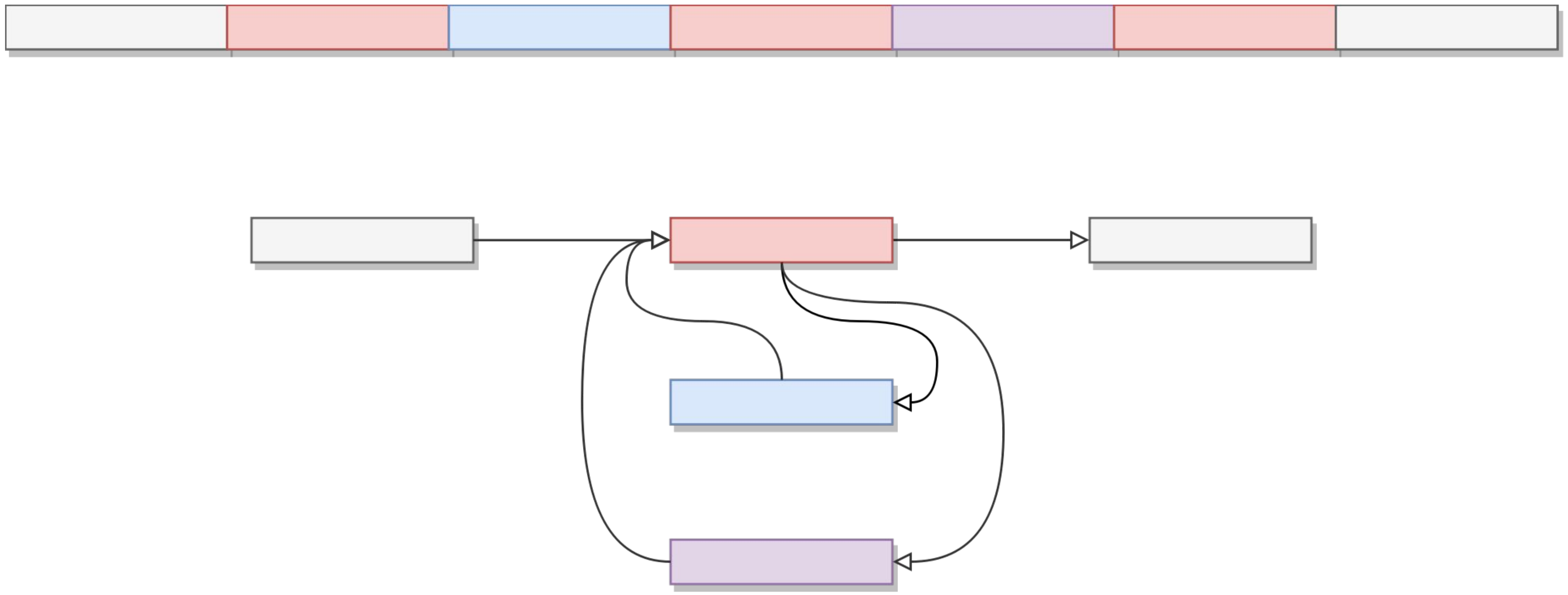


dBG at K=51 from a *Salmonella* genome, generated with Bandage
Source: https://github.com/rrwick/Bandage/wiki/Effect-of-kmer-size

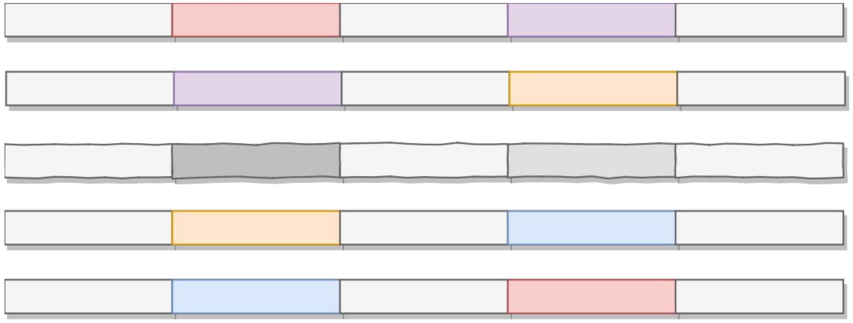


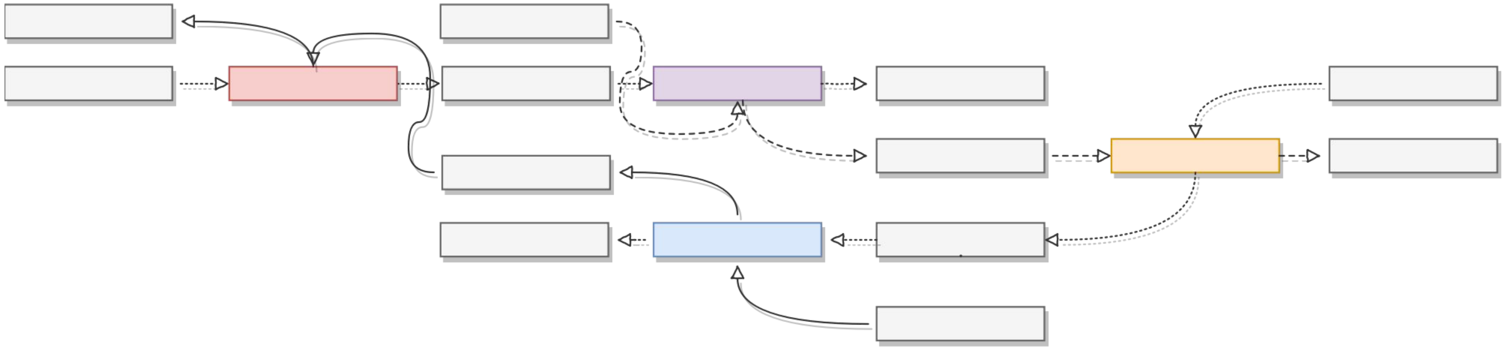


Where does this graph complexity come from, other than errors?



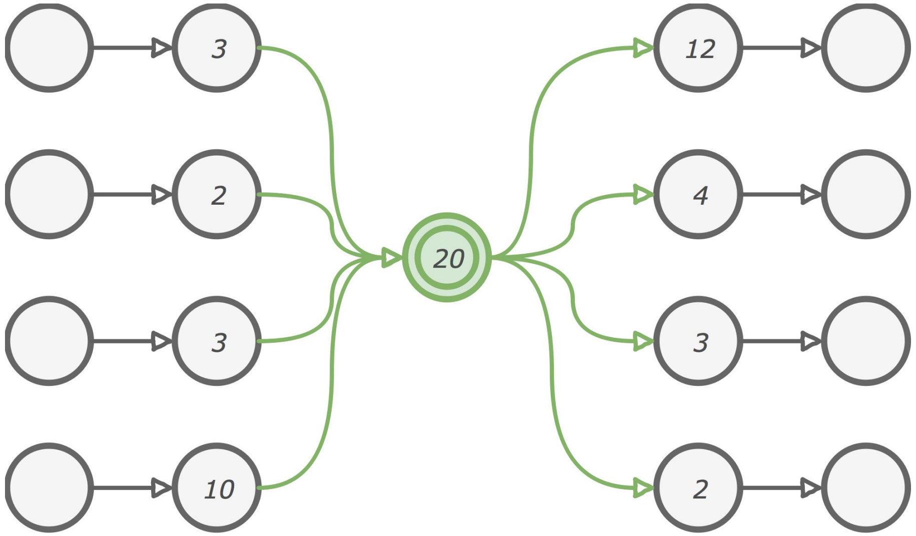
Genomes have lots of repetetive sequence: sometimes tandem repeats...





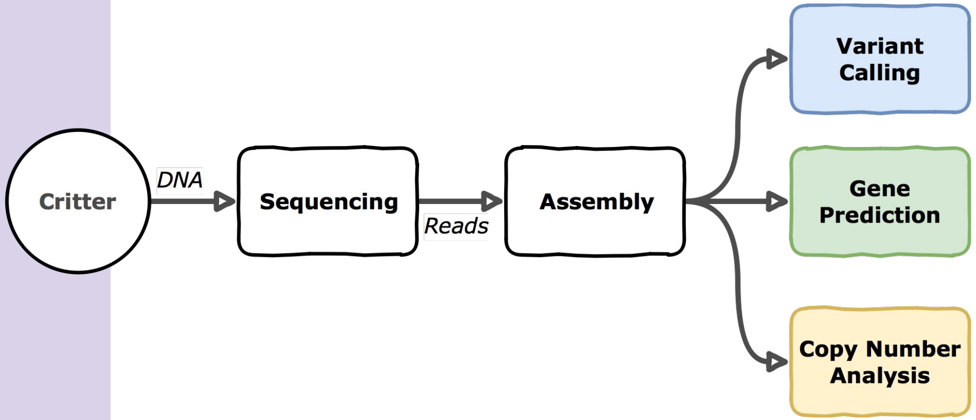
…sometimes more complex structures, like this one that sometimes show up when sequencing messenger RNA

- Often times, it isn’t possible to output a faithful representation of the original DNA
- Our reads might be too short for the repeat content, we might not completely sample
- And increasingly, the field is moving toward the idea that we shouldn’t try.

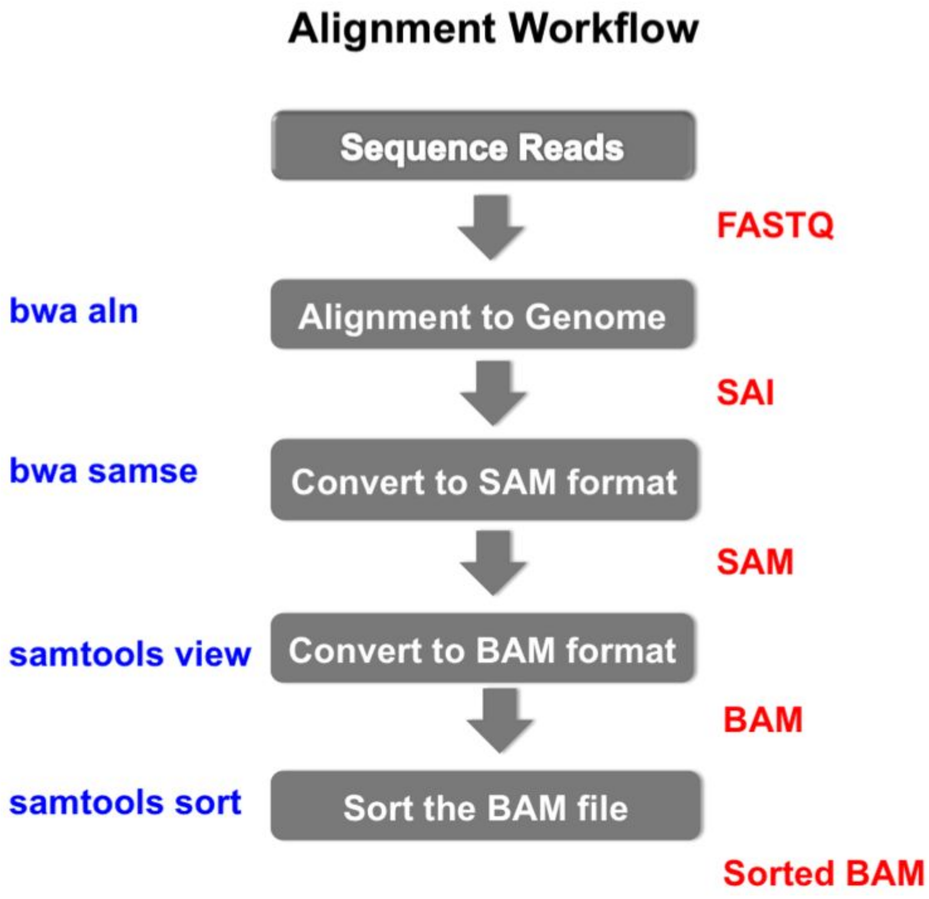


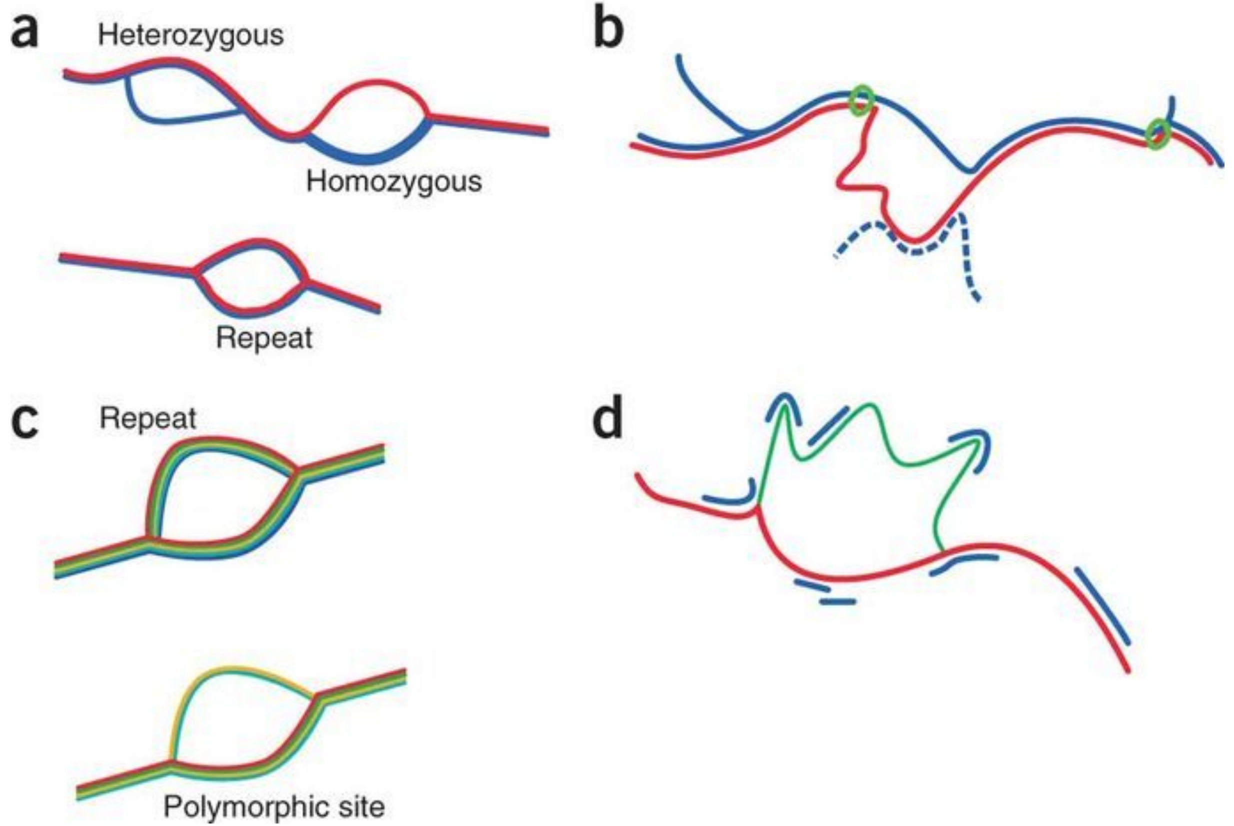
What do we do with the data?

- Traditionally, analysis methods have relied on assembly outputting contigs, then mapping reads back to those contigs or aligning those contigs to other databases
- Recent approaches are moving toward doing these things directly on the graph

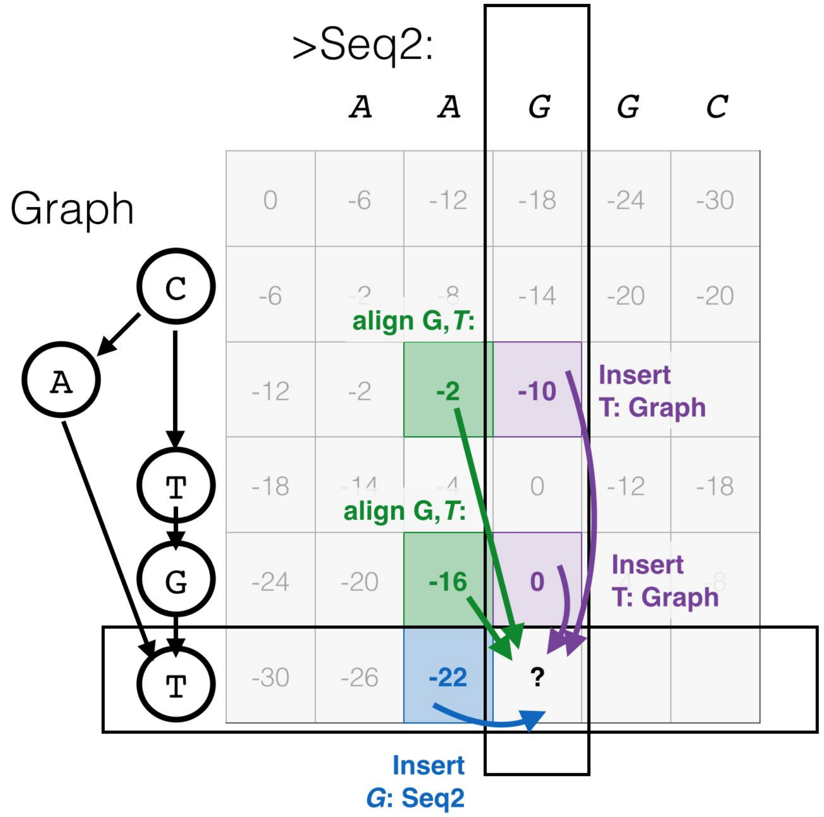


- After that assembly step, things get messy
- Commonly used alignment protocols (done before variant calling can begin) passes over the reads over four times!





Iqbal, Zamin, et al. "De novo assembly and genotyping of variants using colored de Bruijn graphs." Nature genetics 44.2 (2012): 226.



		A	A	G	G	C
	0	-6	-12	-18	-24	-30
C	-6	-2	-8	-14	-20	-20
A	-12	-2	-2	-10		
T	-18	-14	-4	0	-12	-18
G	-24	-20	-16	0	-4	-8
T	-30	-26	-22	?		

http://simpsonlab.github.io/2015/05/01/understanding-poa/

Conclusion:

Graphs are great for genomics. We already use them a lot, but let’s use them even more.

Acknowledgements

My PI: *Titus Brown*
The DIB Lab: *Lisa Johnson, Harriet Alexander, Luiz Irber, Daniel Standage, Phil Brooks, Tessa Pierce, Karen Word, Taylor Reiter, Tamer Mansour*

Links

Lab Website: http://ivory.idyll.org/lab/

My Website: http://www.camillescott.org/

Github: https://github.com/camillescott

Lab Github: https://github.com/dib-lab/

Bandage: https://github.com/rrwick/Bandage