SevenBridges

DNA Sequence Assembly

Mladen Lazarevic
Milan Kovacevic
milan.kovacevic@sbgenomics.com

Agenda

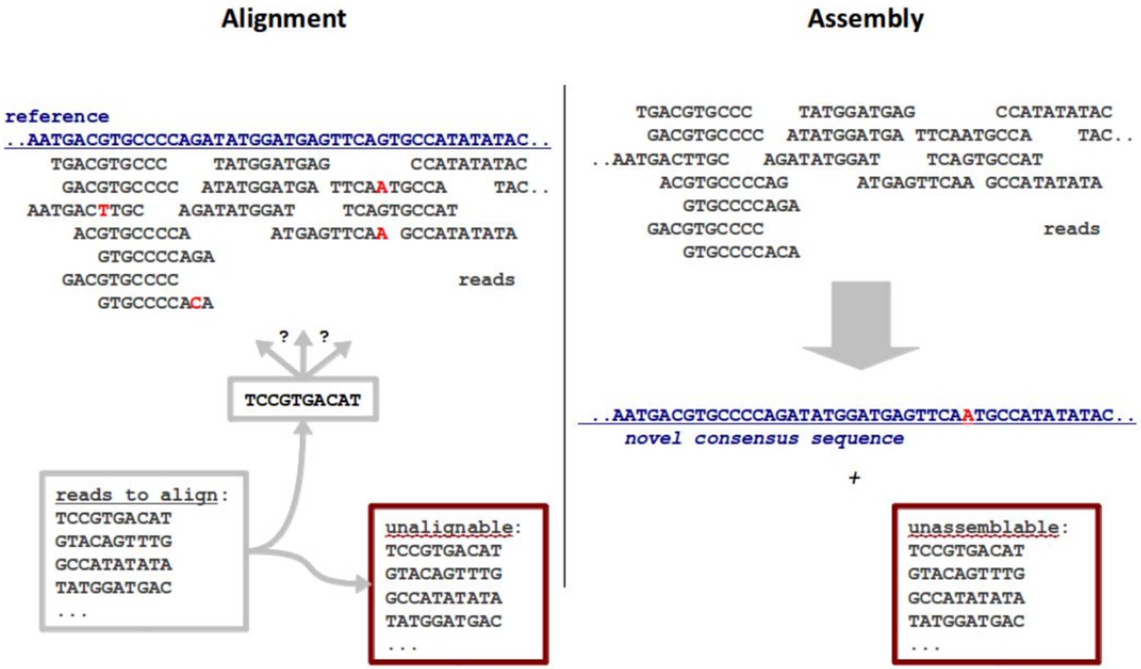
- Assembly – basic overview
- DeBrujinGraph assembly
- String Graphs in assembly
- Assembly metrics
- Python examples

Recap

- What is PCR?
- What is a flow cell?
- What are paired-end reads?
- What to do with the sequencing reads?
- Human Genome Project?
- How many chromosomes do people have?
- Define the Central Dogma, genes and exome.
- What do we store in FASTA, FAI and FASTQ file formats?

De novo assembly

- **Assembly:** Set of sequences which best approximate the original sequenced material
- Why?
- Reference
- Novel features



Alignment

```
reference
..AATGACGTGCCCCAGATATGGATGAGTTCAGTGCCATATATAC..
    TGACGTGCCC    TATGGATGAG        CCATATATAC
     GACGTGCCCC  ATATGGATGA TTCAATGCCA     TAC..
  AATGACTTGC    AGATATGGAT     TCAGTGCCAT
      ACGTGCCCCA        ATGAGTTCAA GCCATATATA
        GTGCCCCAGA
     GACGTGCCCC                           reads
        GTGCCCCACA
```

? ?

TCCGTGACAT

reads to align:
TCCGTGACAT
GTACAGTTTG
GCCATATATA
TATGGATGAC
...

unalignable:
TCCGTGACAT
GTACAGTTTG
GCCATATATA
TATGGATGAC
...

Assembly

```
    TGACGTGCCC    TATGGATGAG        CCATATATAC
     GACGTGCCCC  ATATGGATGA TTCAATGCCA     TAC..
..AATGACTTGC    AGATATGGAT     TCAGTGCCAT
      ACGTGCCCCAG       ATGAGTTCAA GCCATATATA
        GTGCCCCAGA
     GACGTGCCCC                           reads
        GTGCCCCACA
```

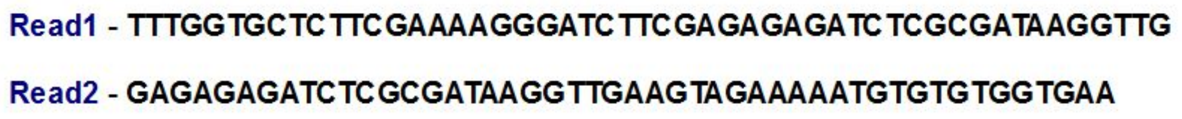
..AATGACGTGCCCCAGATATGGATGAGTTCAATGCCATATATAC..
novel consensus sequence

+

unassemblable:
TCCGTGACAT
GTACAGTTTG
GCCATATATA
TATGGATGAC
...

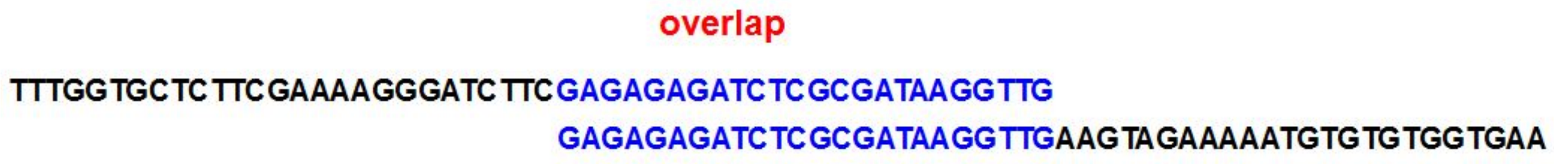
De novo assembly

- Best case scenario: check every pair of (long) reads for overlaps
- Computationally expensive: **n** reads, **~n**2** operations
- Our task: how to get the **best assemblies** for the **smallest expense** – in terms of sequencing and computational expenses



Read1 - TTTGGTGCTCTTCGAAAAGGGATCTTCGAGAGAGATCTCGCGATAAGGTTG

Read2 - GAGAGAGATCTCGCGATAAGGTTGAAGTAGAAAAATGTGTGTGGTGAA

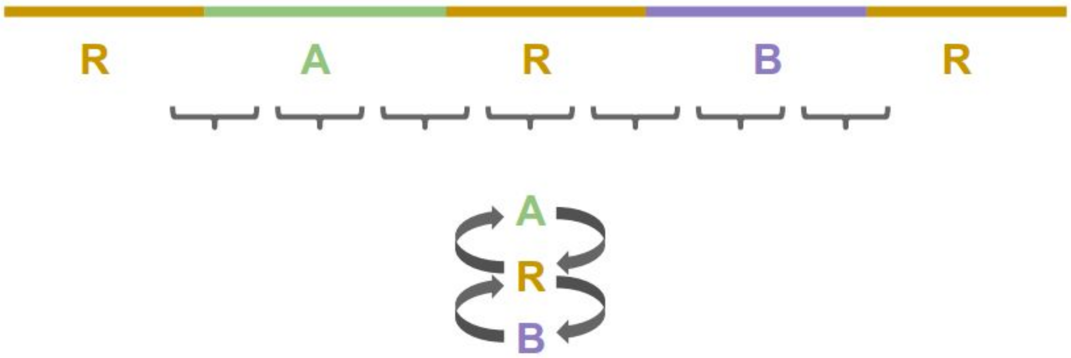


overlap

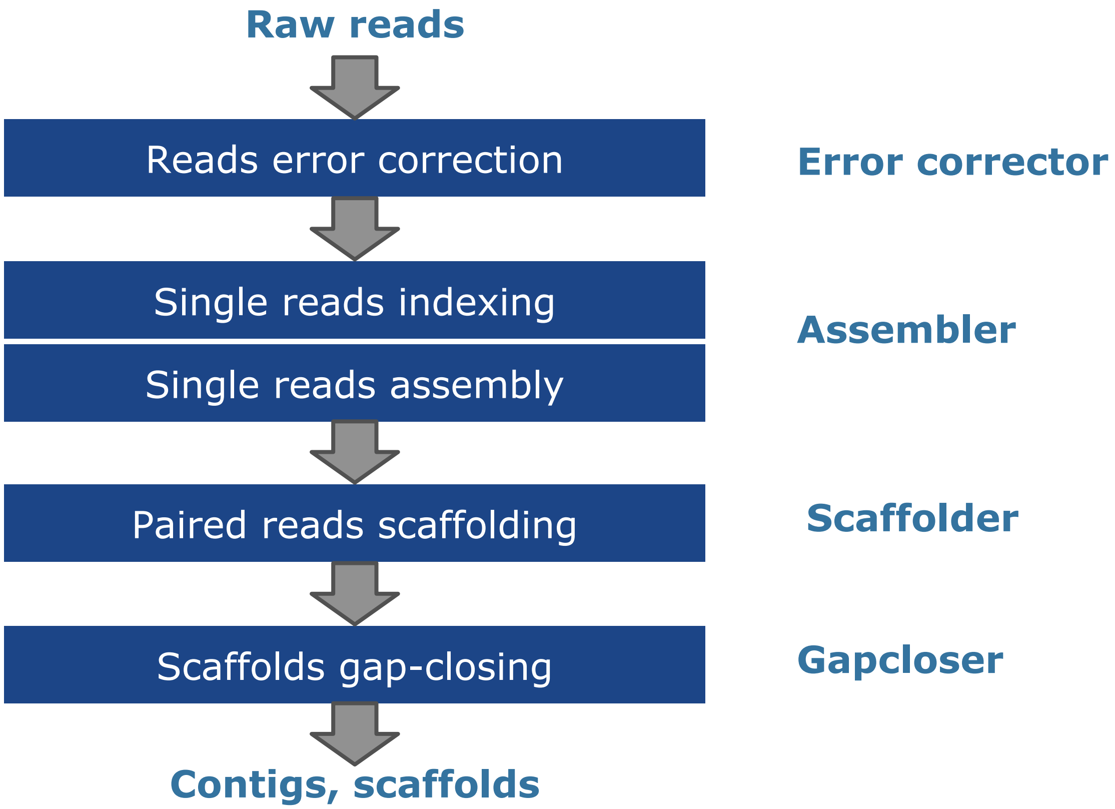
```
TTTGGTGCTCTTCGAAAAGGGATCTTCGAGAGAGATCTCGCGATAAGGTTG
                           GAGAGAGATCTCGCGATAAGGTTGAAGTAGAAAAATGTGTGTGGTGAA
```

De novo assembly

- “[...] **repeats** are the single biggest impediment to all assembly algorithms and sequencing technologies” - Koren 2012 Nature Biotechnology
- Short read libraries have a hard time resolving large repeats, even with mate-pair, or "jumping" libraries



A typical assembly pipeline



Raw reads

Reads error correction

Single reads indexing

Single reads assembly

Paired reads scaffolding

Scaffolds gap-closing

Contigs, scaffolds

Error corrector

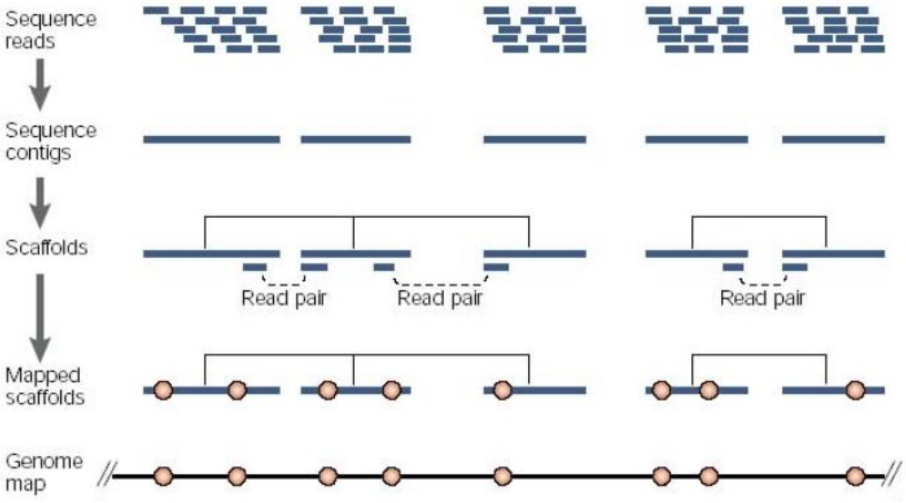
Assembler

Scaffolder

Gapcloser

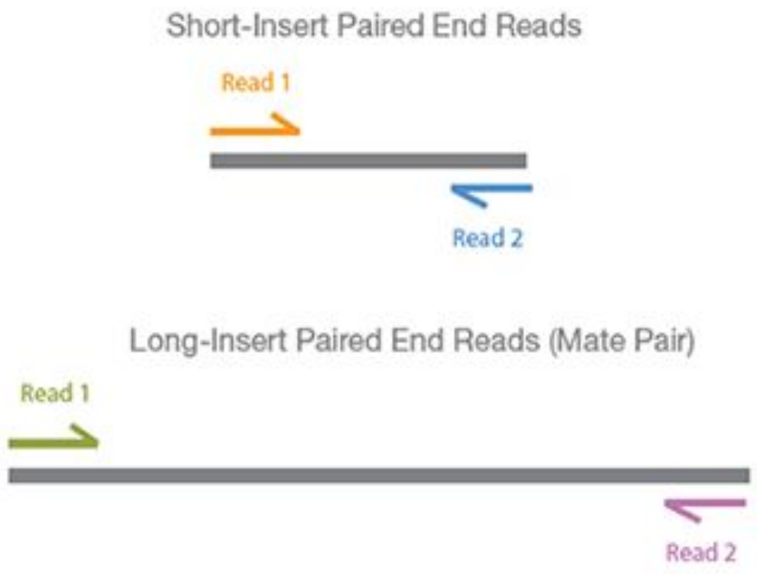
A typical assembly pipeline

de novo whole-genome shotgun assembly



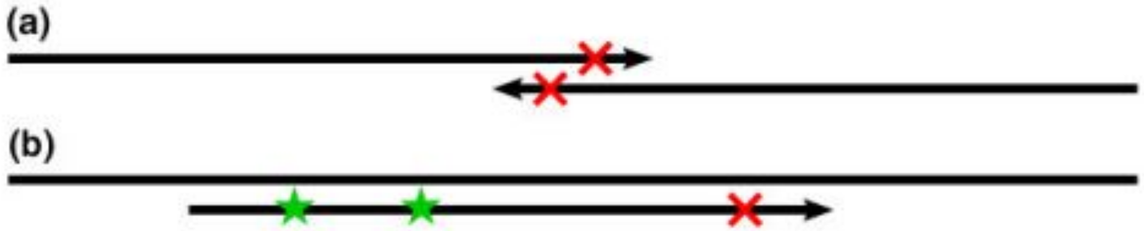
Vocabulary

- **Paired end reads:** read1, insert < 500 bp, read2
- **Mate pair reads:** read1, insert > 1 kbp, read2
- **k-mer:** any sequence of length k
- **Contig:** gap-less assembled sequence
- **Scaffold:** sequence which may contain gaps


Short-Insert Paired End Reads
Read 1
Read 2
Long-Insert Paired End Reads (Mate Pair)
Read 1
Read 2

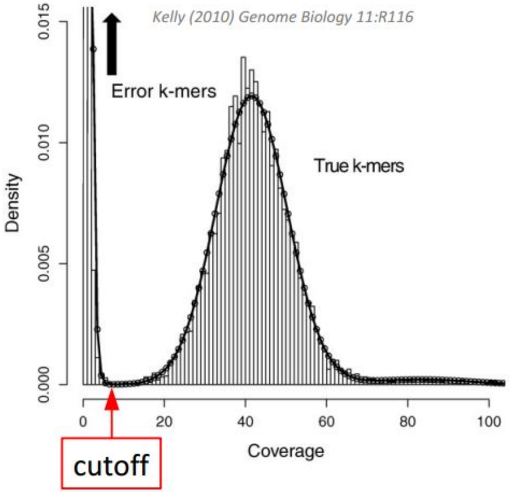
Error correction

- Assembly: errors (mostly at 3’ read ends) disrupt overlaps
- Alignment: where real variation is present, errors add to differences and disrupt alignment



Error correction

- Determining trusted and untrusted k-mers
- Untrusted -> trusted, using highest likelihood



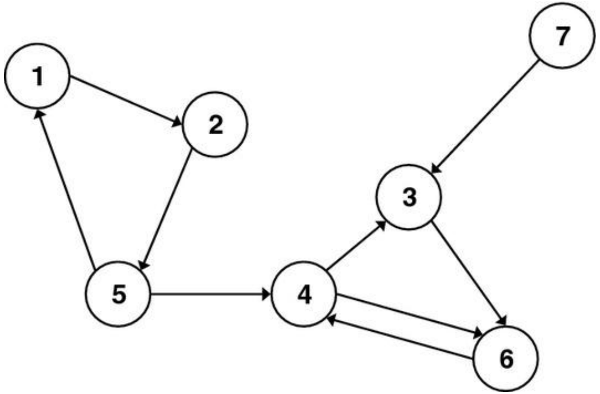
Kelly (2010) Genome Biology 11:R116

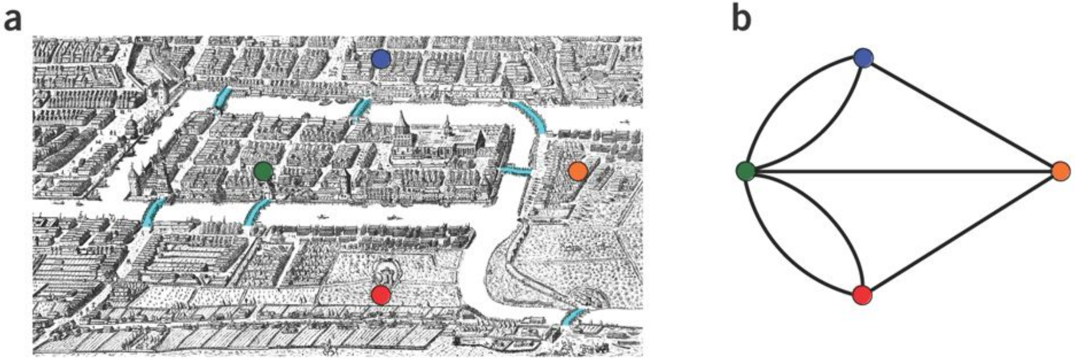
e.g.:

```
CCGGATTTACGCCGAGTGA x 43
CCGGATTTACGCTGAGTGA x 1
            ^
```

Graphs

- A **graph** is a set of nodes and a set of edges
- Edges can be **directed** or **not**





Graphs and de novo assembly

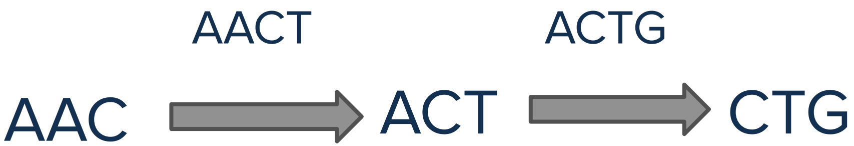
- **Overlaps** between reads – fundamental information used for de novo assembly
- **Graphs** enable representation of these overlaps
- Two different types of graphs for sequencing data are known :
  - **de Bruijn** graphs (pronounced “brain”)
  - **string** graphs

De Bruijn graphs

- For a fixed integer **k**:
  - **nodes** - all **k-1**-mers present in reads
  - **edges** - for each **k**-mer **x** present in reads there is an edge between **k-1**-mer **prefix** of **x**, and **k-1**-mer **suffix** of **x**

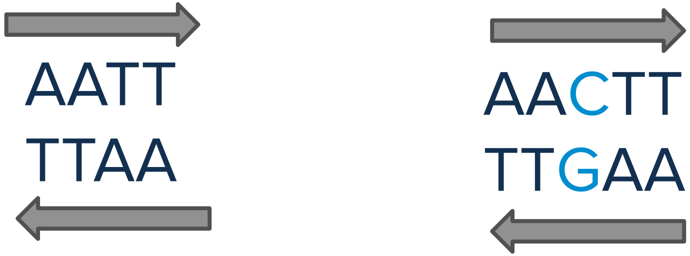
- Example for a single read and k = 4:

AACTG



De Bruijn graphs

- In assembly, we always identify a read with its reverse complement
- **k** is practically exclusively an **odd** integer as a result (otherwise that causes ambiguities in the strand-specificness of the graph)

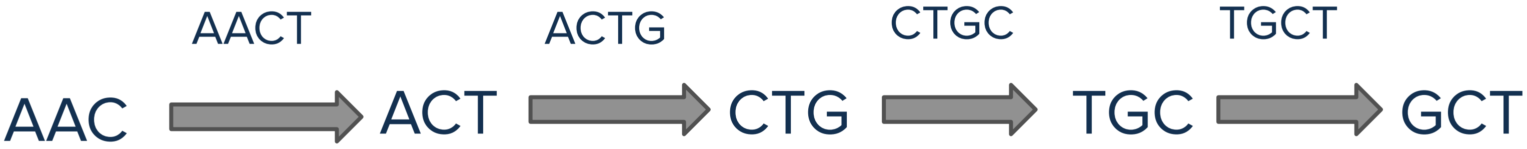

AATT
TTAA

AACTT
TTGAA

De Bruijn graphs

Example for multiple reads and k = 4:

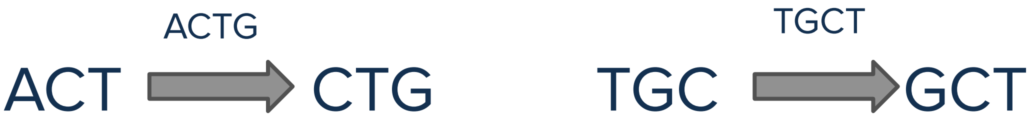
AACTG
  ACTGC
    CTGCT



De Bruijn graphs

Similar example for multiple reads and k = 4:

ACTG
TGCT

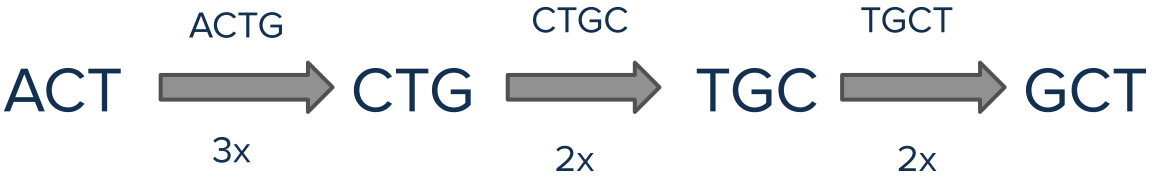


No connection between two graphs because there are no overlaps >= k-1

De Bruijn graphs

- What happens if we add redundancy?
- Previous example for multiple reads and k = 4:

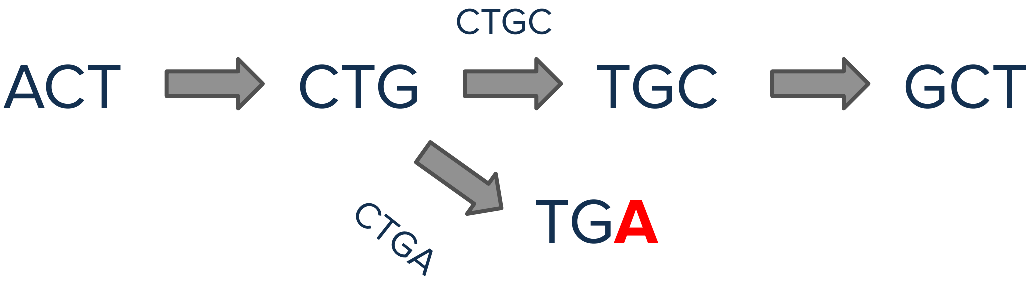
ACTG
ACTG
ACTG
CTGC
CTGC
TGCT
TGCT



De Bruijn graphs

- How does a sequencing error impact de Bruijn graph?
- Previous example for multiple reads and k = 4:

ACTG
ACTG
ACTG
CTGC
CTGA
TGCT
TGCT



De Bruijn graphs

- What is the effect of a small repeat on the graph?
- Example for multiple reads and k = 4:

A**CTG**

CTGC

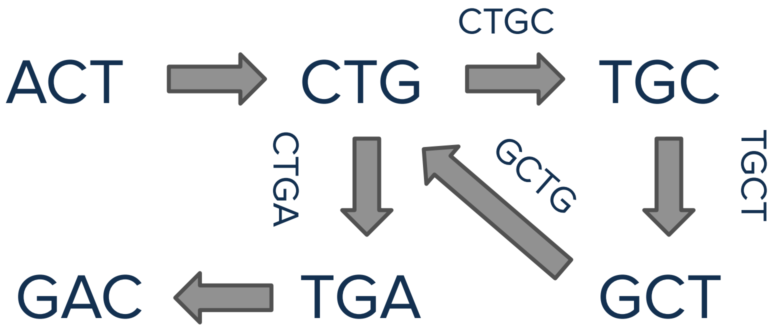
TGCT

G**CTG**

CTGA

TGAC

ACTGCTG_AC_



Summary

- Given any sequence and k-mer size, we can create a De Bruijn graph in an unique manner.
- The other direction is not true. All De Bruijn graphs cannot be resolved into unique sequences. Unless the De Bruijn graph is in its simplest form, it usually resolves into many possible sequences.
- Larger the k-mers, more easy it is to convert the De Bruijn graph into an unique sequence.

Summary

What are the limitations of De Bruijn graphs?

- Reads are immediately split into shorter k-mers; can’t resolve repeats as well as overlap graph
- Read coherence is lost. Some paths through De Bruijn graph are inconsistent with respect to input reads.

- Single most important benefit of De Bruijn graph is speed and simplicity.

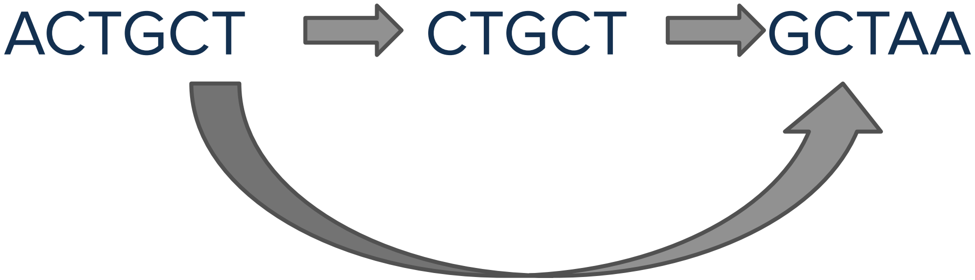
String graphs

- String graph is a modification of an overlap graph.
- Overlap graph:
  - **nodes** – reads
  - **edges** – two nodes are connected if they overlap
- What does “overlap” mean? Similar to “align”, no silver bullet

Overlap graph

- Given **p > 0**, we say that r1 and r2 overlap if a suffix of r1 of **length >= p** is exactly a prefix of r2 of same length
- Example for multiple reads and p = 3:

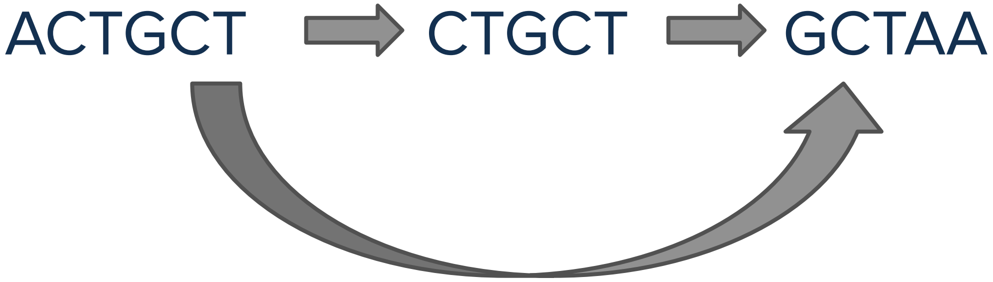
```
ACTGCT
 CTGCT
    GCTAA
```

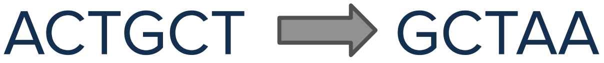


ACTGCT CTGCT GCTAA

String graph

- A string graph is obtained from an overlap graph by removing redundancy:
  - **redundant reads** (those fully contained in another read)
  - **transitively redundant edges** (if **x -> y** and **y -> z**, then keep only **x -> z**)
- Overlap and string graph for previous example:

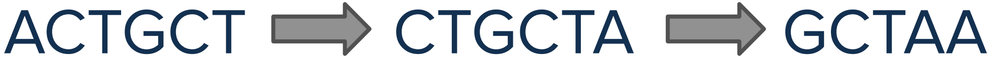


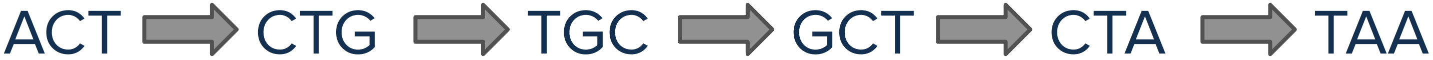


String vs De Bruijn graph

String and De Bruijn graph for the same example:

```
ACTGCT
 CTGCTA
   GCTAA
```





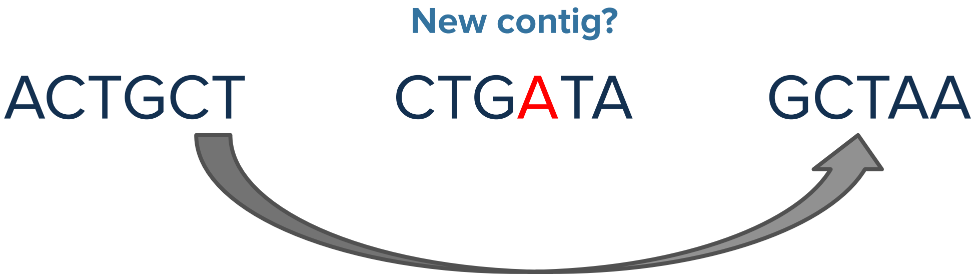
String vs De Bruijn graph

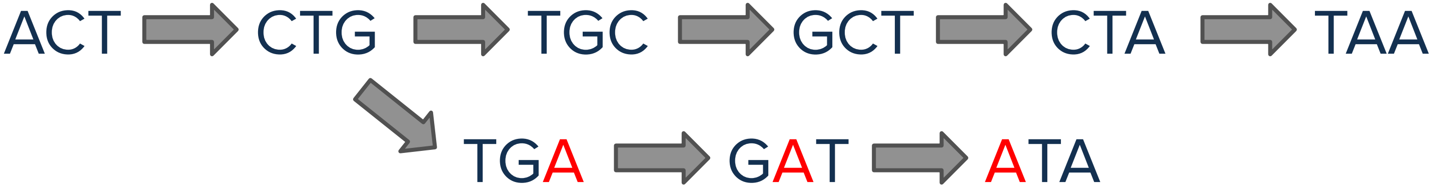
Previous example, with error:

ACTGCT

CTG**A**TA

GCTAA





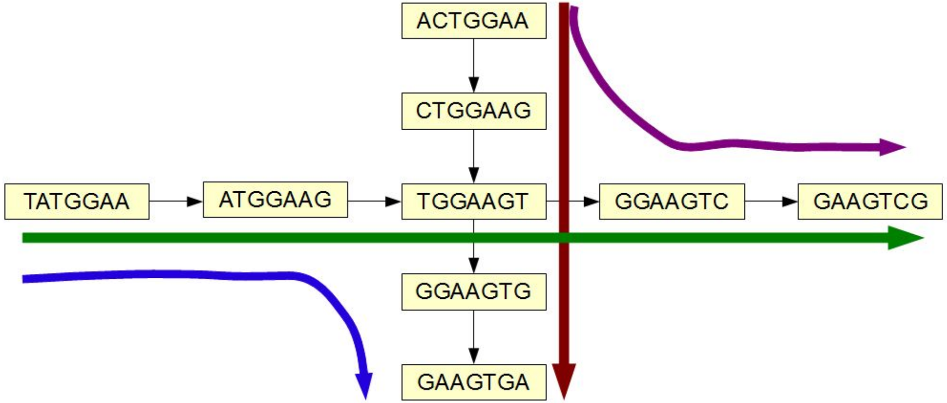
String vs De Bruijn graph

- Which is better?
  - String graphs capture whole read information
  - de Bruijn graphs are conceptually simpler:
    - single node length
    - single overlap definition

- Historically, **string graphs** were used for **long reads**
- and **De Bruijn** graphs for **short reads**

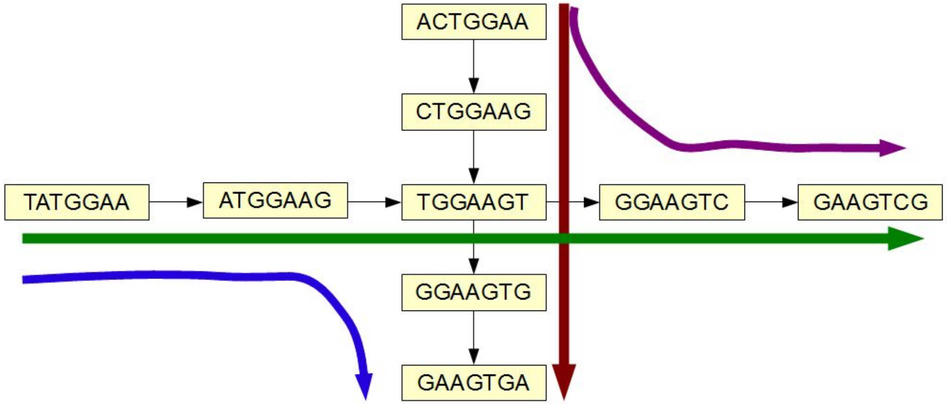
De Bruijn graph

- How do we choose **k**?
- Large **k**:
  - sequencing errors bigger problem
  - graph less connected
- Small **k**:
  - ambiguous paths
- Balance, try different values for **k**

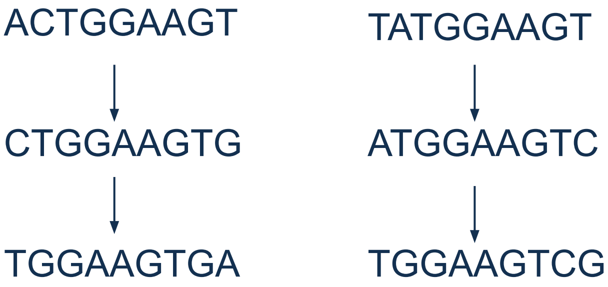


Original sequences – TATGGAAGTCG, ACTGGAAGTGA

De Bruijn graph



Original sequences – TATGGAAGTCG, ACTGGAAGTGA



k=10

k=8

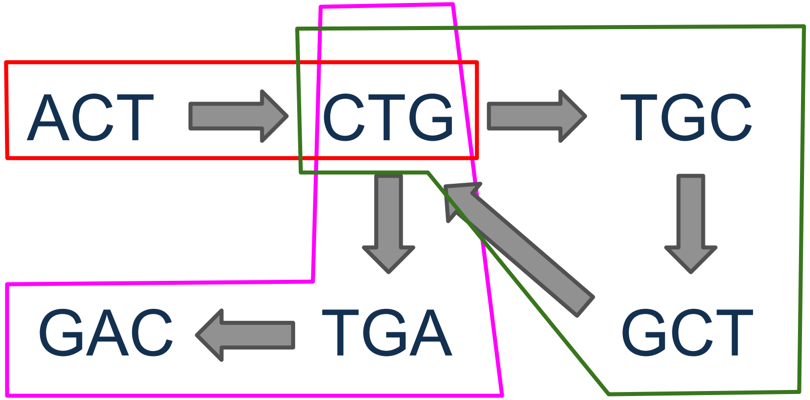
Contigs construction

Contigs construction in practice: return a set of paths covering the graph, such that all possible assemblies contain these paths

ACTG

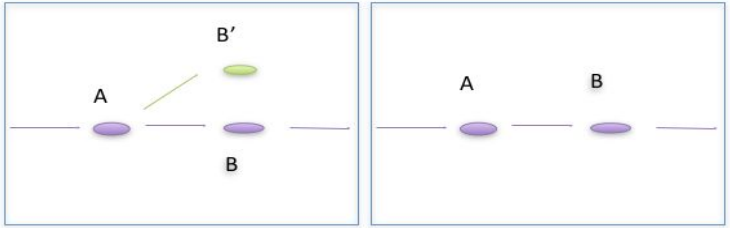
CTGCT

CTGAC

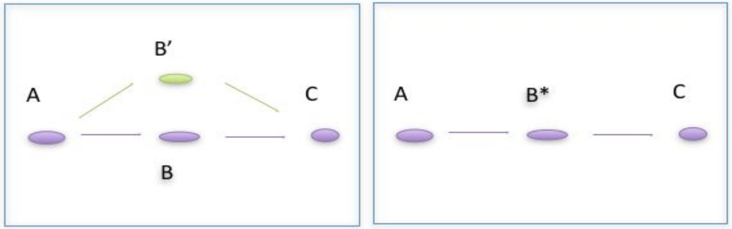


Graph topology based error eorrection

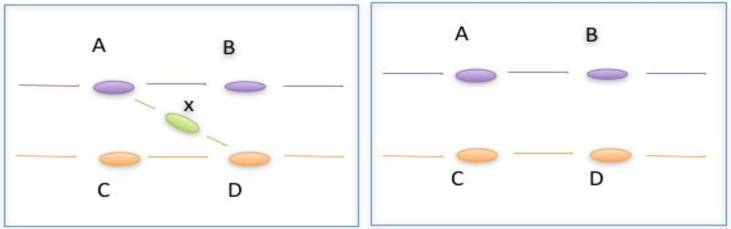
- Errors at end of read
    - Trim off ‘dead-end’ tips



- Errors in middle of read
    - Pop Bubbles



- Chimeric Edges
    - Clip short, low coverage nodes



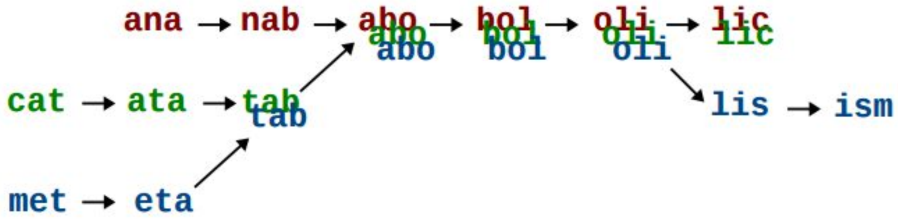
Building De Bruijn graph

K-mer size equal to 4

anabolic
ana
nab
abo
bol
oli
lic

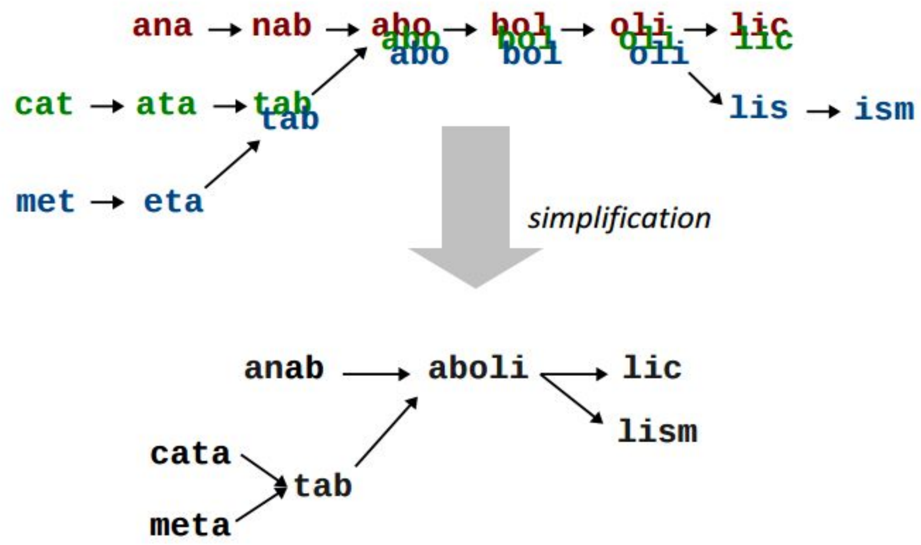
catabolic
cat
ata
tab
abo
bol
oli
lic

metabolism
met
eta
tab
abo
bol
oli
lis
ism

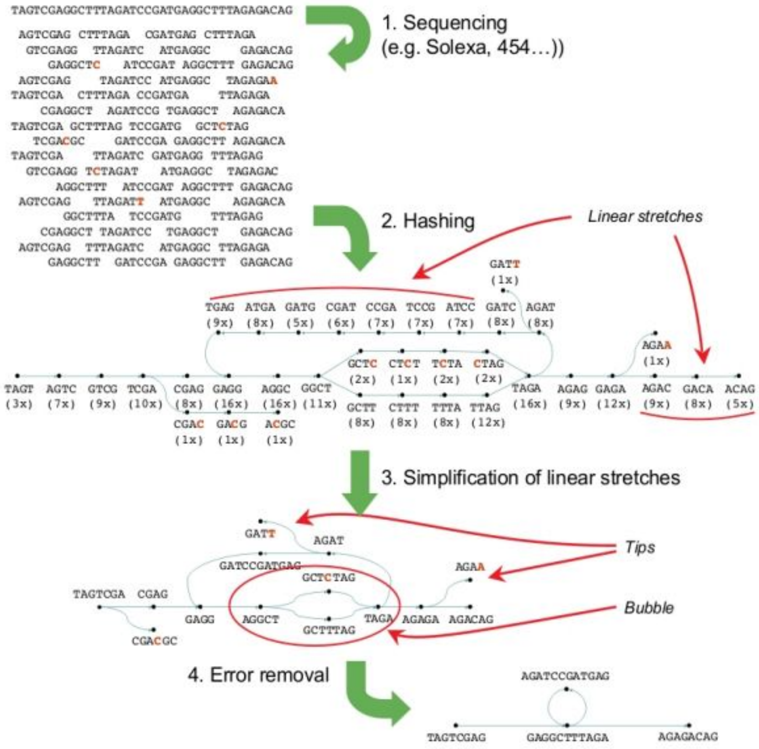


Building De Bruijn graph

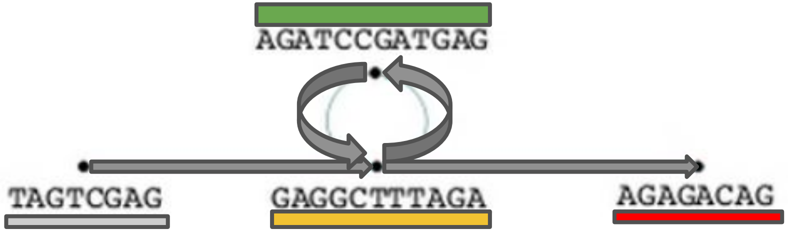
Weight = number of times k-mer occurs



Contigs construction



Contigs construction



Potential assemblies:

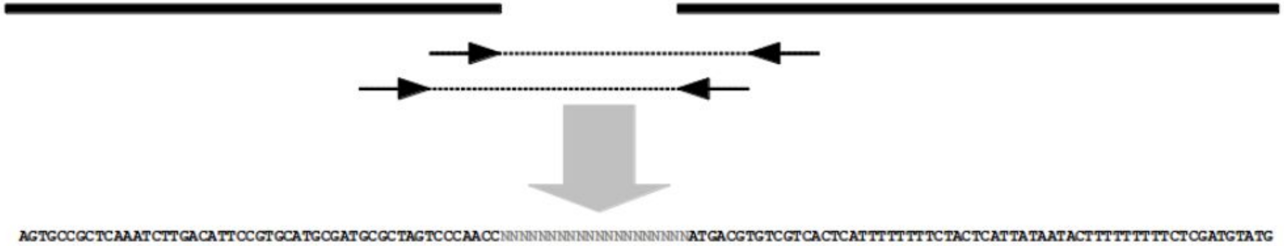
TAGTCGAGGAGGCTTTAGAAGATCCGATGAGGAGGCTTTAGAAGAGACAG

TAGTCGAGGAGGCTTTAGAAGATCCGATGAGGAGGCTTTAGAAGATCCGATGAGGAGGCTTTAGAAG
AGACAG

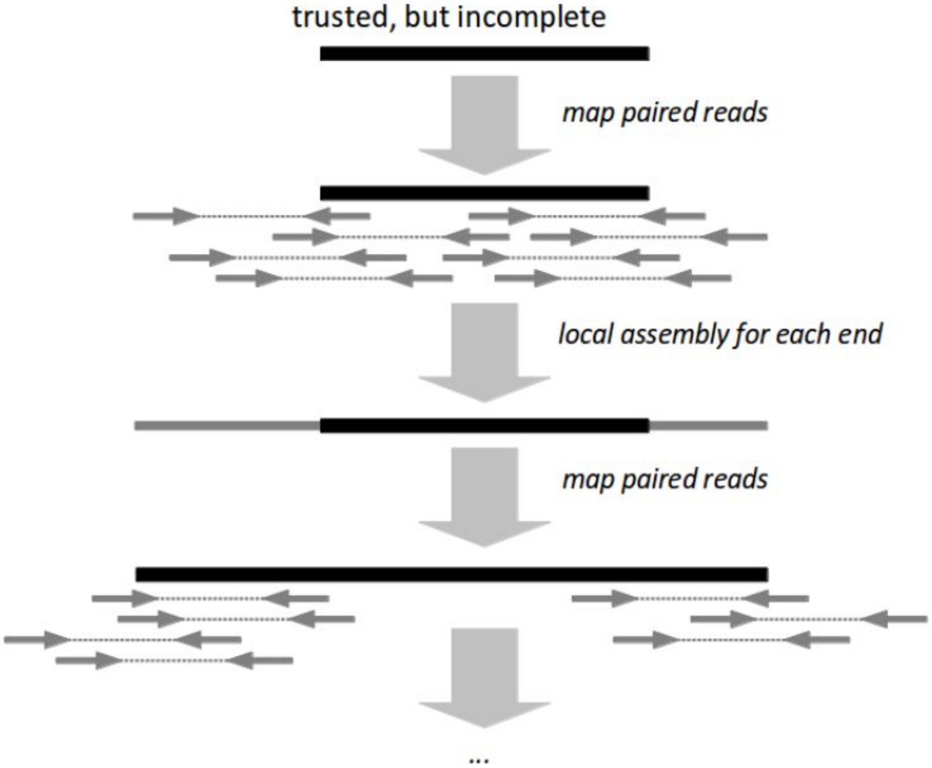
...

Scaffolds construction

- Scaffolding using pairing information
- This is why libraries containing **multiple insert sizes** are used:
  - Gnerre 2011:
    - 45x overlapping PE reads (insert size: 180 bp, reads: >100 bp)
    - 45x short jump MP reads (insert size: 3 kb)
    - 5x (optional) long jump MP reads (insert size: 6 kb)
    - 1x (optional) fosmid jump MP reads (insert size: 40 kb)
  - Ribeiro 2012:
    - 50x overlapping PE reads (insert size: 180 bp, reads: >100 bp)
    - 50x PacBio reads (reads: 1-3 kb)
    - 50x long jump MP reads (insert size: 2-10 kb)

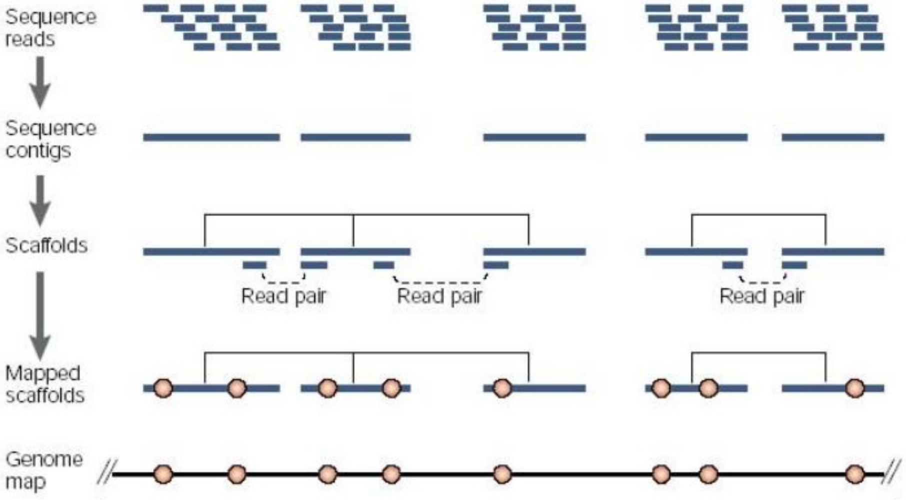

AGTGCCGCTCAAATCTTGACATTCCGTGCATGCGATGCGCTAGTCCCAACCNNNNNNNNNNNNNNNNNNATGACGTGTCGTCACTCATTTTTTTCTACTCATTATAATACTTTTTTTTCTCGATGTATG

Gap closing / contig extension



Summary

de novo whole-genome shotgun assembly

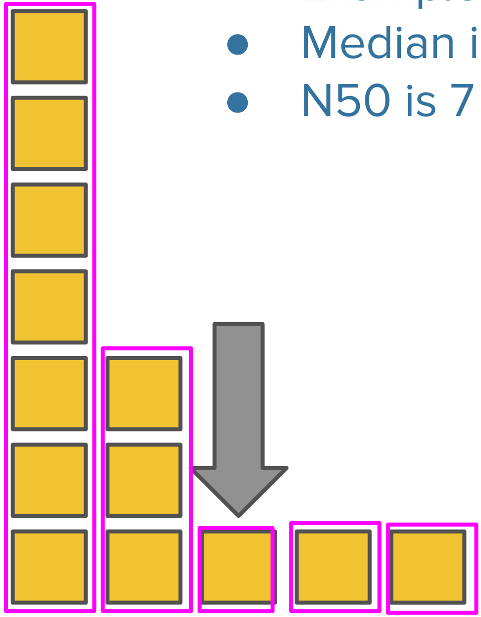


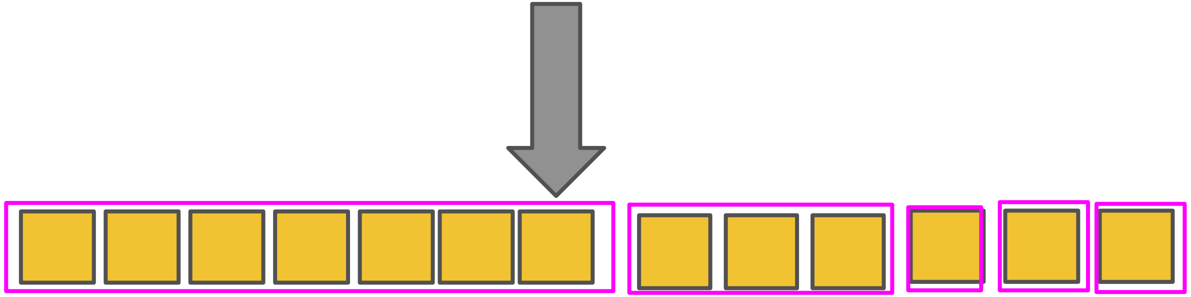
Metrics

- How do we evaluate different assemblies?
- There is no trivial ranking between assemblies
- Reference-free metrics:
    - Number of contigs / scaffolds
    - Total length of the assembly
    - Length of the largest contig / scaffold
    - Percentage of gaps in scaffolds
    - **Nx, NGx of contigs / scaffolds**
    - **Internal consistency**
    - Number of predicted genes
- Reference-using metrics:
    - **Coverage**
    - Assembly errors

Nx / NGx of contigs / scaffolds

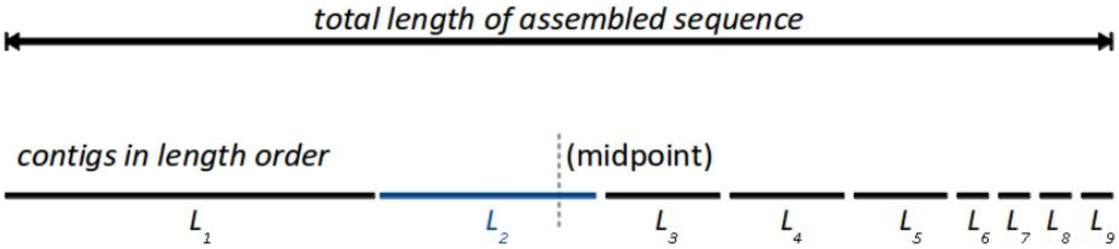
- Nx, where x is percentage value
- **N50** is analogous to **median**
- Example set {7, 3, 1, 1, 1}
- Median is 1
- N50 is 7





Nx / NGx of contigs / scaffolds

- **Nx** is largest contig length at which longer contigs cover **x%** of the total assembly length

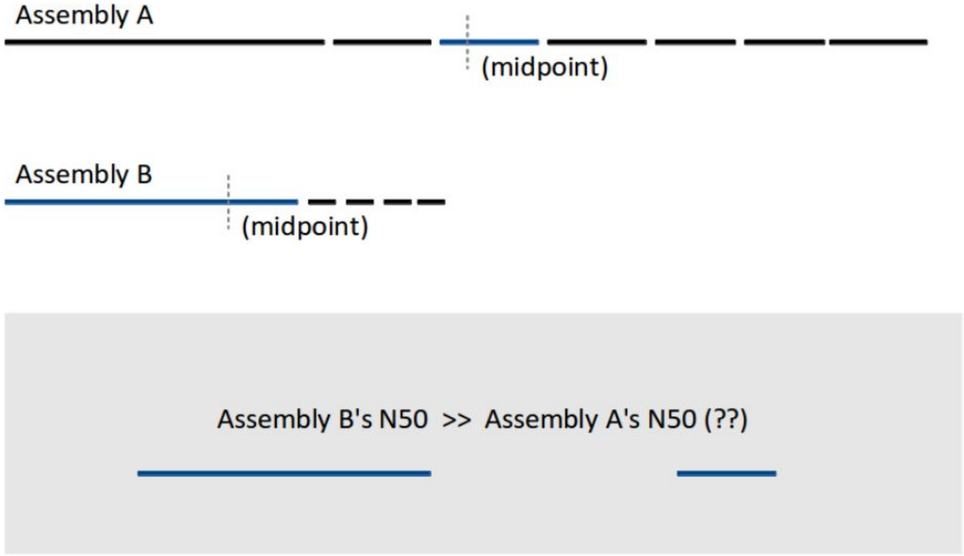


N50 = L_2

- A practical way to compute Nx:
  - Sort contigs by decreasing lengths
  - Take the first contig (the largest): does it cover x% of the assembly?
  - If yes, this is the Nx value. Else, repeat by trying the next one (the second largest)

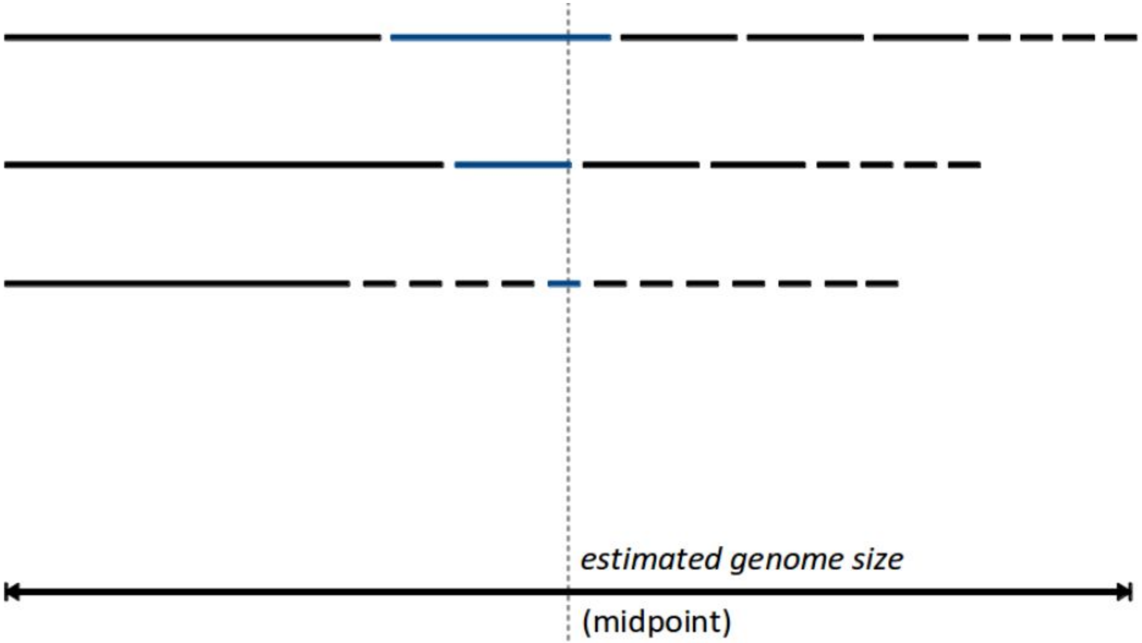
Nx / NGx of contigs / scaffolds

What’s the problem with Nx?



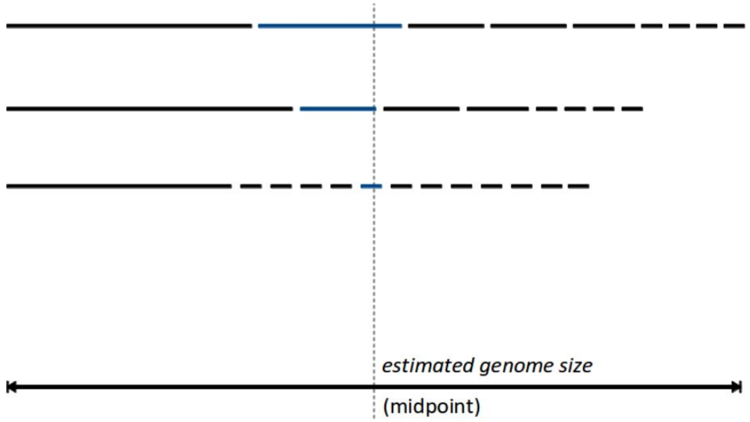
Nx / NGx of contigs / scaffolds

Solution - NGx



Nx / NGx of contigs / scaffolds

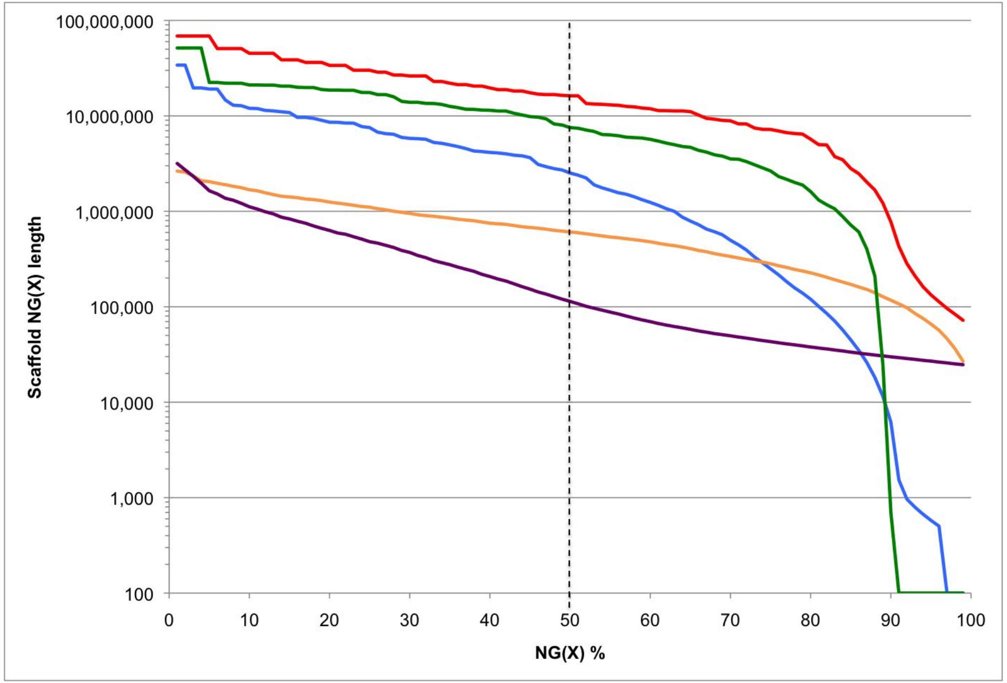
- **NGx** is largest contig length at which longer contigs cover **x%** of the total genome length



- A practical way to compute NGx:
  - Sort contigs by decreasing lengths
  - Take the first contig (the largest): does it cover x% of the genome?
  - If yes, this is the NGx value. Else, repeat by trying the next one (the second largest)

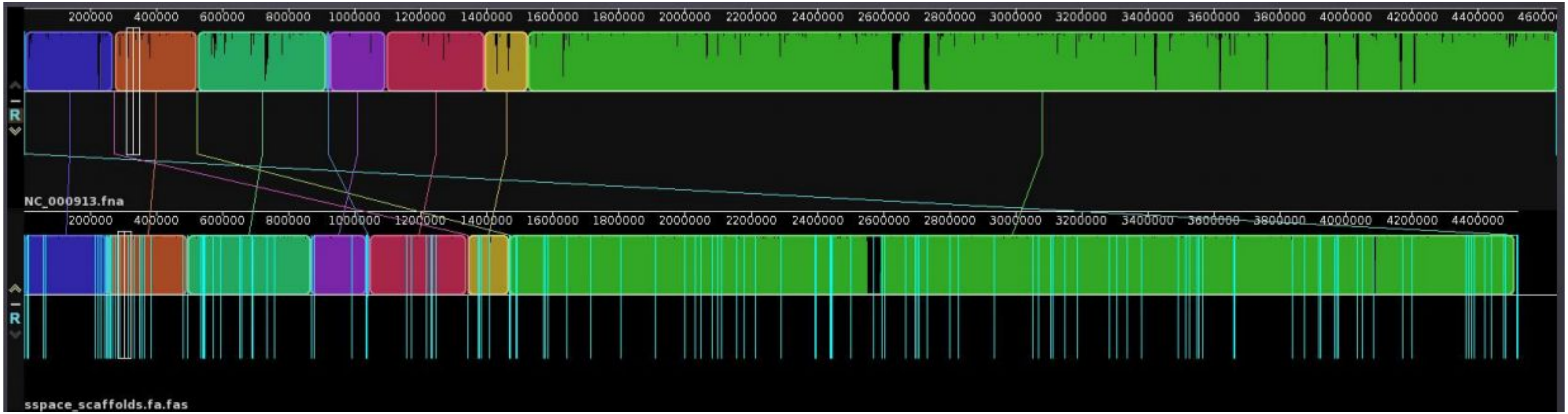
Nx / NGx of contigs / scaffolds

Assemblathon 2



Metrics

- **Internal consistency:** percentage of paired reads correctly aligned back to the assembly (happy pairs)
- **Coverage:** percentage of bases in the reference which are covered by the alignment



Further reading

- A short and nice paper containing essential explanations about assembly graphs - How to apply de Bruijn graphs to genome assembly, http://www.nature.com/nbt/journal/v29/n11/pdf/nbt.2023.pdf

- Assembly lecture from MITs “Foundations of Computational and Systems Biology”: https://www.youtube.com/watch?v=ZYW2AeDE6wU

- Assemblathon - assembly (http://assemblathon.org/) competition and the paper: Assemblathon 2: evaluating de novo methods of genome assembly in three vertebrate species

- Nice assembly ppt presentation - http://www.cs.jhu.edu/~langmea/resources/lecture_notes/assembly_dbg.pdf

- Nice assembly ppt presentation from MIT: https://ocw.mit.edu/courses/biology/7-91j-foundations-of-computational-and-systems-biology-spring-2014/lecture-slides/MIT7_91JS14_Lecture6.pdf

Exercise 1

Write a function “kmerize” which takes in a string (read) and a k-mer length, and returns a list of all unique k-mers present in the string.

Run example:

```
> kmerize("ACGCGTCGC", 3)
> ['ACG', 'CGC', 'GCG, 'CGT', 'GTC', 'TCG']
```

Exercise 1b

Write a function “kmer_count” which takes in a string (read) and a k-mer length, and returns a dictionary containing all unique kmers and number of appearances each of these kmers.

Run example:

```
> kmer_count("AAACCCGCGC", 2)
> {'AA': 2, 'AC': 1, 'CC': 2, 'CG': 2, 'GC': 2}
```

Exercise 2

Write a function “de_bruijn_ize” which takes in a string (read) and a k-mer length, and returns a tuple with all unique k-1-mers (nodes) and a list holding, for each k-mer, its left k-1-mer and its right k-1-mer in a pair (edges).

Run example:

```
> de_bruijn_ize("ACGCGTCG", 3)
> ({'AC', 'GT', 'GC', 'CG', 'TC'},
[('AC', 'CG'), ('CG', 'GC'), ('GC', 'CG'), ('CG', 'GT'),
('GT', 'TC'), ('TC', 'CG')])
```

Exercise 3

Simple De Bruijn Graph implementation with Eulerian walk-finder
Solution (page 16):

http://www.cs.jhu.edu/~langmea/resources/lecture_notes/assembly_dbg.pdf

Code:

http://nbviewer.jupyter.org/github/BenLangmead/comp-genomics-class/blob/master/notebooks/CG_deBruijn.ipynb