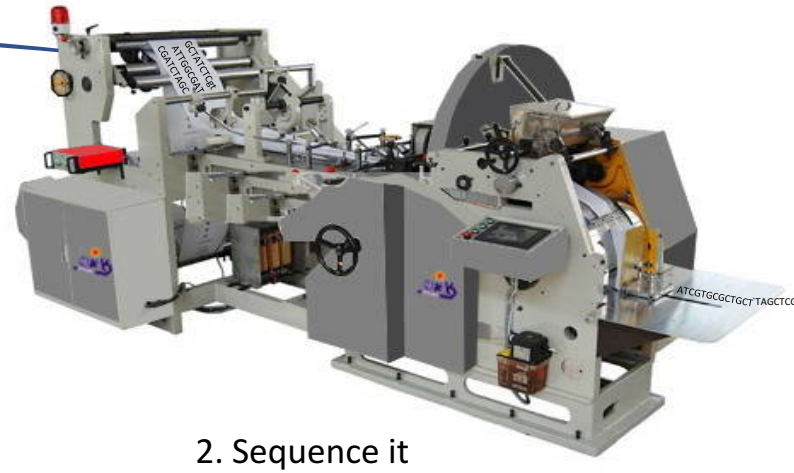
Genome Assembly

Tim Kahlke
tim.kahlke@uts.edu.au
https://github.com/timkahlke
Twitter: @AdvancedTwigTec

How to sequence a genome …

1. DNA



2. Sequence it

ATCGTGCGCTGCT TAGCTCGACTGCATATATCGCTCTCTACGATCTCGATCGTACTCTCGCGCCTACATAACAGCTATCAGTATCGATACACTGATCAGCTGATCAGTCATCGTGCGCTGCTACGATCAGTCATATCATCGATCGCGCGCGGATCTAC…

3. Voila, a genome!

What is a genome assembly?

1. DNA

2. Fragmentation (Shot Gun)

3. Sequencing

ATCTCGATCG CTCGATCG
CTCGATCT CGCGTACGTC ATCTCGATCG
ATCTCGATCG CGCGTACGTC
ATCTGCTA CTCGATCTGCTA CTCGATCTGCTA
CGCGTACGTC CTCGATCTGCTA

4. Assembly

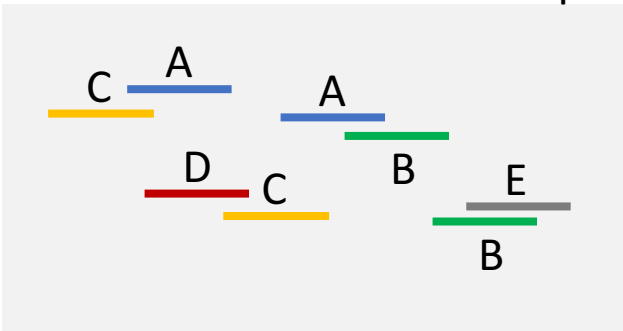
…TAGCTCGACTGCATATATCGCTCTCTACGATCTCGATCGTACTCTCGCGCCTACATAACAGCTATCAGTATCGATACACTGATCAGCTGATCAGTCATCGTGCGCTGCTACGATCAGTCATATCATCGATCGCGCGGATCTAC…

ATGAACGATCGCTACGATTCGACACGCTCGCATC
ACACGCTCGCATCTCGGCTCATCGTCGTACGTACG
GTACGTACGATGCTACGTCTGCTACGTC

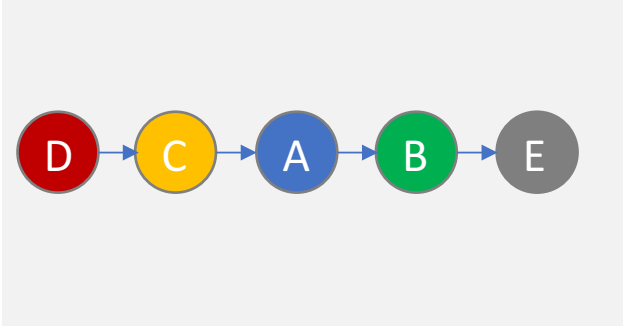
…ATGAACGATCGCTACGATTCGACACGCTCGCATCTCGGCTCATCGTCGTACGTACGATGCTACGTCTGCTACGTC…

Naïve assembly procedure

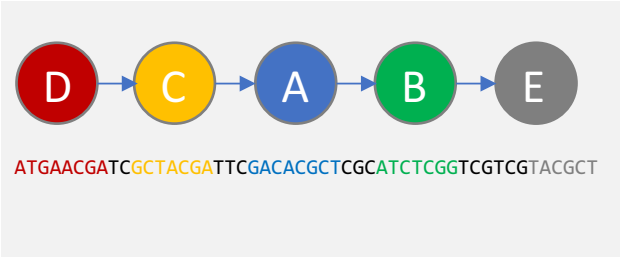
1. Screen all reads for overlaps



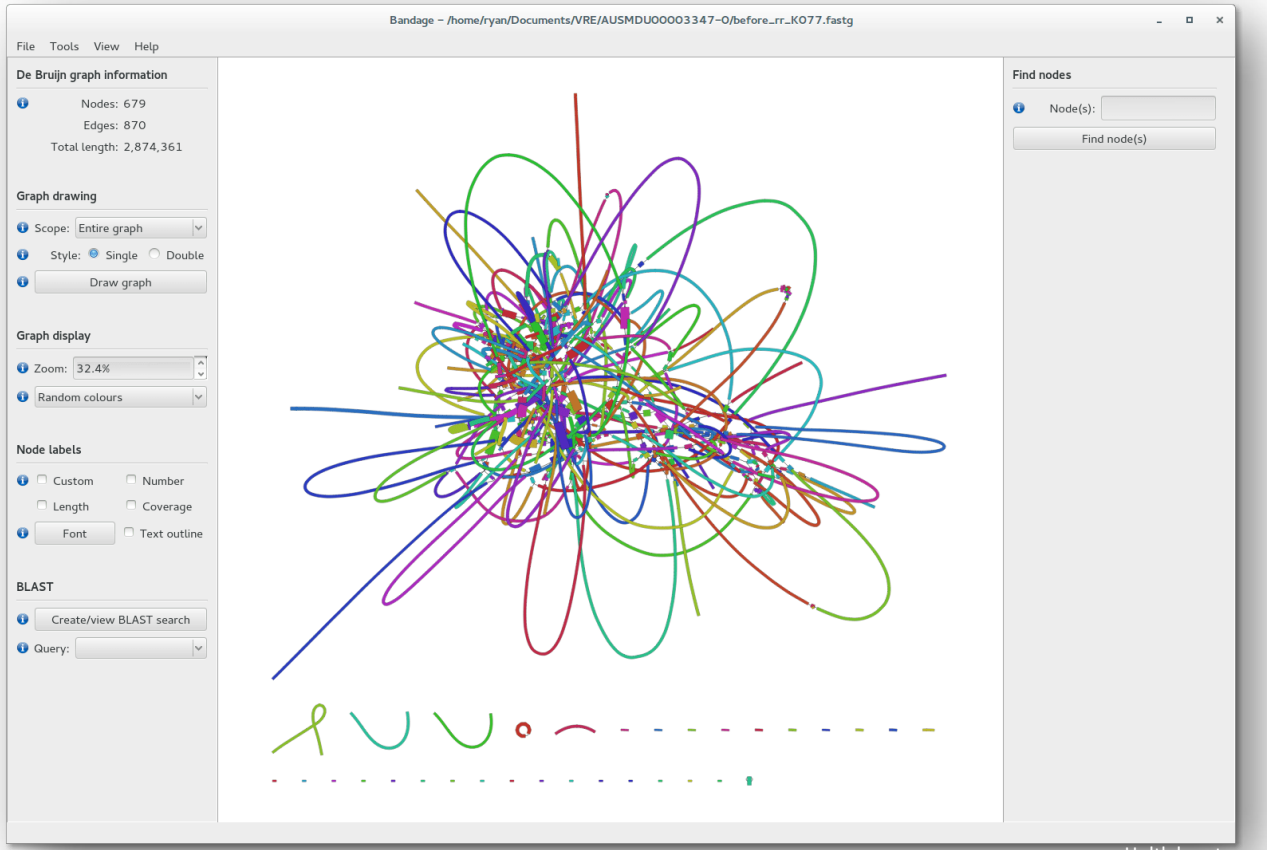
2. Build an assembly graph



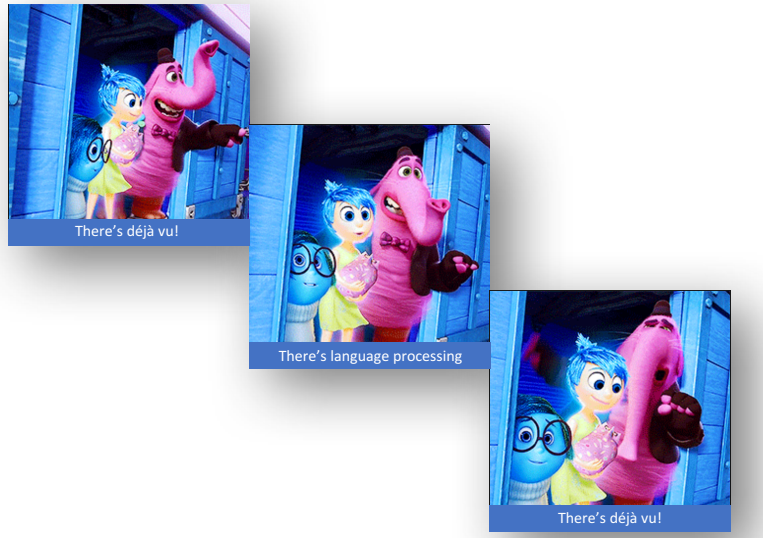
3. Sequence = assembly graph



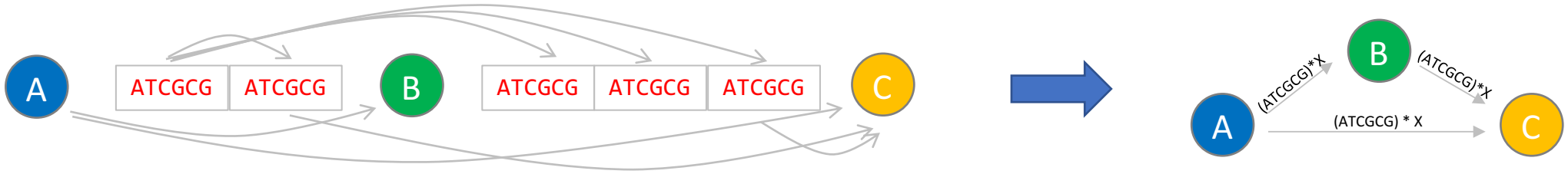
Assembly graph



Repeat sequences




```
ATGAACGATCATCGCGATCGCGACTACGATTCGACACGCTCATCGCGATCGCGATCGCGTACGTACGATGCT
ATGAACGATCAT                CGATTCGAC        ATCGCGATCGCG       GTACGTACGATGCT
          ATCGCGA                                ATCGCGA
                  ATCGCGA                                ATCGCGA
                          GACTACGA                      ATCGCGATC
                                                 CATCGCGA
       CATCGCGA
    ATCATCGC                      GACACGCTCAT          GCGATCGCGTACG
```



Repeat sequences

- Repeats that are longer than the (average) read length can **never** be assembled properly
- Order of flanking regions can not be determined
- Lead to *bubbles* in the assembly graph

Commmon repeats

- Transposable elements
- Satellite sequences, e.g. telomeres/centromeric regions

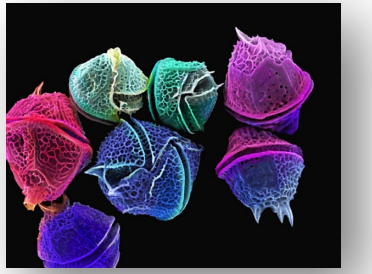
Additional complications

Polyploidy

Multiple sets of chromosomes

Ultimaohora.com

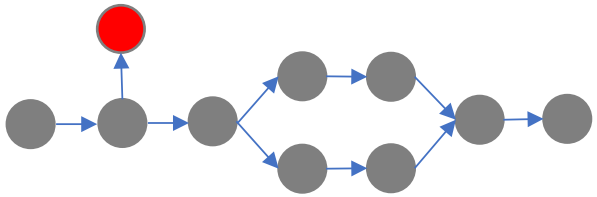
Gene duplication



blueoceanemporium.blogspot.com

Sequencing errors

- Lead to *dead ends* in the assembly graph



Assembly strategies

Overlap – layout – Consensus (OLC)

- OLC assemblers are one of two main types of assemblers
- First developed for Sanger sequencing
- Implemented in Celera assembler (Myer *et al.* 1995)

Working in 3 phases

1. Overlap: Find all overlaps of reads
2. Layout: Create an approximate layout graph
3. Consensus: Create a consensus sequence

OLC – Overlap phase

- Compare all reads against each other
- Identify overlaps of reads of at least *k* length and at most *e* errors
- BLAST, Smith-Waterman …

Very (!) computationally expensive!

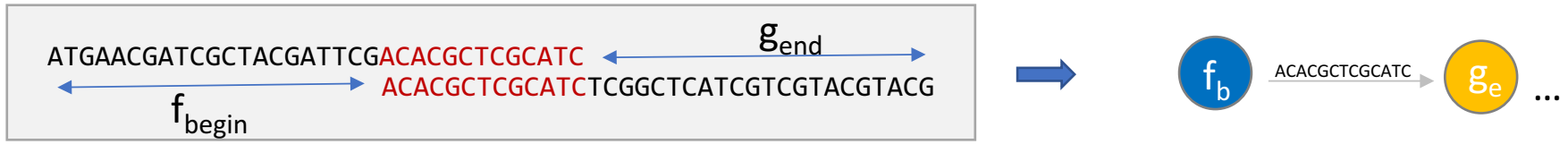
Close to $O(N^2)$, i.e., complexity increases exponentially with linear increase of input data

OLC - Layout phase

Construct a graph from all overlaps.

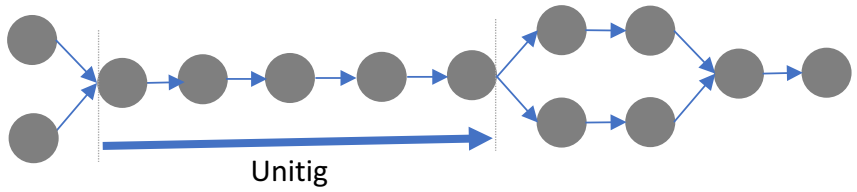
For two read *a* and *b*

- Non-overlapping parts are represented by nodes
- Overlaps are represented by edges



OLC - Unitigs

A unitig is "*a maximal interval subgraph of the graph of all fragment overlaps for which there is no conflicting overlaps to an interior vertex*" (Myers 2000).



- Are (almost) always correct, i.e., represent a true existing part of the original genome

OLC – Remove redundancies

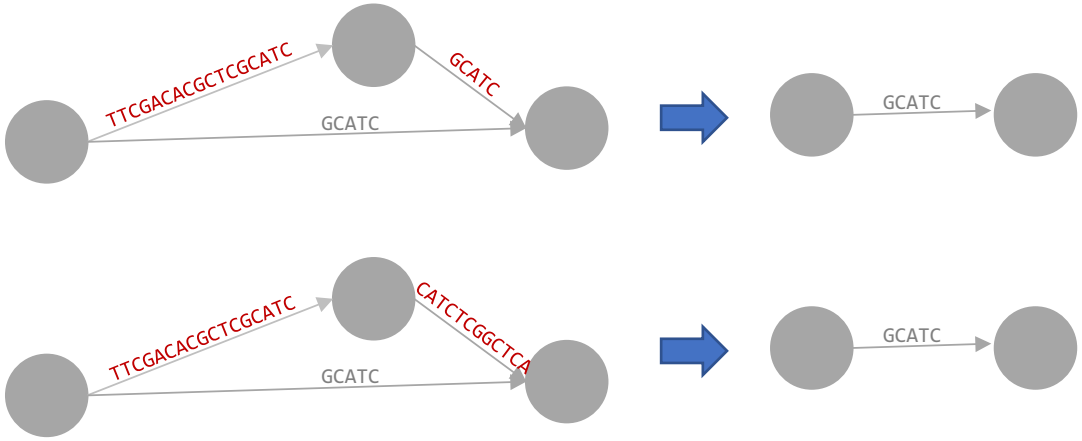
Simplify the overlap graph

1. Remove *contained* reads

```
ATGAACGATCGCTACGATTCGACACGCTCGCATC
                    TTCGACACGCTCGCATC
                                  GCATCTCGGCTCATCGTCGTACGTACGTGCTCGAT
```

2. Remove *transient* reads

```
ATGAACGATCGCTACGATTCGACACGCTCGCATC
                    TTCGACACGCTCGCATCTCGGCTCA
                                  GCATCTCGGCTCATCGTCGTACGTACGTGCTCGAT
```

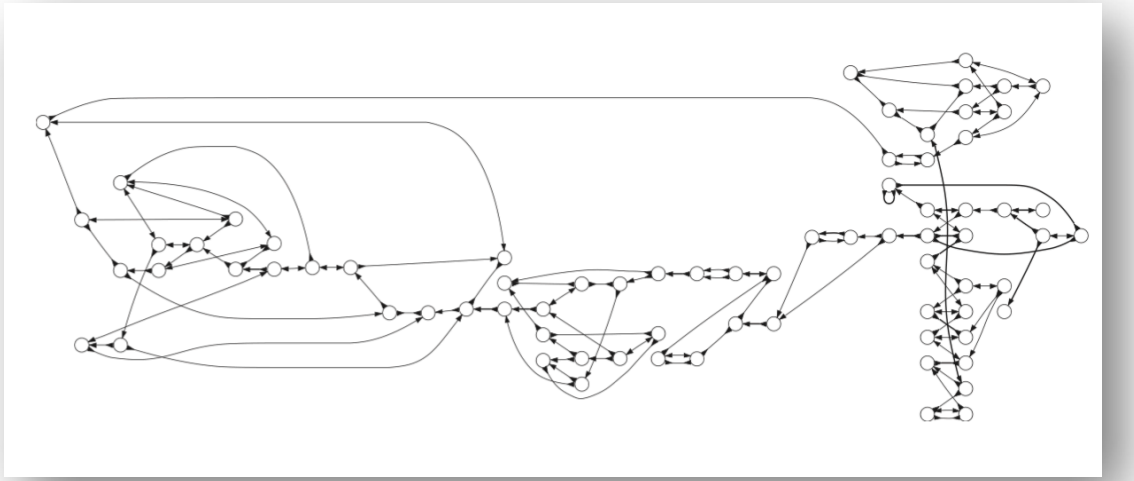


Shortest Common Substring problem

Find a *path* in the assembly graph that represents the **s**hortest **c**ommon **s**ubstring

1. visits each vertex (node) exactly ones (*Hamiltonian* path)
2. represents the **s**hortest **c**ommon **s**substring

NP-hard problem!

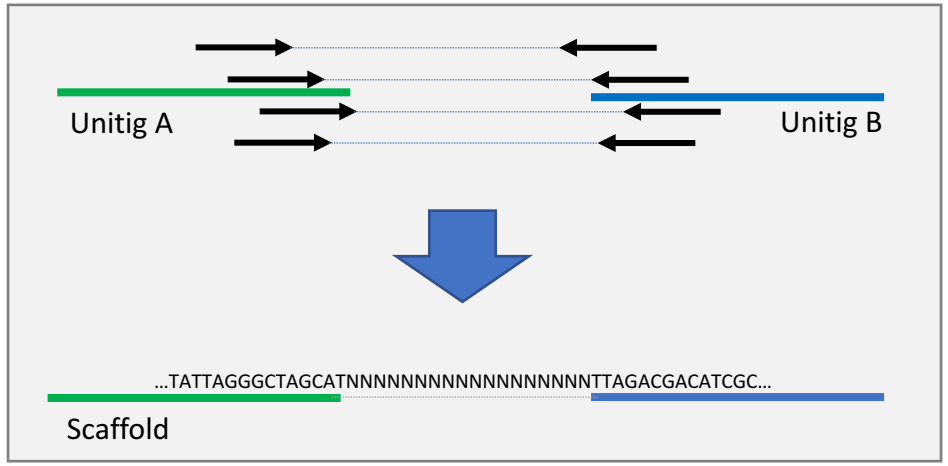


Celera assembly graph of *C. jejuni*

Myers 2005

OLC – Scaffolding

- Further connection of unitigs
- Use mate-pair information to connect unitigs
- Create ordered sequences separated by gaps of approximately known size



- Where possible fill gaps with singletons & reads
- Create contiguous ordered sequences, *contigs*

OLC - Consensus

Determine the sequence of all unitigs and contigs

- Majority rule of all reads
- Incorporate quality scores
- Remove

```
ATGAACGATCGCTACGATTCGACACGCTCGCATCTCG
                TTCGACA-GCTCGCATCT-GGCTCATATT
                      ACGCTCGCATCTCGGCTCAT--TCGTACGTACGTGCTCGAT
                                           ATTCGTACGTACGTGCTCGATGCTACG
ATGAACGATCGCTACGATTCGACACGCTCGCATCTCGGCTCATATTCGTACGTACGTGCTCGATGCTACG
```

Problems with OLC-assemblers

- Overlap costly

  Celera comparisons for 27,000,000 human reads =~1,458,000,000,000 comparisons

- Large overlap graph

  High memory requirements for large data sets

- SCS problem can’t be solved

De Bruijn graph algorithms

1. Create a *hash* (list) of all possible substrings of length *k* (*kmers*) of all reads

TACGCG → TAC
TACGCG → ACG
TACGCG → CGC
TACGCG → GCG

CGCGTC → CGC
CGCGTC → GCG
CGCGTC → CGT
CGCGTC → GTC

2. Count occurances of *kmers*

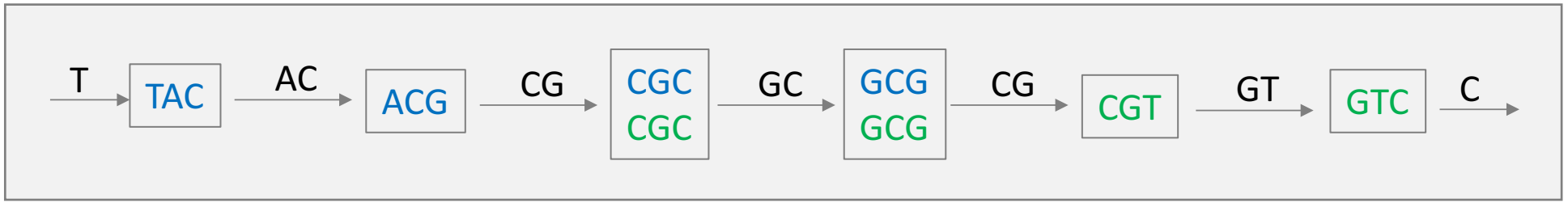
TACGCG → TAC
TACGCG → ACG
TACGCG → CGC
TACGCG → GCG

CGCGTC → CGC
CGCGTC → GCG
CGCGTC → CGT
CGCGTC → GTC

De Bruijn graph

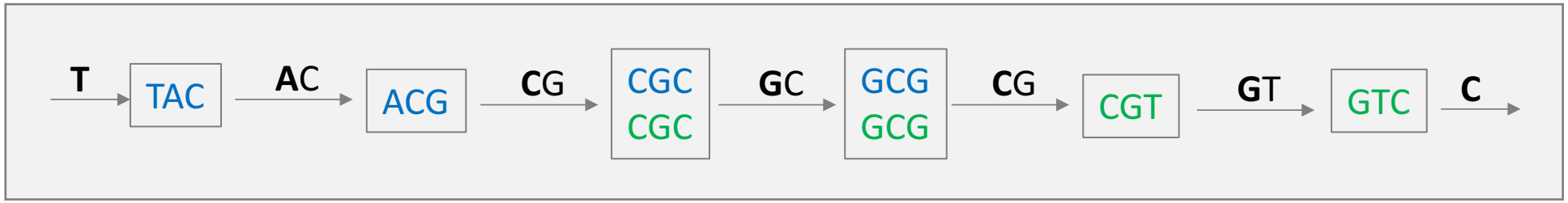
Build a graph

- Each *kmer* represents a node
- Connect nodes with matches of *k-1* words
- Label edges with *k-1* overlap



De Bruijn graph

- Traverse through the graph
- Concatenate the first letter of each edge

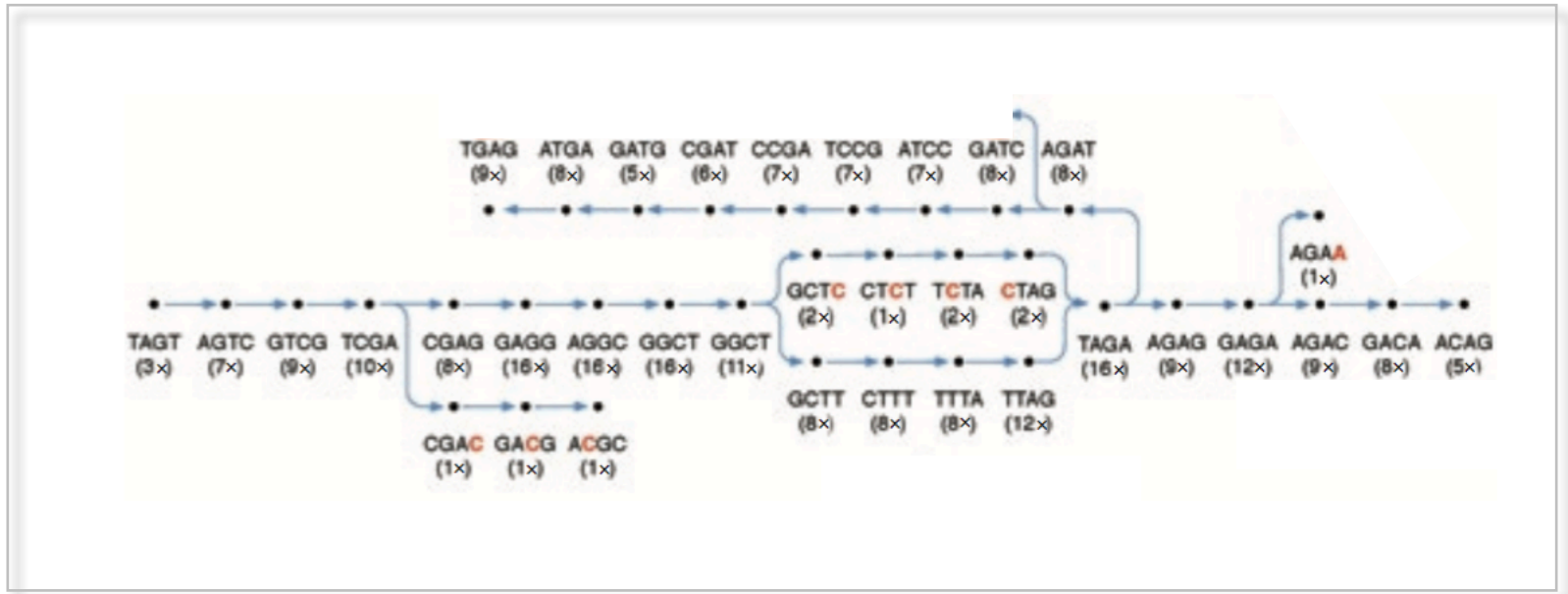


TACGCGC

TACGCG

CGCGTC

De Bruijn graph

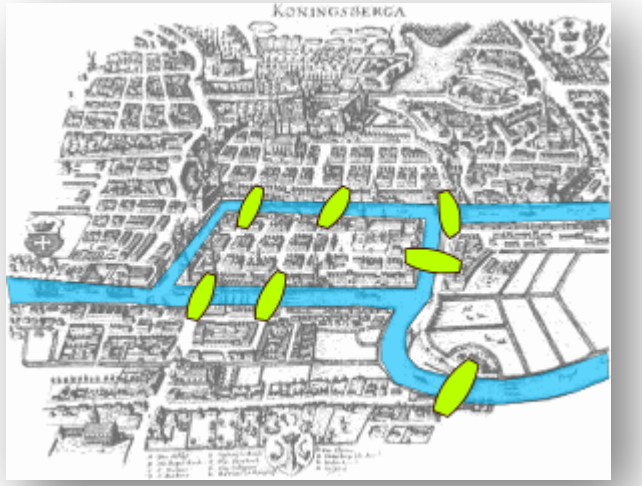


Ficek et al 2009

De Bruijn

Advantage

- No expensive overlap step
- Eulerian path: all *edges* have to be visited once
- The genome sequence can be created by walking through the De Bruijn graph



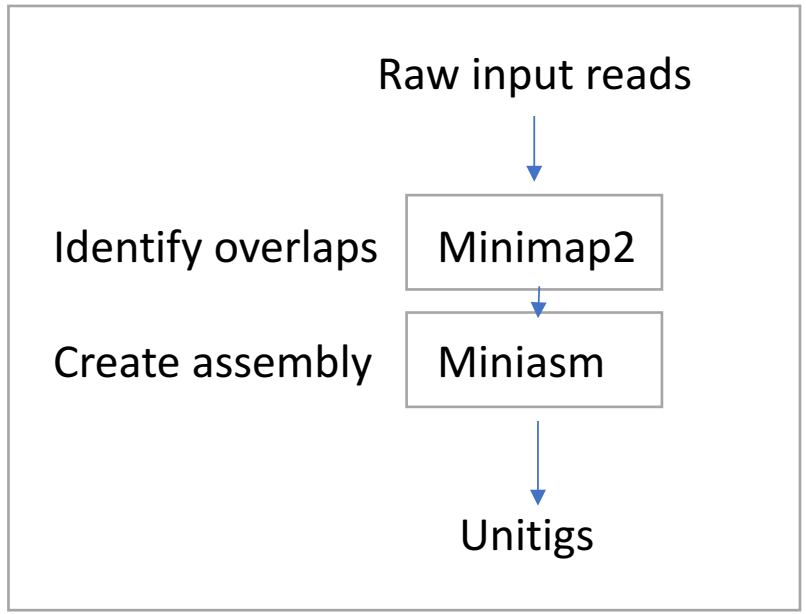
Disadvantages

- Long read information lost
- Can result in highly fragmented assemblies & artifacts

Minimap / miniasm pipeline

Minimap / Miniasm pipeline

- Pipeline for mapping and assembly of long reads
- Creates unitigs
- Very fast
- Similar to OLC but without consensus
- No consensus performed => error rate of unitigs equal to input raw reads



Minimap

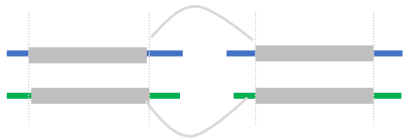
Fast mapping of long-read sequences to identify overlaps

1. Seed

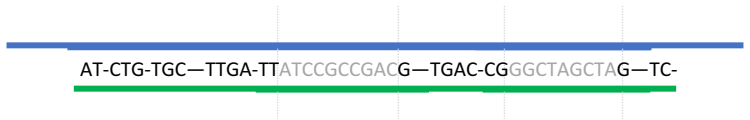
   Identify shared *kmers* between two sequences

2. Chain

   Chain anchors (seed alignments) given a maximum mismatch threshold



3. Align sequences between anchors



Miniasm

1. Trimming

Identify high-coverage overlaps (default 4) of >=2000bp and trim ends

2. Build assembly graph

Similar to OLC

3. Clean assembly graph

- Remove transitive reads
- Trim unitigs at tips with coverage <4
- Remove *bubbles* of paths <=50kb

4. Determine unitigs

Traverse through assembly graph to determine unitig sequences

Alignment quality

What is a good alignment?

So far there is no good standard for assembly QC assessment

- If possibe compare to reference genome / closely related organism
  - Percent identity
  - Gaps / breaks
- Minimum/maximum length of contigs/unitigs
- # of contigs/unitigs
- Number of reads that map back to assembly: Assumption: the higher the number of reads the assembly includes the better.

Assembly metrics – N50

- *N* – metrics: measure of contiguity of a set of sequences
- *N50:* is related to the median and mean length of a set of sequences
- N50 = *length of the shortest read in the group of longest sequences that together represent (at least) 50% of the nucleotides in the set of sequences.*

Example

Six reads of length 10, 11, 12, 13, 14, 15, 16.

Total length: 81 (50% = 40.5)

Sum read length shortest to longest until sum >=40.5

16 + 15 + 14 = 45

N50 = 14

Assembly metrics – N50

Problems

Has to be seen in context of read numbers

- Only assemblies with similar read numbers are comparable

- Many short contigs don't affect N50

  Set 1: 10, 11, 12, 13, 14, 15, 16

  Set 2: 2, 2, 3, 4, 12, 13, 14, 15, 16

  N50 = 14

Hybrid assemblies

Hybrid assemblies

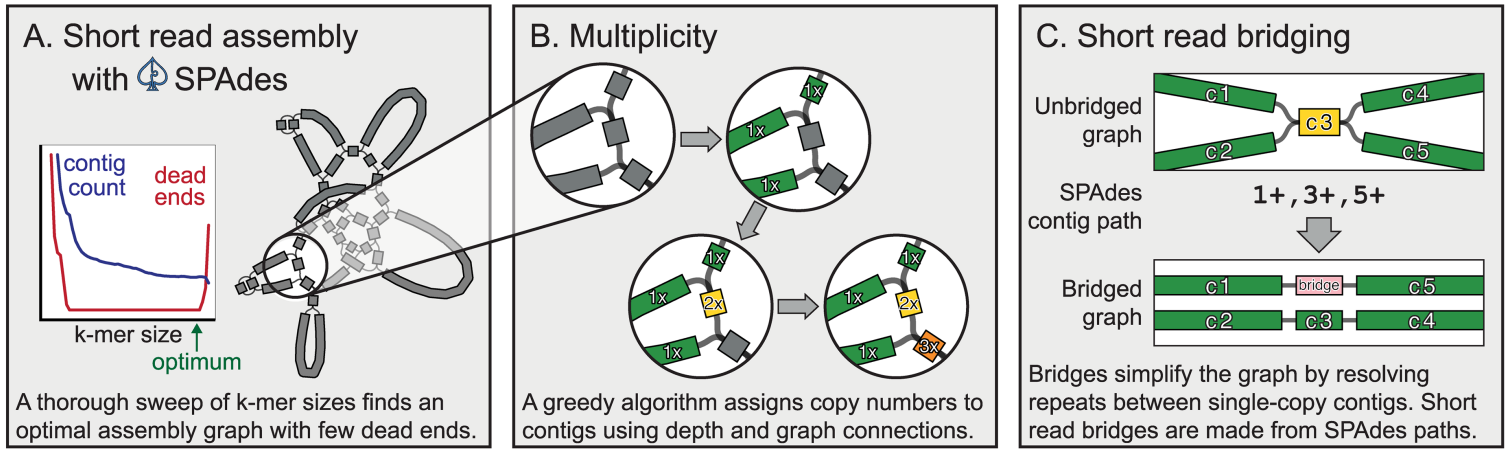
- Genome assembly using long reads (PacBio & Nanopore) as well as short reads (Illumina)

Different approaches

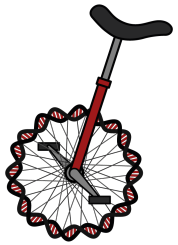
1. Long-reads first, e.g. Cerulean
    - Use long reads for overlap assembly
    - Use short reads for error correction
2. Short-reads first, e.g. Unicycler
    - Create short-reads assembly
    - Use long-reads to bridge *bubbles*

Unicycler

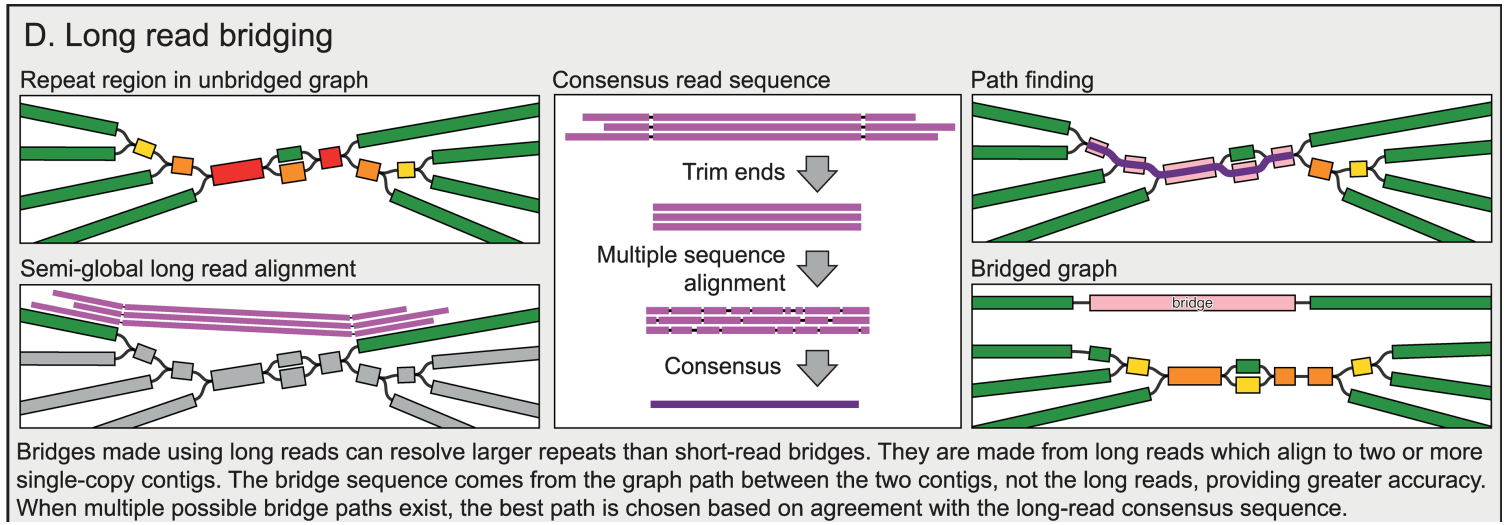
Short-read assembly phase



Wick *et al.* 2017

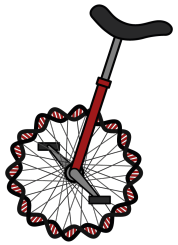


Unicycler

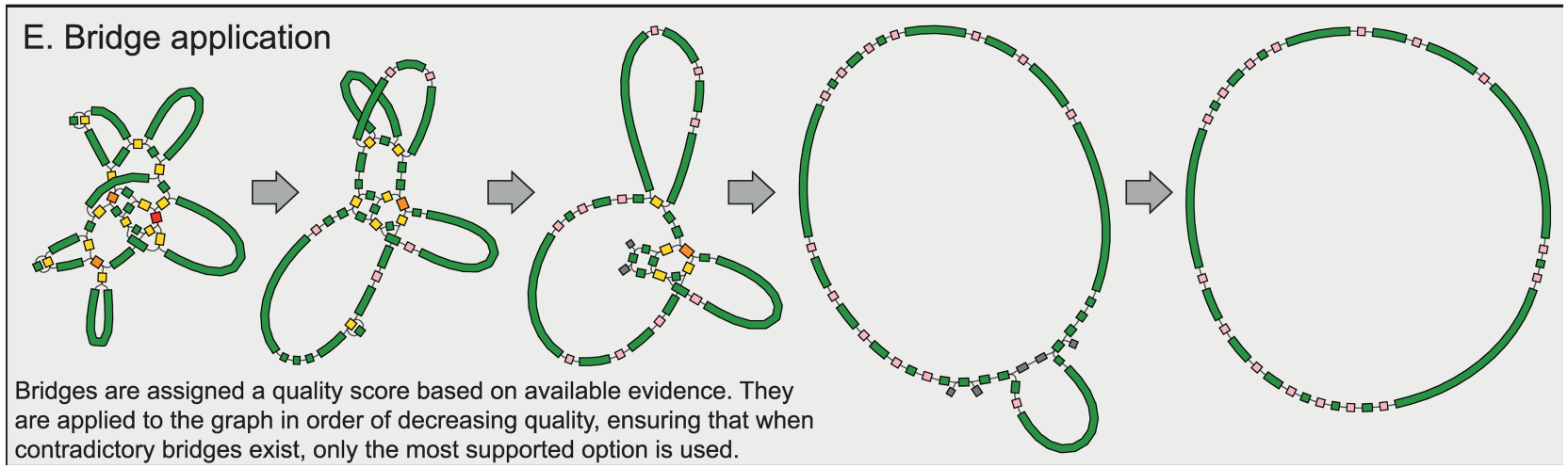


Bridges made using long reads can resolve larger repeats than short-read bridges. They are made from long reads which align to two or more single-copy contigs. The bridge sequence comes from the graph path between the two contigs, not the long reads, providing greater accuracy. When multiple possible bridge paths exist, the best path is chosen based on agreement with the long-read consensus sequence.

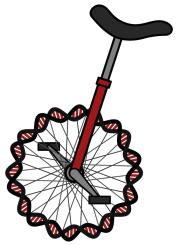
Wick *et al.* 2017



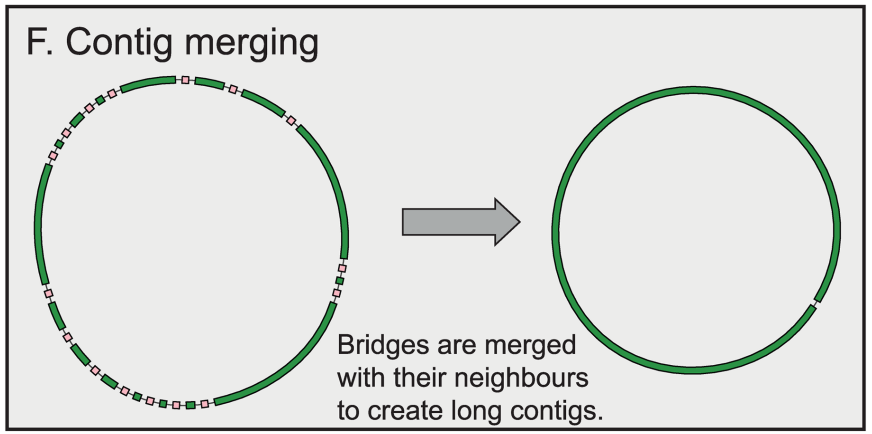
Unicycler

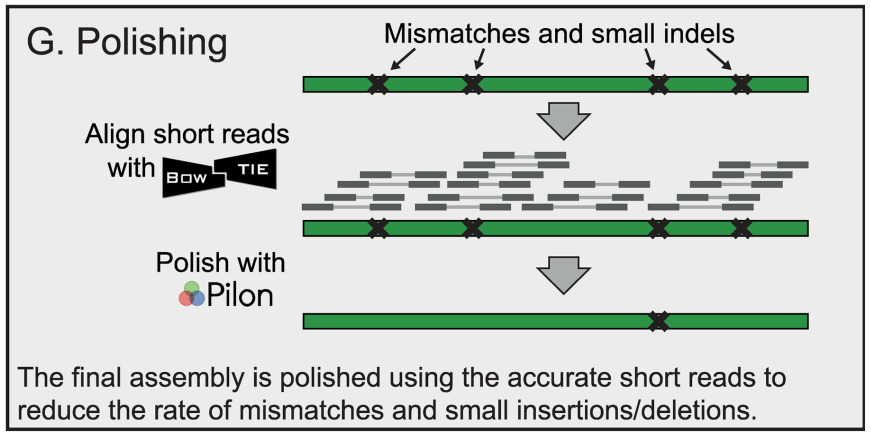


Wick *et al.* 2017



Unicycler





Wick *et al.* 2017

Questions ?