DNA Sequencing and Data Analysis

Prof Noam Shomron
Hadas Volkov

Lecture 4, May 2, 2024

DNA Sequencing and Data Analysis

The De-novo Shotgun Assembly Problem

Thursday 18:30 to 21:00

Hangar H2

nshomron@gmail.com

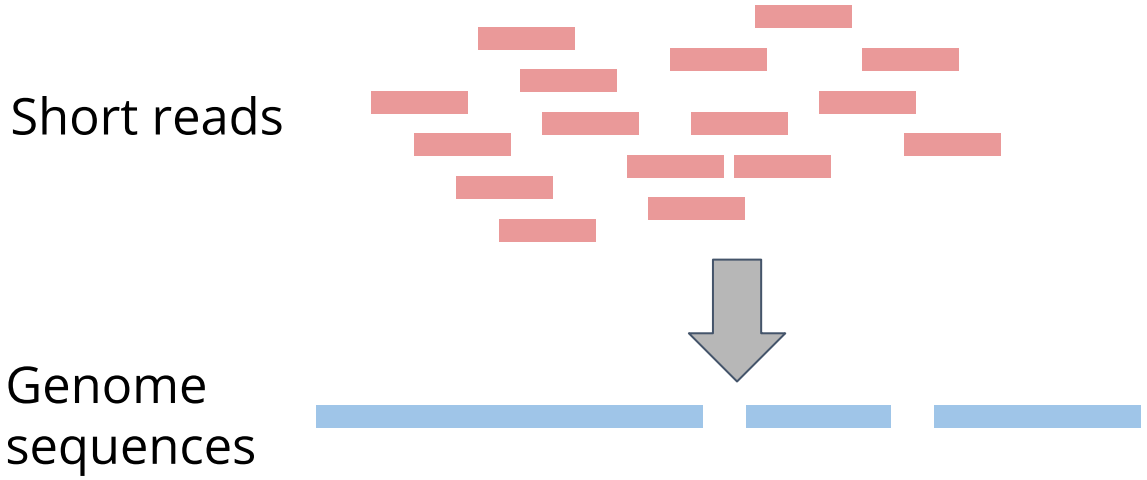
hadas.volkov@post.runi.ac.il

What is De Novo Genome Assembly?

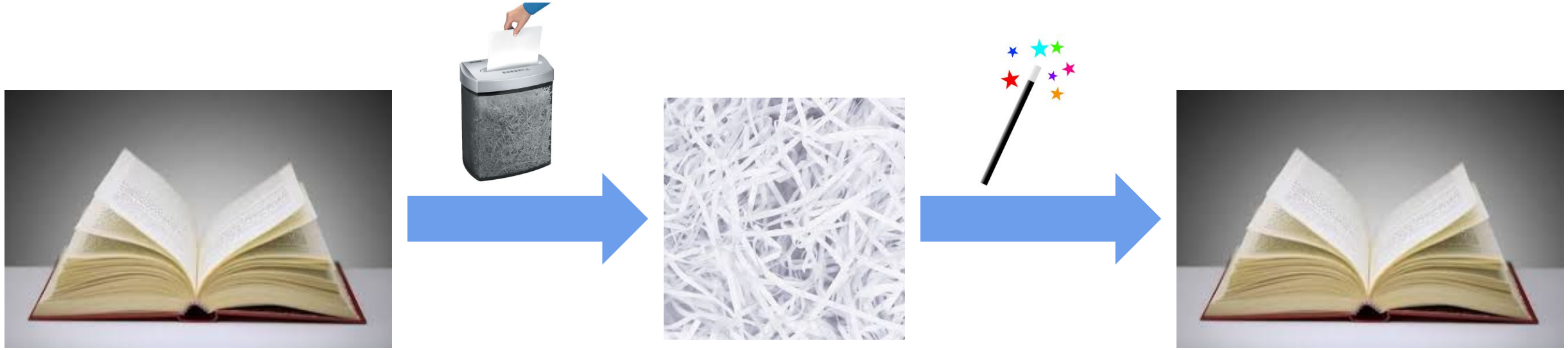
Genome assembly - constructing long genomic sequences from shorter ones

De novo = “from scratch”

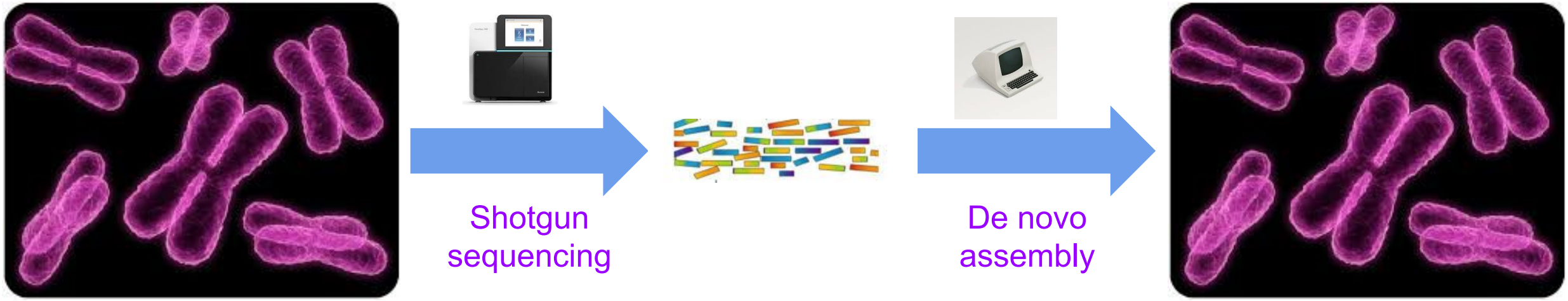
In NGS context - short reads → whole genome, without any external reference



The Assembly Problem

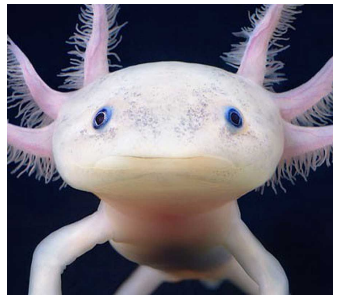


The Assembly Problem



Why Do We Need De Novo Assembly?

Completely new organism



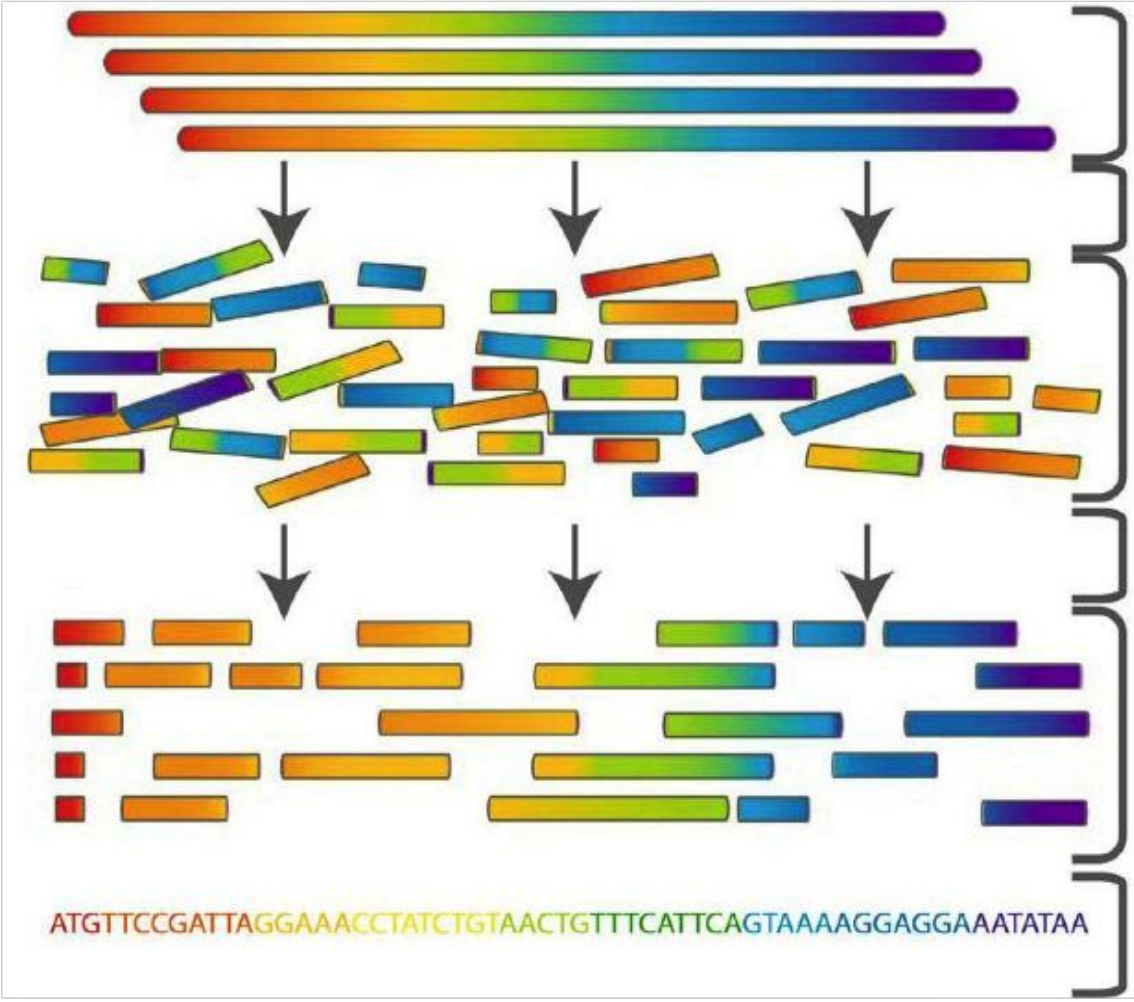
Existing reference is too different from what we are interested in

Identify large structural variation

Detect novel sequences not present in the reference

Cancer genomics

Genome Assembly by Reads Overlap



Genomic DNA

Fragmentation + Sequencing

Sequence reads

Assembly

Connection between reads found

Consensus sequence

Coverage Definition

CTAGGCCCTCAATTTT
CTCTAGGCCCTCATTTTT
GGCTCTAGGCCCTCATTTTT
CTCGGCTCTAGGCCCTCATTT
TATCTCGACTCTAGGCCCTCA
TATCTCGACTCTAGGCC
TCTATATCTCGGCTCTAGG
GGCGTCTATATCTCG
GGCGTCTATATCT
GGCGTCTATATCT
GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT

Coverage = 5

Average Coverage

CTAGGCCCTCAATTTT
CTCTAGGCCCTCATTTTT
GGCTCTAGGCCCTCATTTTT
CTCGGCTCTAGGCCCTCATTT
TATCTCGACTCTAGGCCCTCA
TATCTCGACTCTAGGCC
TCTATATCTCGGCTCTAGG
GGCGTCTATATCTCG
GGCGTCTATATCT
GGCGTCTATATCT

177 bases

35 bases

GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT

Average Coverage = 177/35 ~= 5-fold (5x)

Suffix Prefix Matching

TCTATATCTCGGCTCTAGG

TATCTCGACTCTAGGCC

Suffix Prefix Matching

TCTATATCTCGGCTCTAGG

TATCTCGACTCTAGGCC

Suffix Prefix Matching

TCTATATCTCGGCTCTAGG

GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT

TATCTCGACTCTAGGCC

If a suffix of read A is similar to a prefix of read then A and B might overlap in the genome

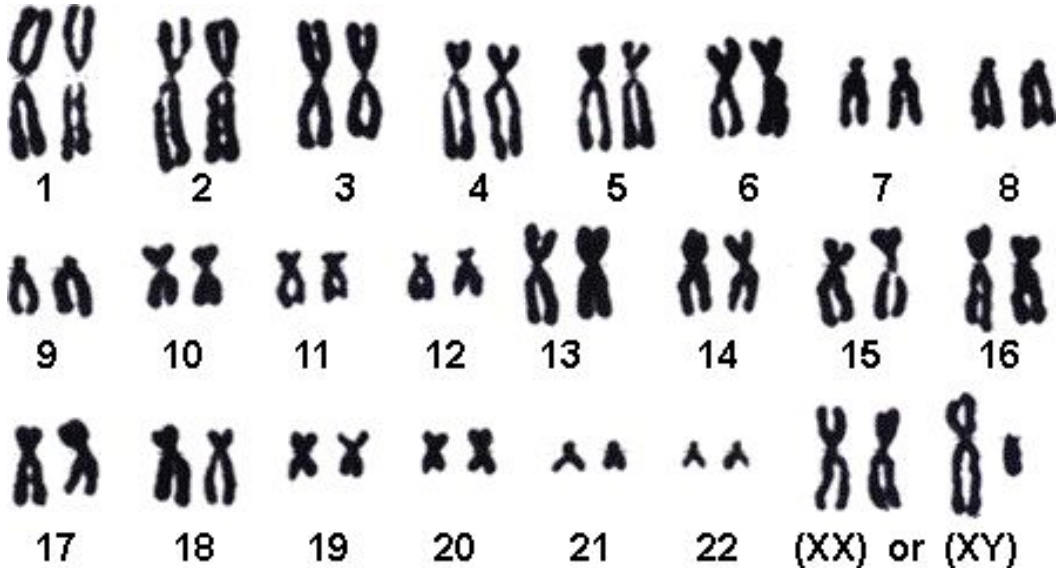
Suffix Prefix Differences

TCTATATCTCGGCTCTAGG

TATCTCGACTCTAGGCC

Why the differences?

1. Sequencing errors
2. Polyploidy



High and Low Coverage

CTAGGCCCTCAATTTT
GGCTCTAGGCCCTCATTTTT
CTCGGCTCTAGGCCCTCATTT
TATCTCGACTCTAGGCC
TCTATATCTCGGCTCTAGG
GGCGTCTATATCTCG
GGCGTCTATATCT

More coverage

GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT
CTAGGCCCTCAATTTT
TATCTCGACTCTAGGCCCTCA
GGCGTCTATATCT

Less coverage

More coverage leads to more and longer overlaps

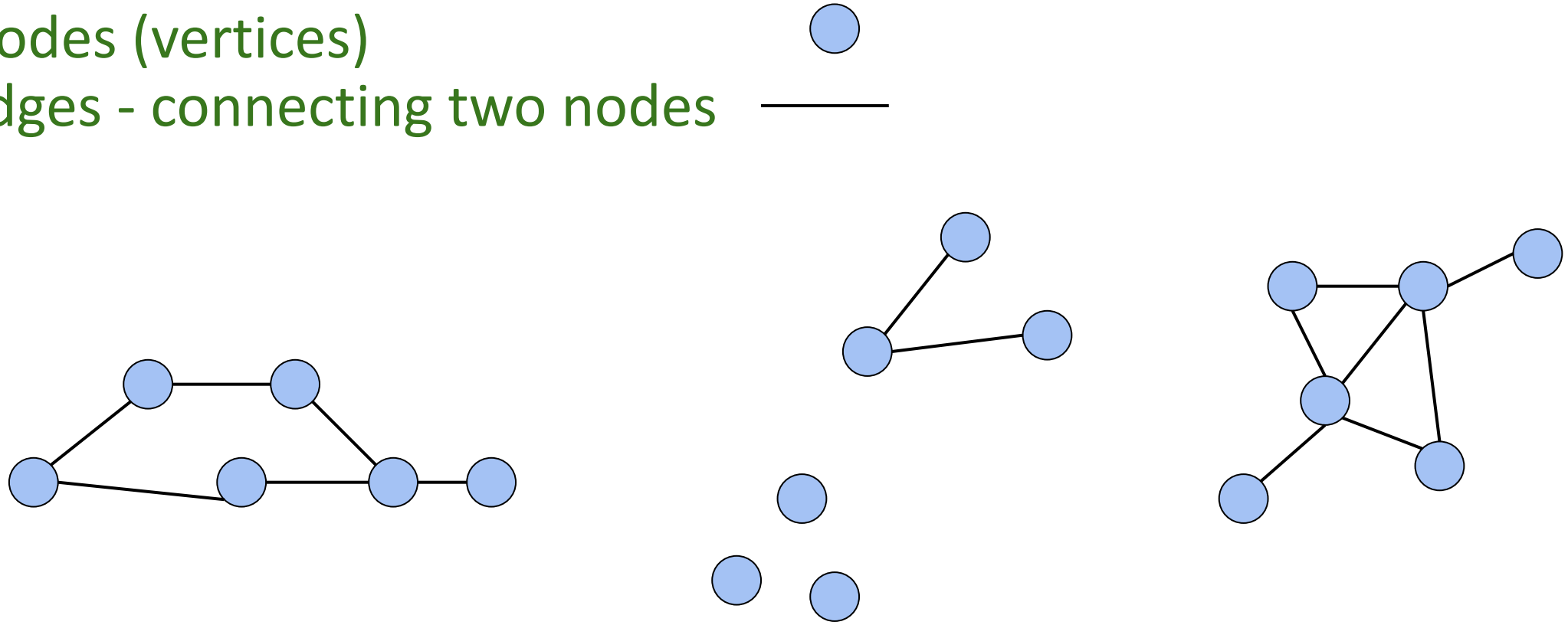
Overlap Consensus

What is the best representation to the set of sequences?

Graph

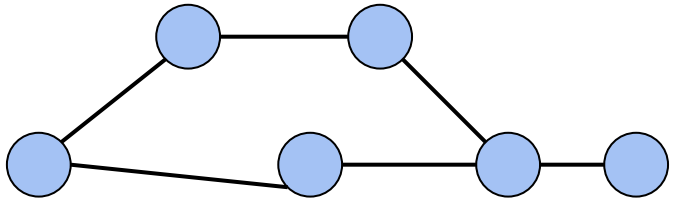
A graph is a set of:

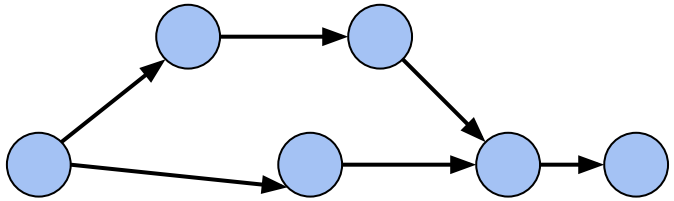
- Nodes (vertices)
- Edges - connecting two nodes

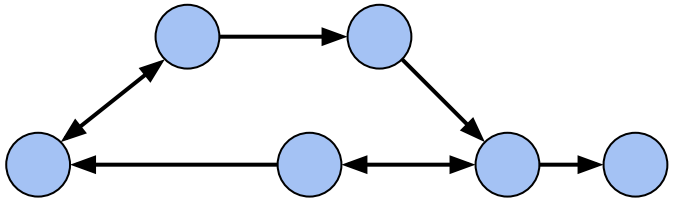


Directed and Weighted Graphs

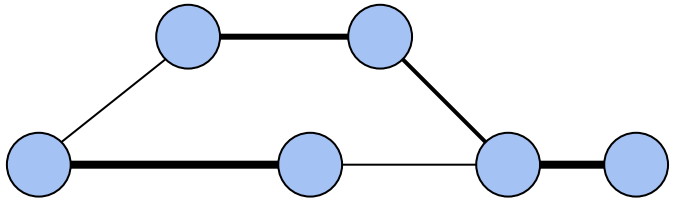
Graphs can be **directed** or **non-directed**

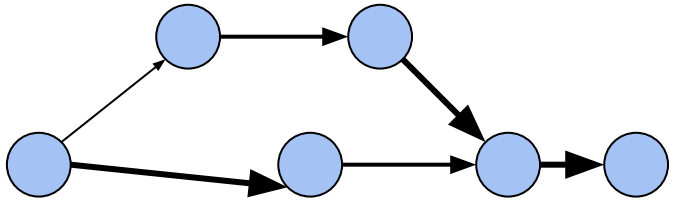






We can assign **weights** to edges



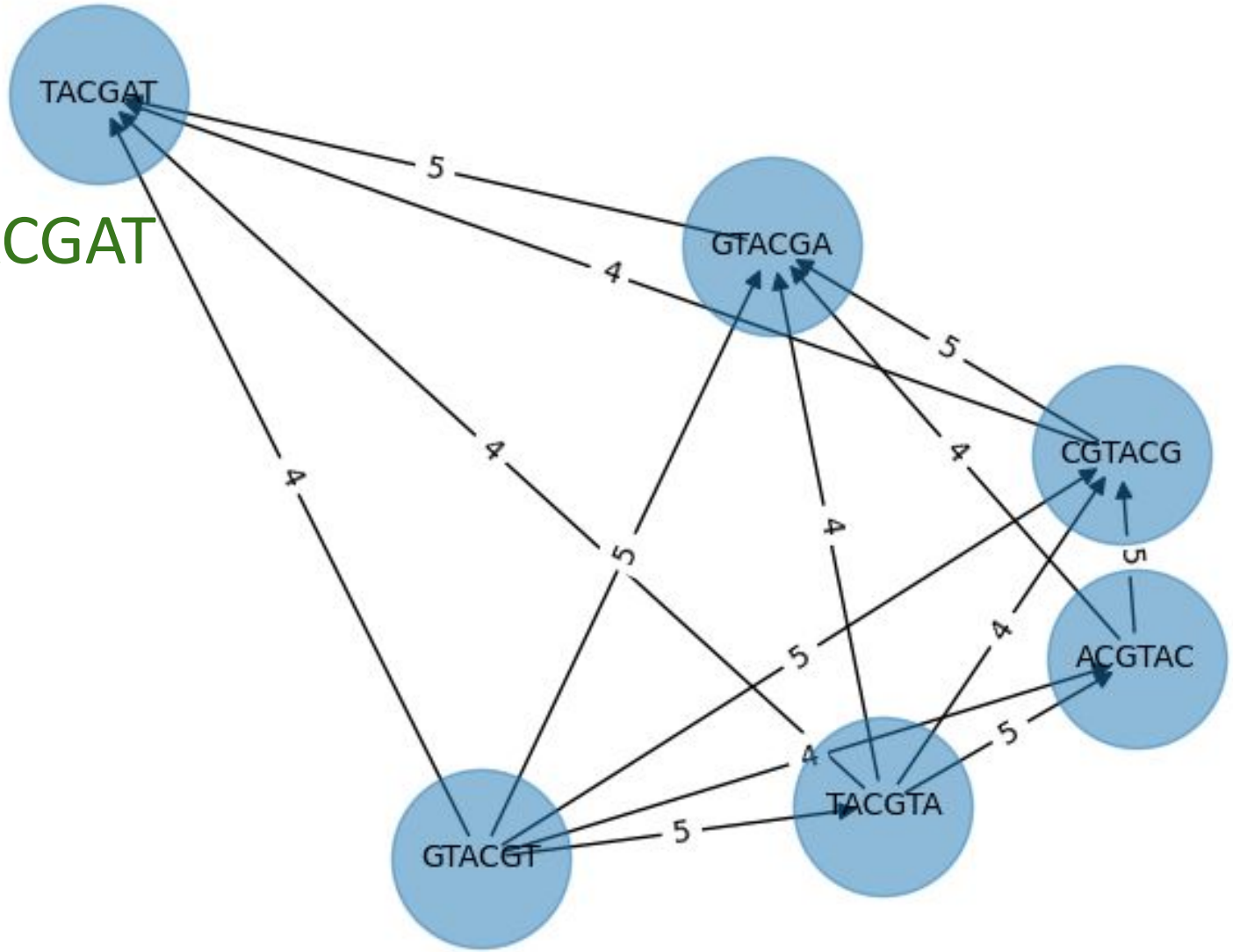


Overlap Consensus Graph

Nodes: all 6-mers in GTACGTACGAT

Edges: overlaps of length > 3

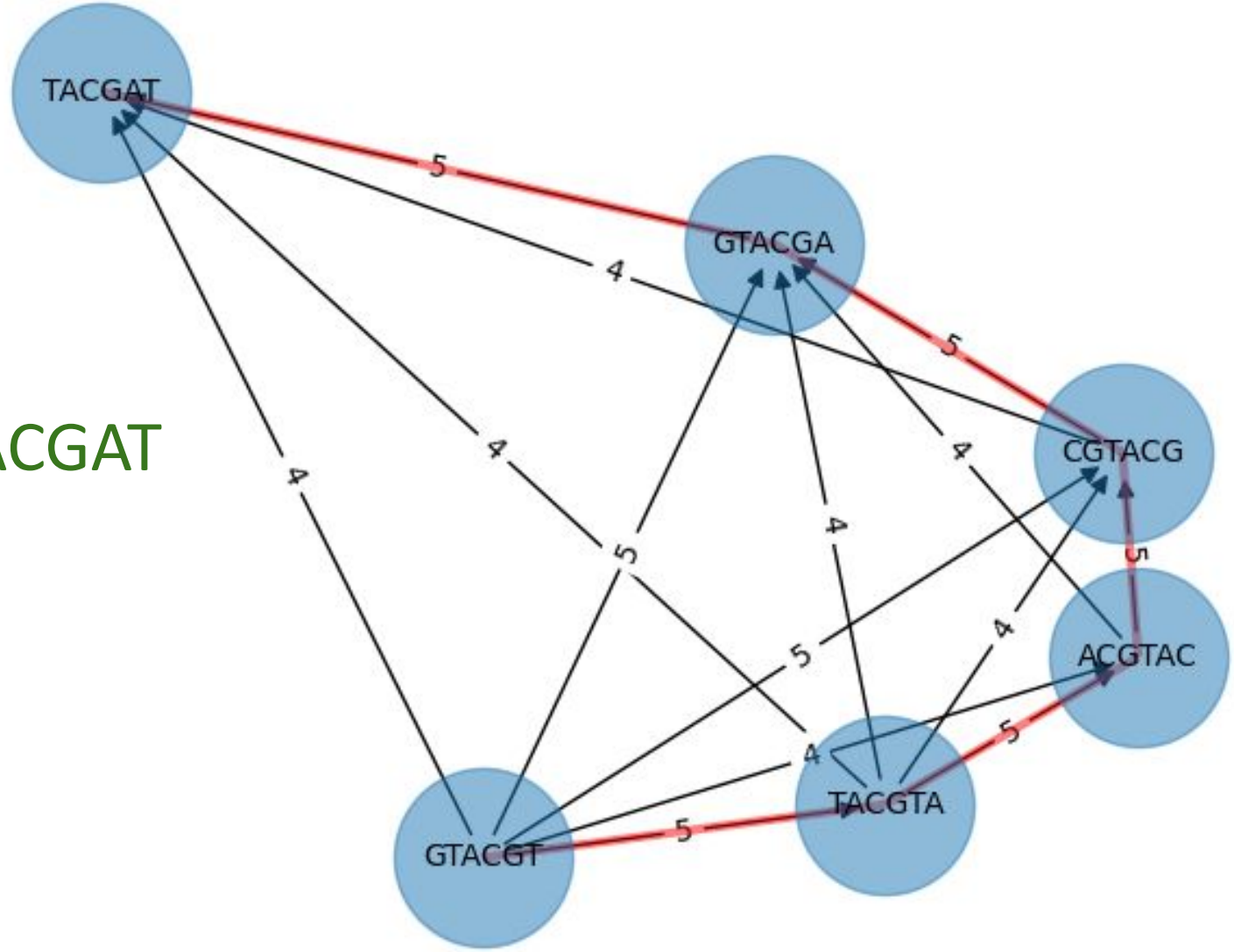
GTACGT
TACGTA
ACGTAC
CGTACG
GTACGA
TACGAT



Overlap Consensus Graph

Nodes: all 6-mers in GTACGTACGAT

Edges: overlaps of length > 3



Shortest Common Superstring

The Shortest Common Superstring problem (SCS) aim to find the shortest possible string that contains every string in a given set as substrings

Example: BAA AAB BBA ABA ABB BBB AAA BAB

Concatenation: BAAAABBBAABAABBBBBAAABAB

24

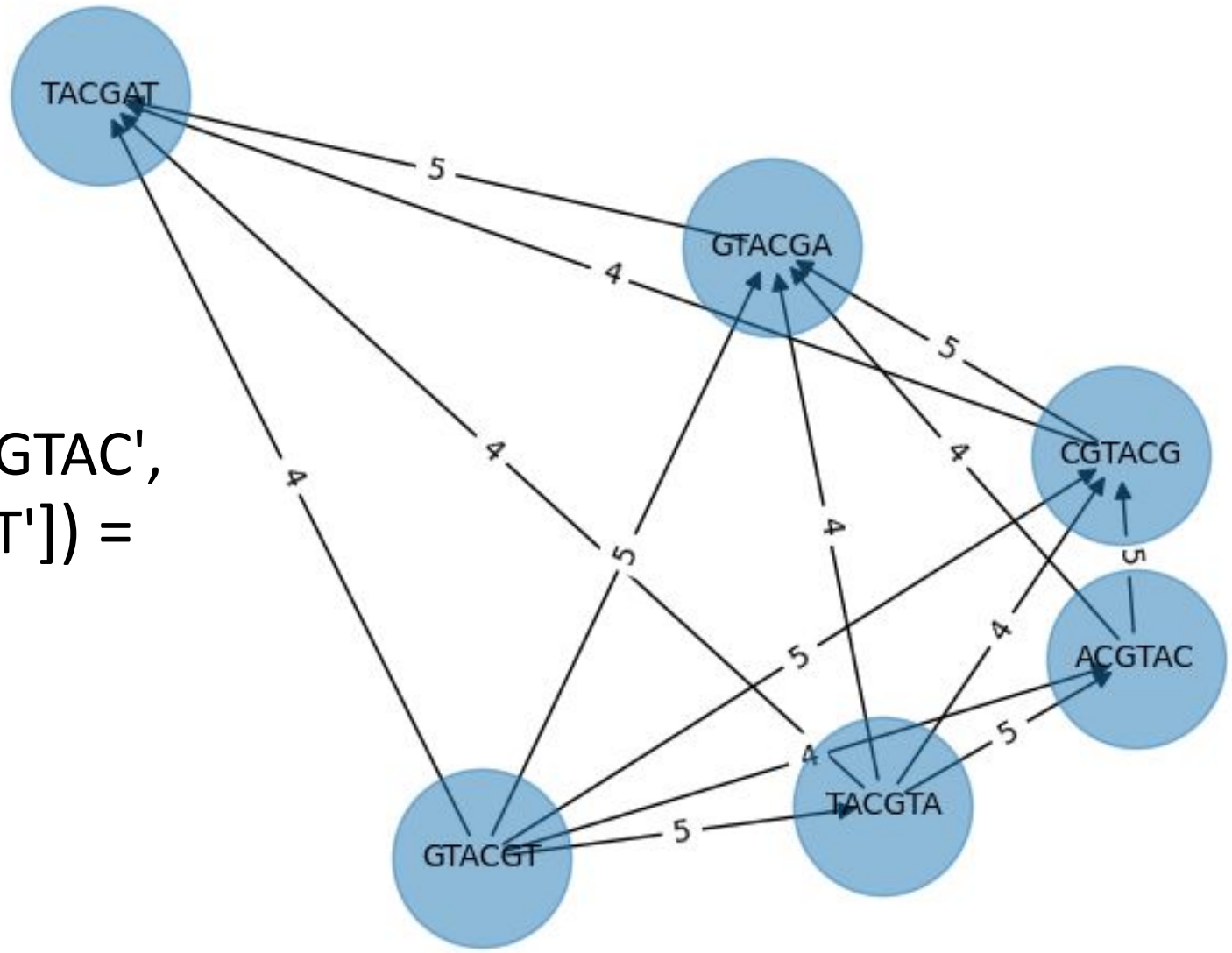
AAA
AAB
ABB
BBB
BBA
BAB
ABA
BAA

SCS: AAABBBABAA

10

Shortest Common Superstring

SCS(['GTACGT', 'TACGTA', 'ACGTAC', 'CGTACG', 'GTACGA', 'TACGAT']) = GTACGTACGAT



Shortest Common Superstring

Brute Force

Order 1: AAA AAB ABA ABB BAA BAB BBA BBB

AAABABBAABABBABBB Superstring 1

Order 2: AAA AAB ABA BAB ABB BBB BAA BBA

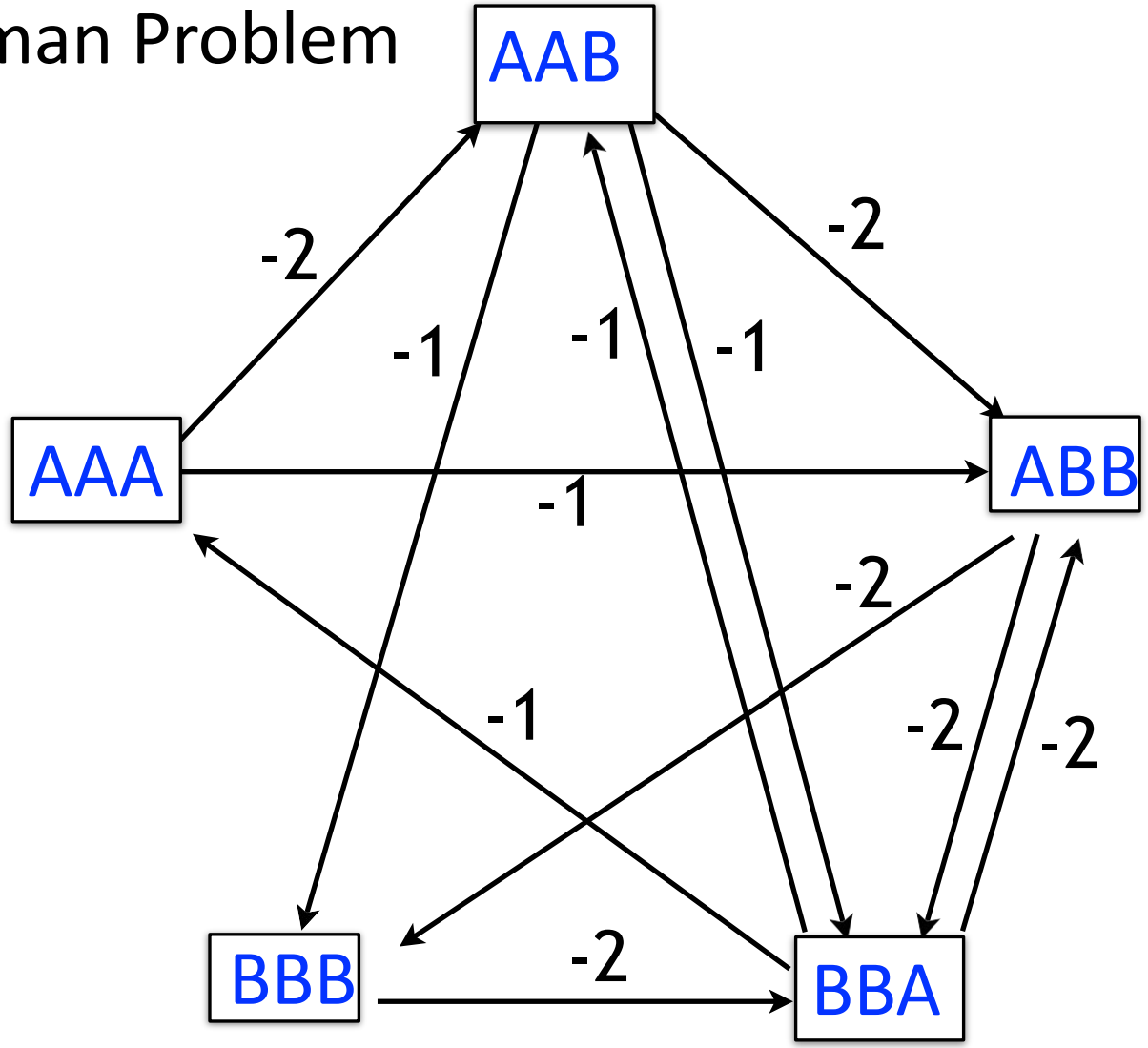
AAABABBBAABBA Superstring 2

O(n!)

Shortest Common Superstring

Traveling Salesman Problem

Modified overlap graph where each edge has cost = - (length of overlap)
SCS corresponds to a path that visits every node once, minimizing total cost along path



Shortest Common Superstring

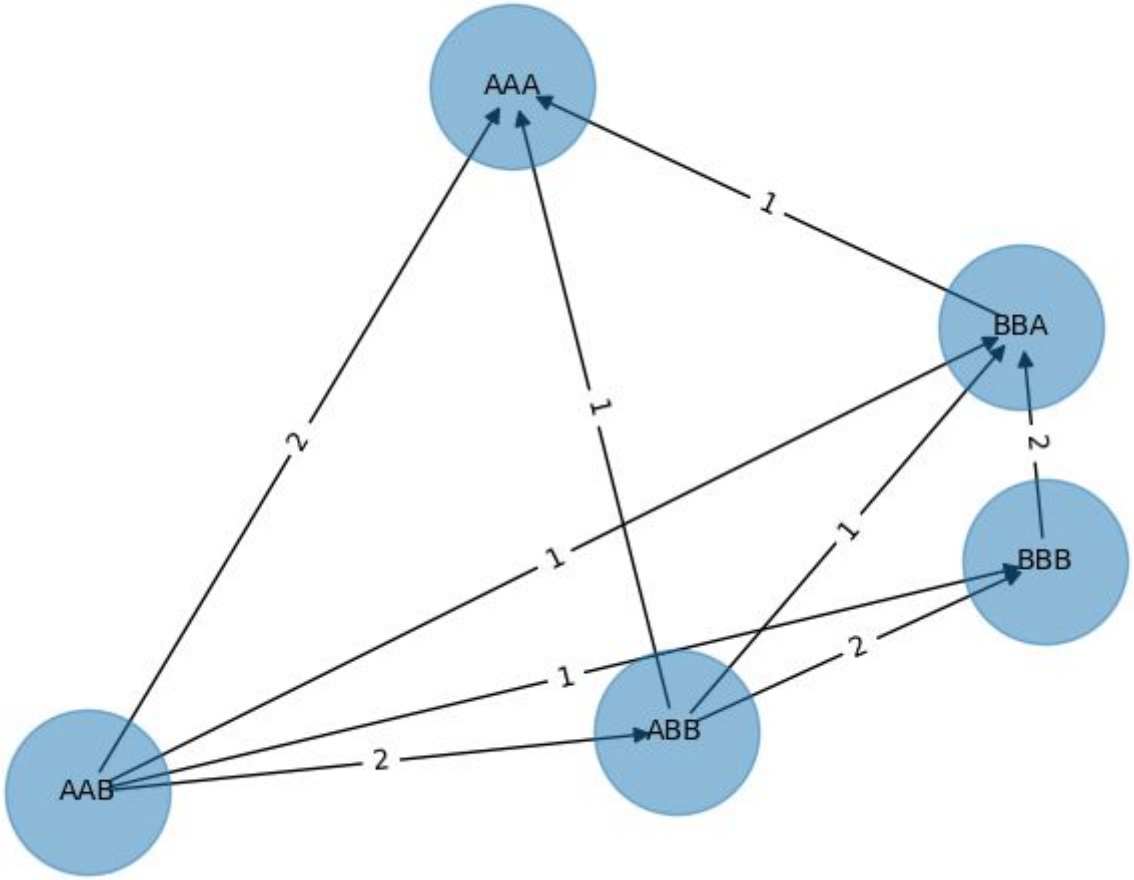
NP-complete

No efficient solution algorithm has been found

Shortest Common Superstring

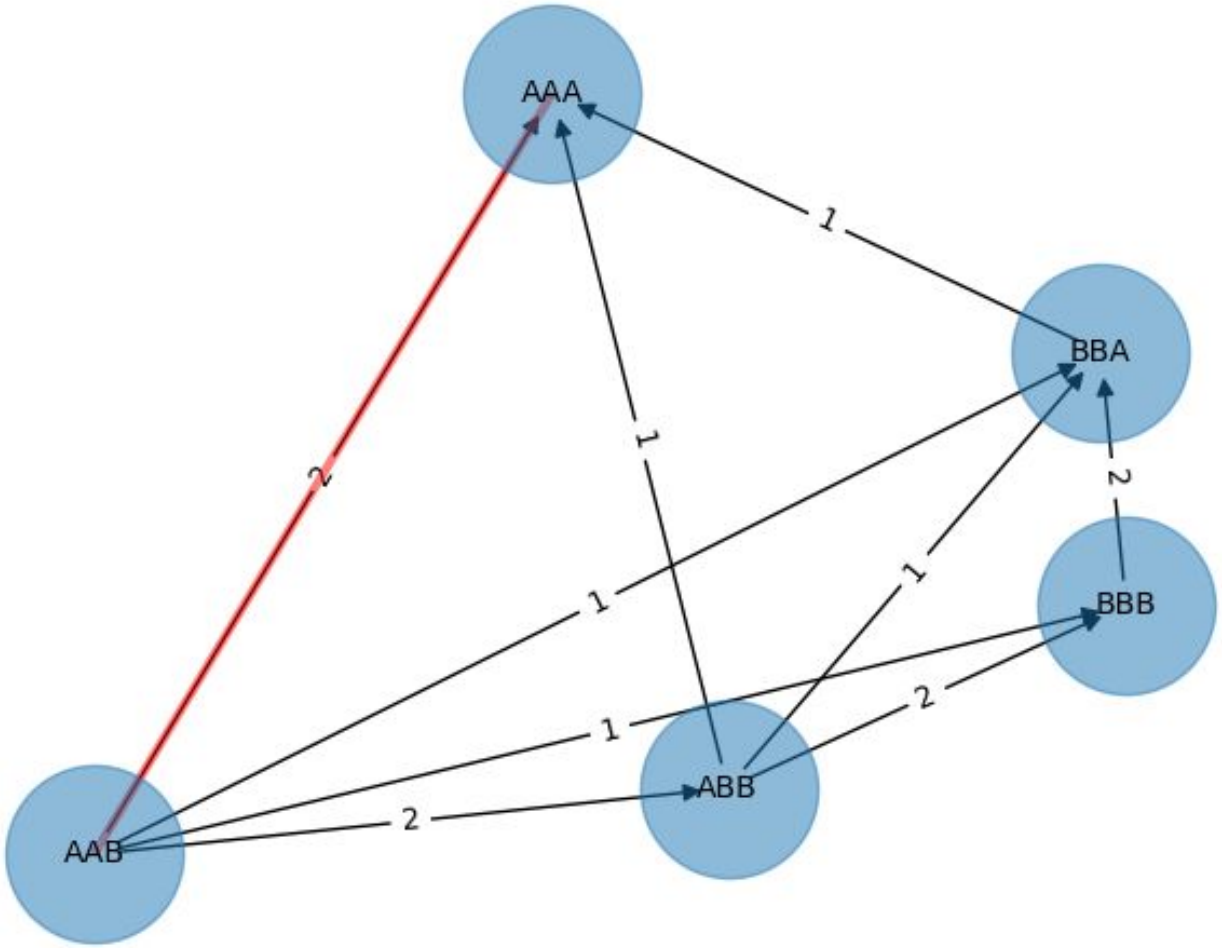
Greedy

Example: BAA AAB BBA ABA ABB BBB AAA BAB



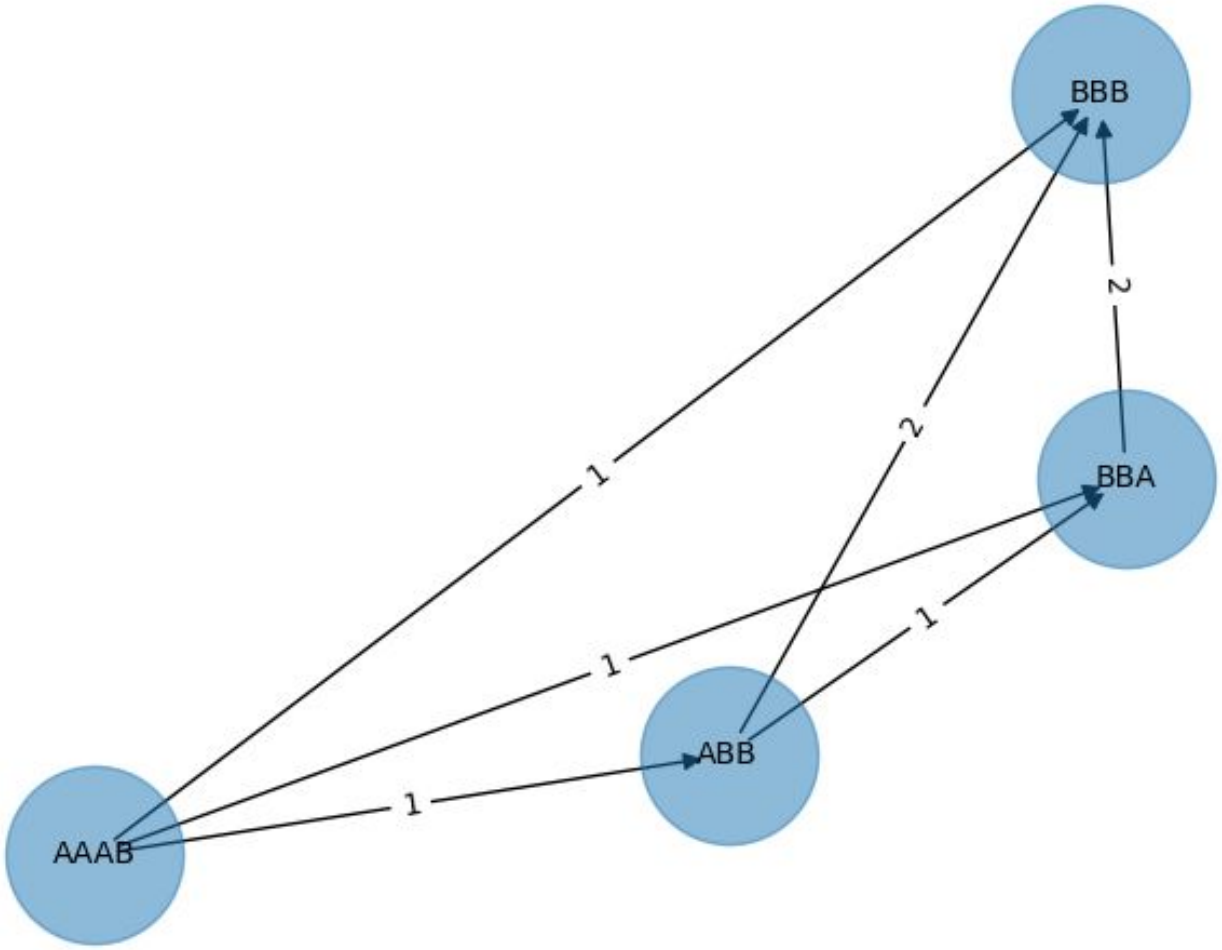
Shortest Common Superstring

Greedy



Shortest Common Superstring

Greedy



Shortest Common Superstring

Greedy

AAABBBA Superstring, length 7

Alternative Shuffling

AAABBABBB Superstring, length 9

Greedy answer isn’t necessarily optimal

Shortest Common Superstring

Greedy

Greedy algorithm is not guaranteed to choose overlaps yielding SCS

But greedy algorithm is a good approximation; i.e. the superstring yielded by the greedy algorithm won't be more than ~2.5 times longer than true SCS

Gusfield, Dan. "Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology." (1997).

Shortest Common Superstring

Greedy

a_*long*_*long*_*long*_time *l=6*

ng_lon _long_ a_long long_l ong_ti ong_lo long_t g_long g_time ng_tim

5 ng_time ng_lon _long_ a_long long_l ong_ti ong_lo long_t g_long

5 ng_time g_long_ ng_lon a_long long_l ong_ti ong_lo long_t

5 ng_time long_ti g_long_ ng_lon a_long long_l ong_lo

5 ng_time ong_lon long_ti g_long_ a_long long_l

5 ong_lon long_time g_long_ a_long long_l

5 long_lon long_time g_long_ a_long

5 long_lon g_long_time a_long

5 long_long_time a_long

4 a_long_long_time

a_*long*_*long*_time

Missing a _long

Shortest Common Superstring

Repeats often foil assembly. They certainly foil SCS, with its “shortest” criterion!

Reads might be too short to “resolve” repetitive sequences. This is why sequencing vendors try to increase read length.

Algorithms that don’t pay attention to repeats (like our greedy SCS algorithm) might *collapse* them

a_long_long_long_time

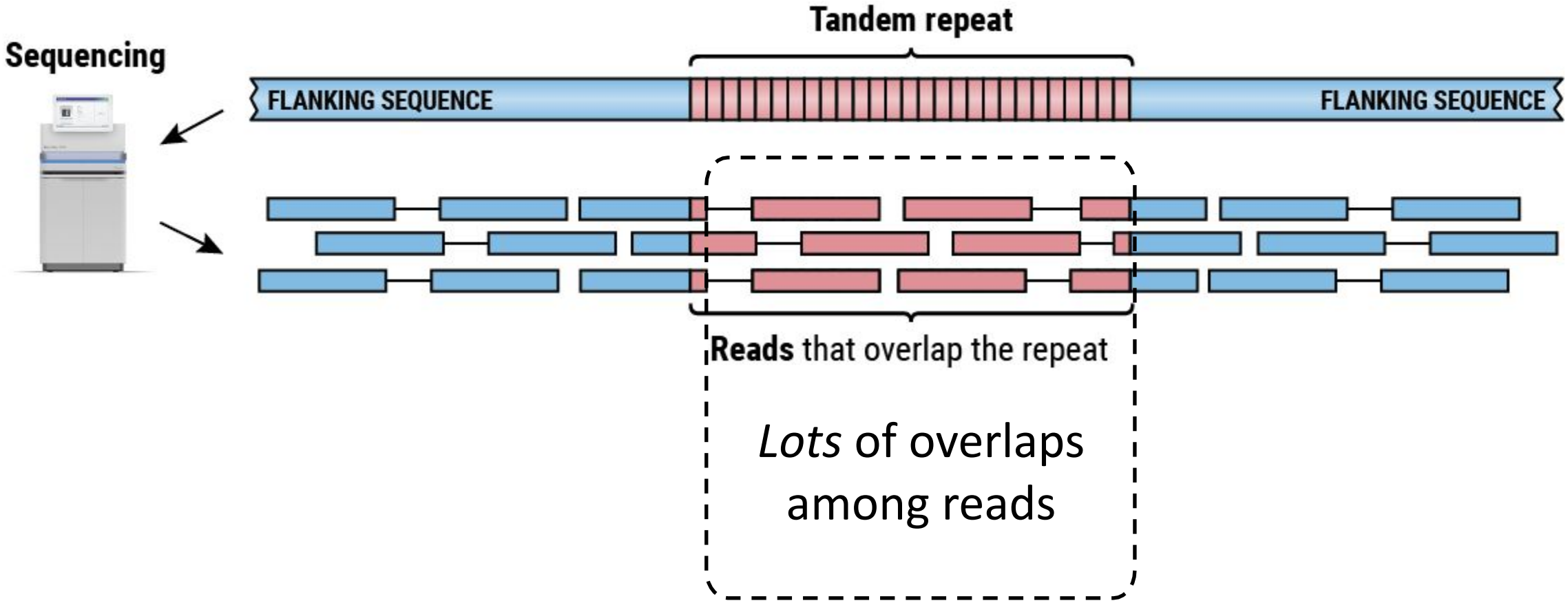
collapse

a_long_long_time

The human genome is ~ 50% repetitive!

Shortest Common Superstring

Repeats



Genome Assembly by Reads Overlap - Challenges

Do we believe short overlaps?

How do we handle sequencing errors?

Computationally ineffective for large data sets

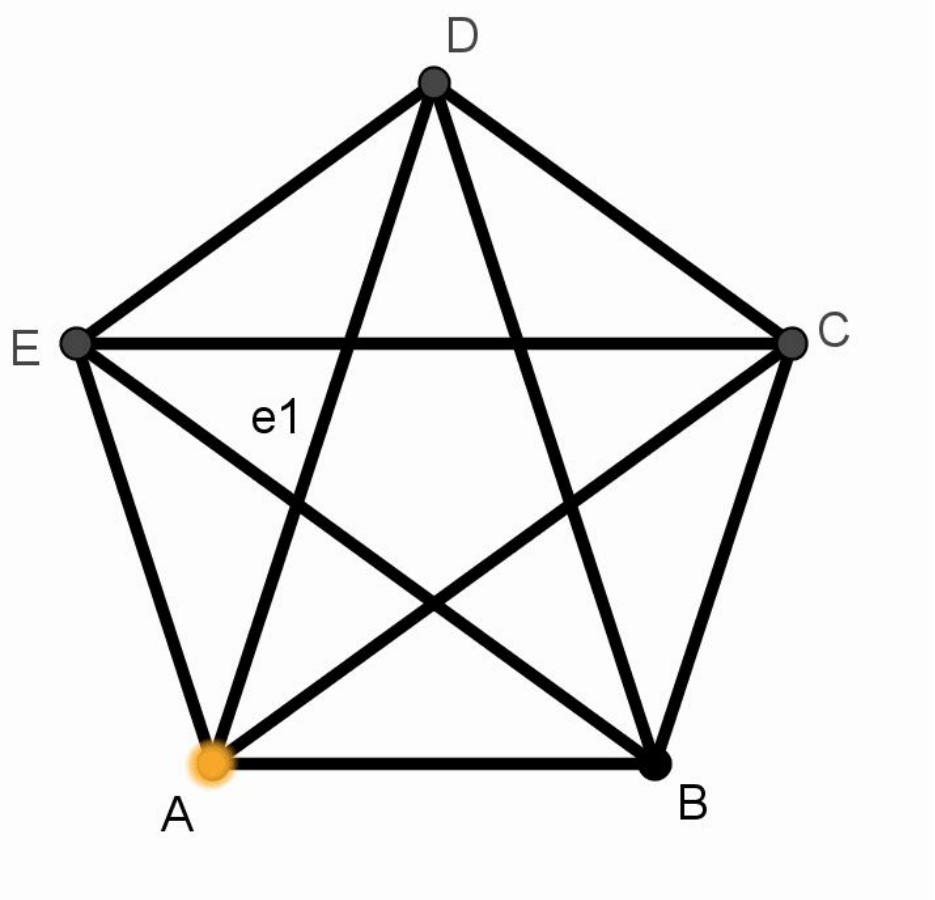
Bottom line: not good enough for large and complex genomes (e.g. human)

We need something smarter!

The Eulerian path

A path in a graph that visits every **edge** exactly once

- Must visit **all** edges
- Can't visit an edge twice
- Can visit a node more than once



K-mers

AGATCCAGCGAGGTCGCTATCCGTTAATTG

5-mers
AGATC
GATCC
ATCCA
…
AATTG

K-mers

AGATCCAGCGAGGTCGCTATCCGTTAATTG

5-mers
AGATC
GATCC
ATCCA
…
AATTG

How many 21-mers are in a 100 bp read?

7-mers
AGATCCA
GATCCAG
ATCCAGC
...
TTAATTG

De Bruijn Graph

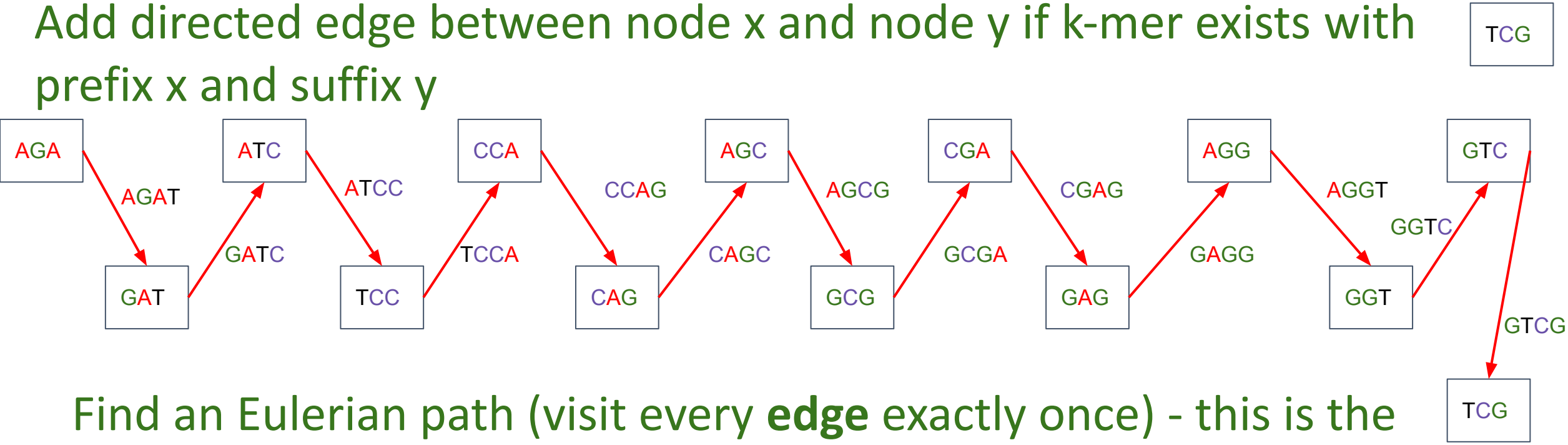
Genome (G=30): AGATCCAGCGAGGTCGCTATCCGTTAATTG

Reads (L=10): AGATCCAGCG
AGCGAGGTCG
GCTATCCGTT
CCGTTAATTG

Break into k-mers (k=4):
AGATCCAGCG → AGAT GATC ATCC TCCA CCAG CAGC AGCG
AGCGAGGTCG → AGCG GCGA CGAG GAGG AGGT GGTC GTCG
…

Create graph nodes - each **unique prefix and suffix** of length k-1 of k-mers

Add directed edge between node x and node y if k-mer exists with prefix x and suffix y



Find an Eulerian path (visit every **edge** exactly once) - this is the assembly!

De Bruijn Graph

A procedure for making a De Bruijn graph for a genome

Assume perfect sequencing where each length-k substring is sequenced exactly once with no errors

Pick a substring length k: 5

a_long_long_long_time

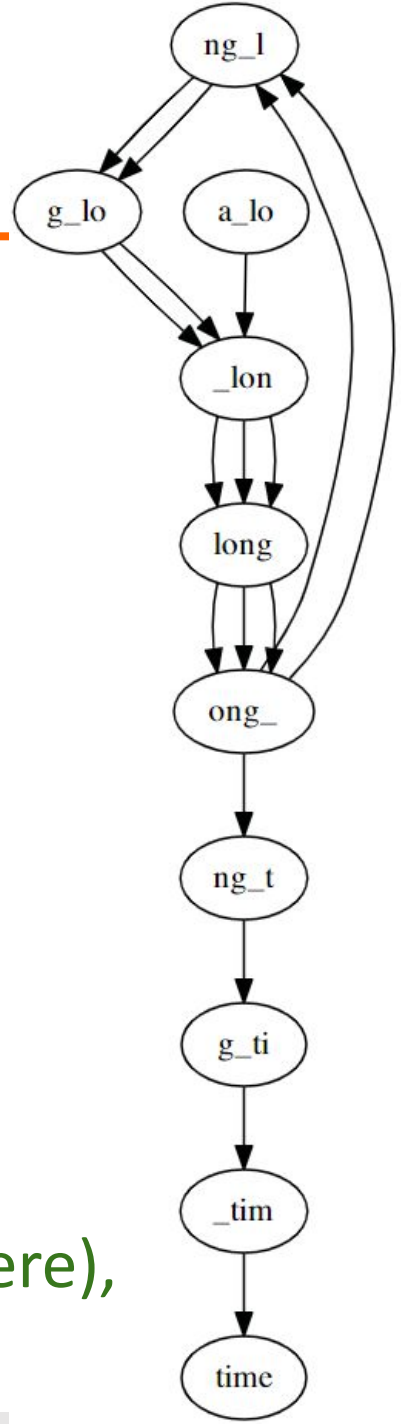
long_

long ong_

Start with each read:

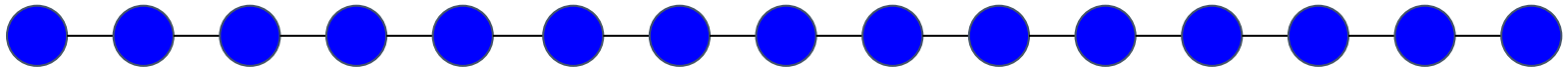
Take each k mer and split into left and right k-1 mers

Add k-1 mers as nodes to De Bruijn graph (if not already there), add edge from left k-1 mer to right k-1 mer

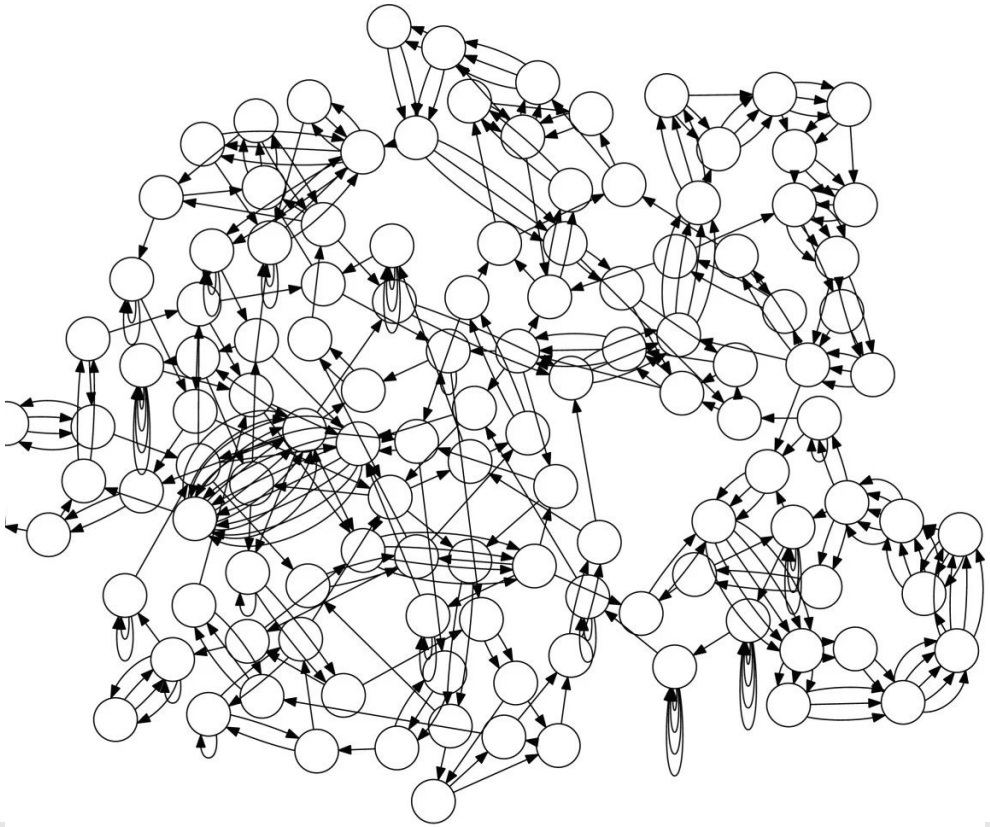


De Bruijn Graph

Ideally, we want our graph to look like this:



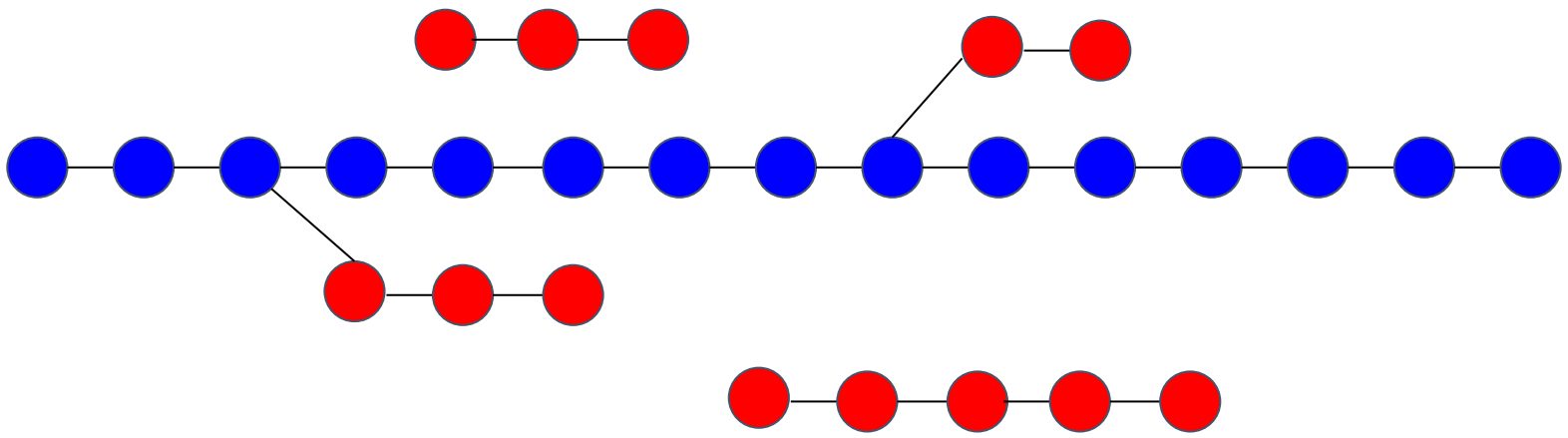
But in practice:



De Bruijn Graph

Sequencing errors

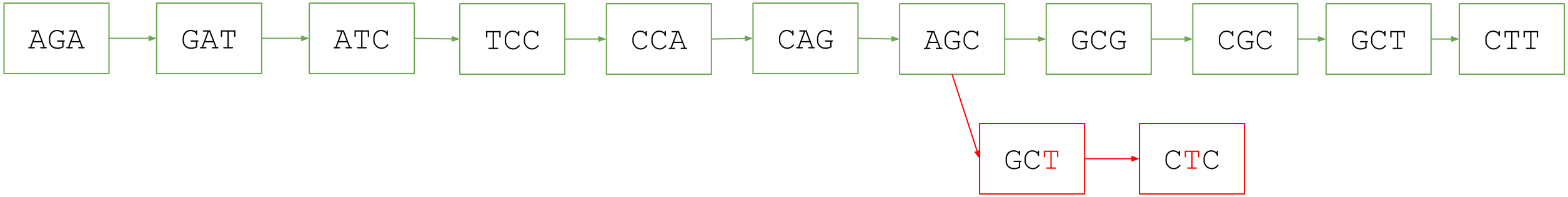
- Side branches
- Disconnected bits



De Bruijn Graph

Sequencing errors

```
AGATCCAGCG      → AGAT GATC ATCC TCCA CCAG CAGC AGCG
 GATCCAGCTC     → GATC ATCC TCCA CCAG CAGC AGCT GCTC
   TCCAGCGCTT   → TCCA CCAG CAGC AGCG GCGC CGCT GCTT
```



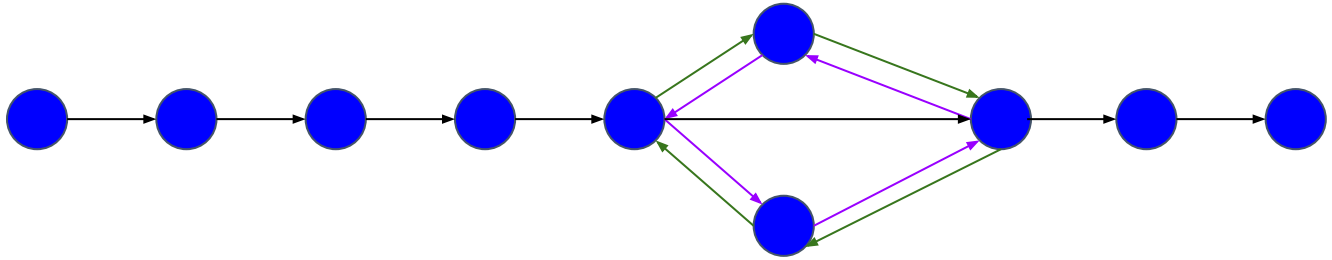
De Bruijn Graph

Genomic Repeats

- Tandem duplication

- Interspersed duplications

- Might create ambiguity - multiple possible eulerian paths

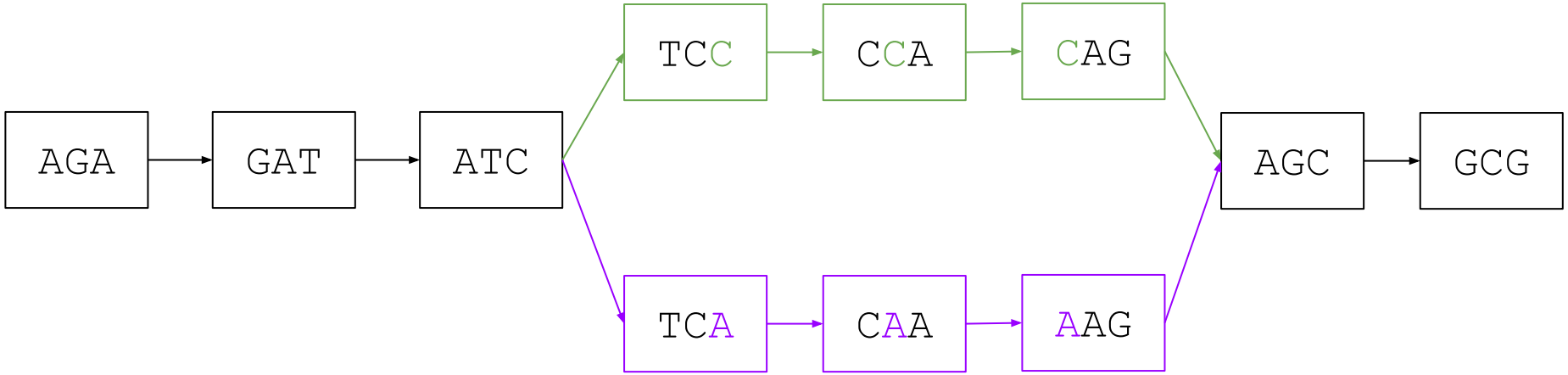


De Bruijn Graph

Heterozygosity

- Creates “bubbles” in the graph

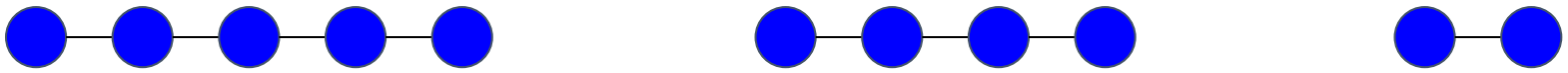
Maternal AGATCCAGCG → AGAT GATC ATCC TCCA CCAG CAGC AGCG
Paternal AGATCAAGCG → AGAT GATC ATCA TCAA CAAG AAGC AGCG



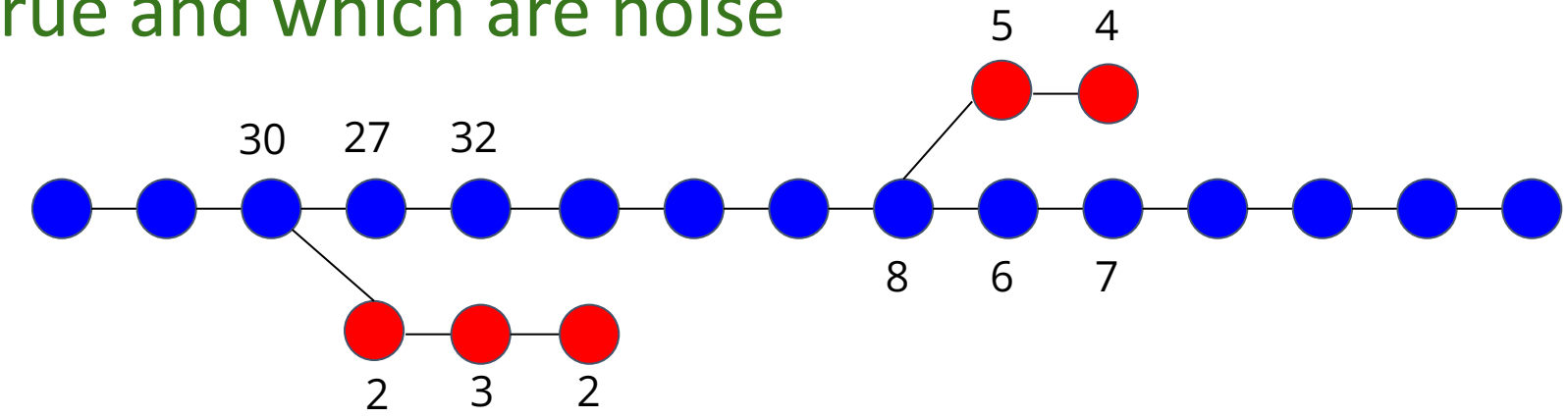
De Bruijn Graph

Uneven sequencing depth

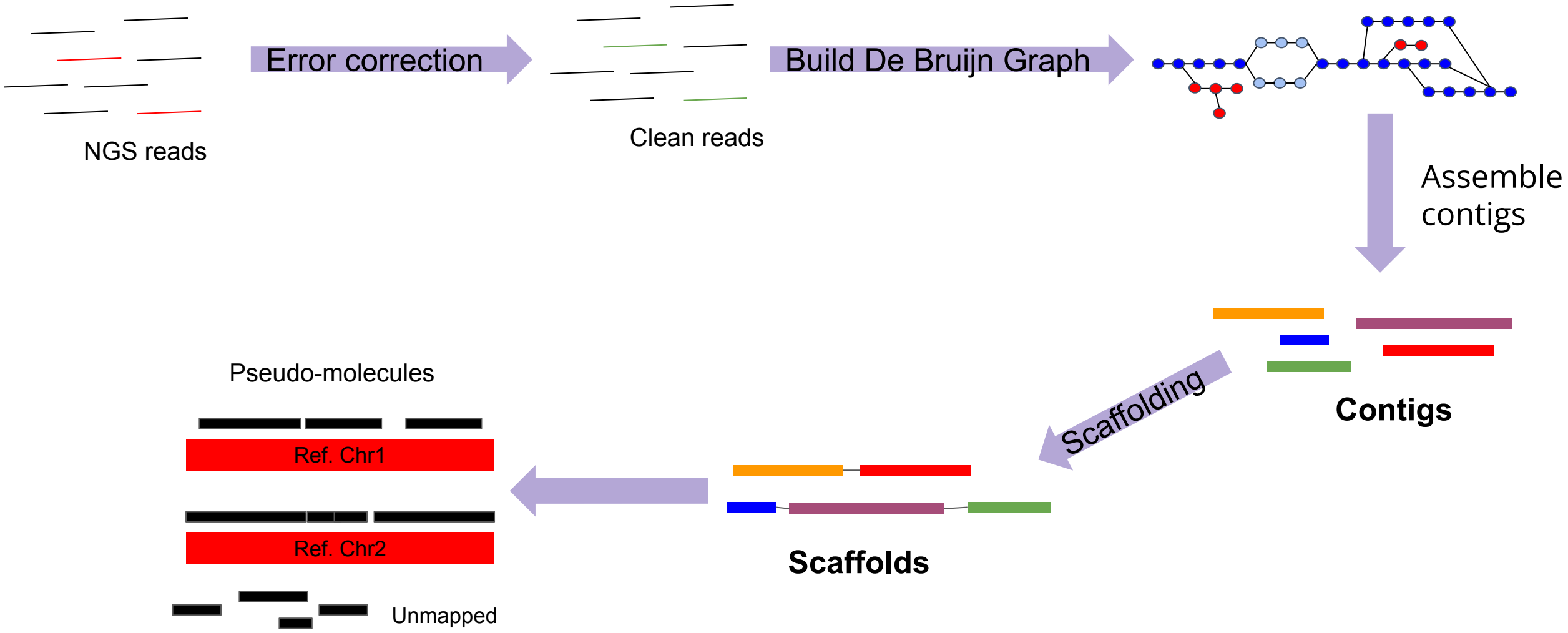
Very low depth (e.g. low complexity regions) can fragment the graph:



Uneven depth can make it hard to determine which branches are true and which are noise



Workflow


NGS reads
Error correction
Clean reads
Build De Bruijn Graph
Assemble contigs
Contigs
Scaffolding
Scaffolds
Pseudo-molecules
Ref. Chr1
Ref. Chr2
Unmapped

Error Correction

Extract all k-mers from all reads

Count how many times each k-mer was observed

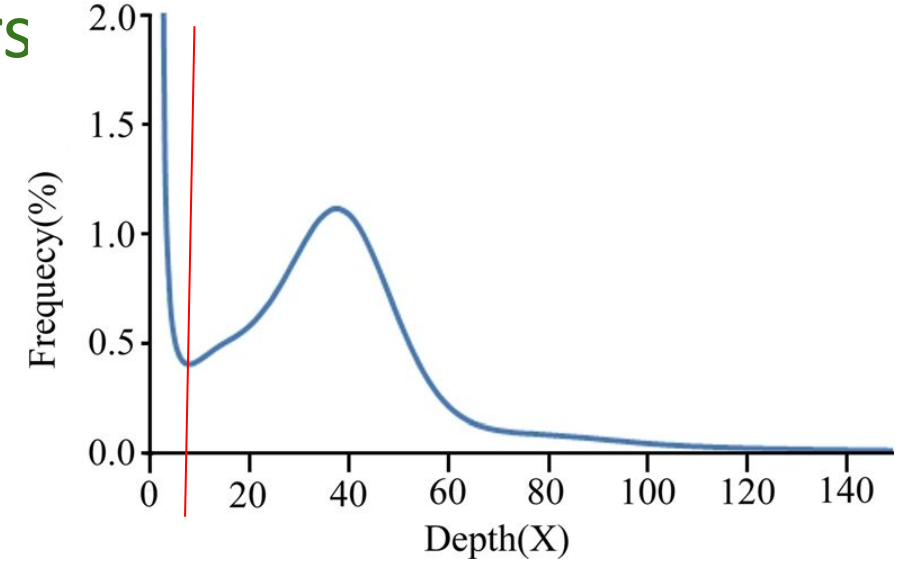
Label rare k-mers as error k-mers

Find reads from which error k-mers came

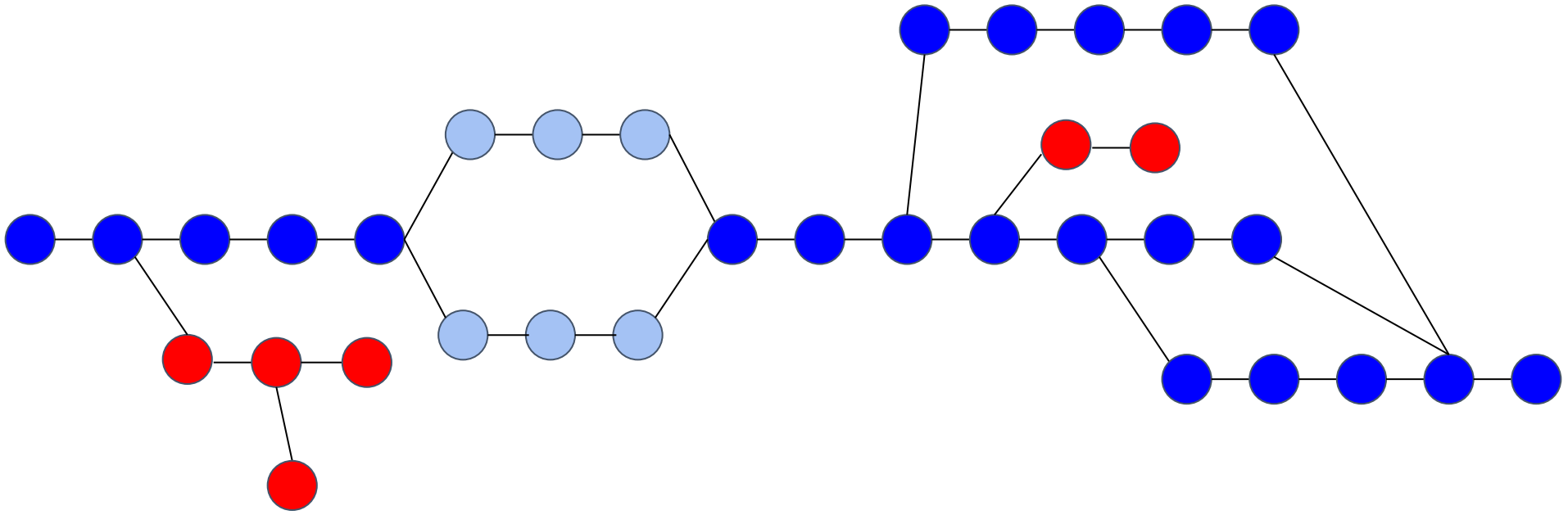
Discard or correct the reads with error k-mers

Count(GGATAGGCACCAGTTAT) = 30

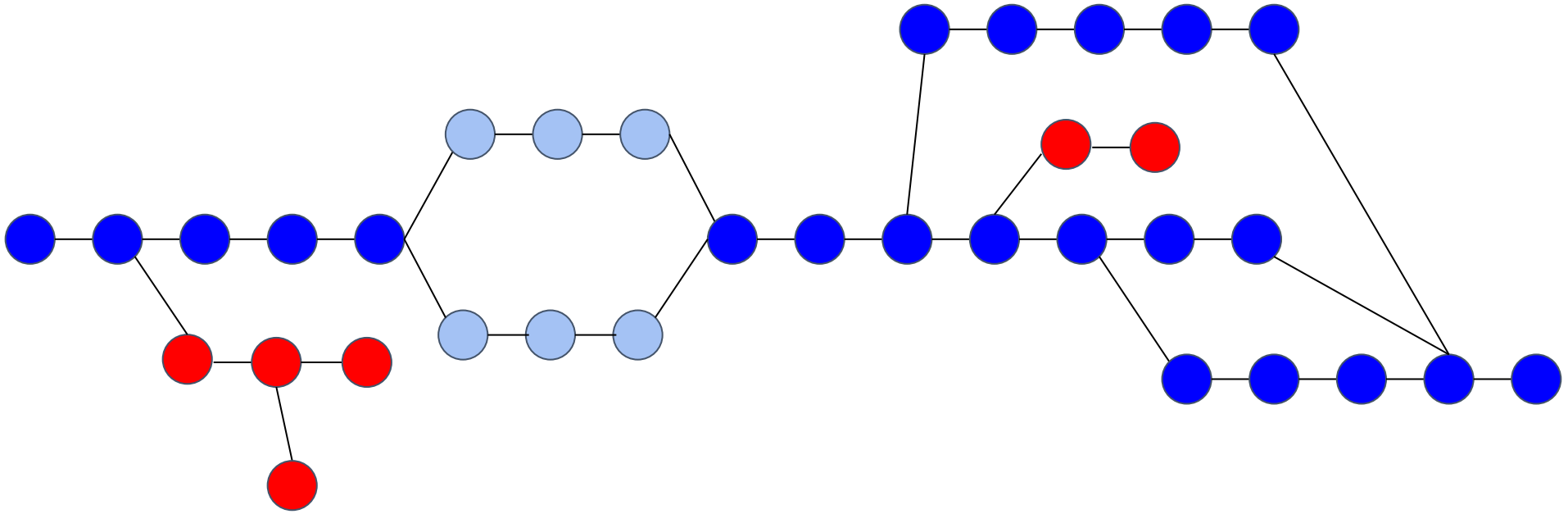
Count(GGATAGGTACCAGTTAT) = 1



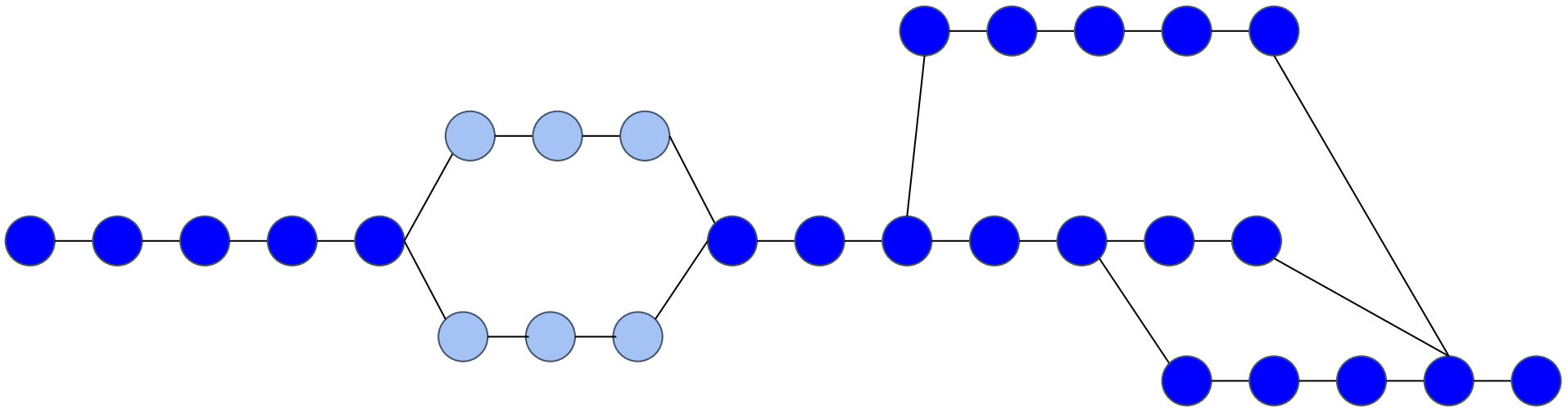
Build De Bruijn Graph



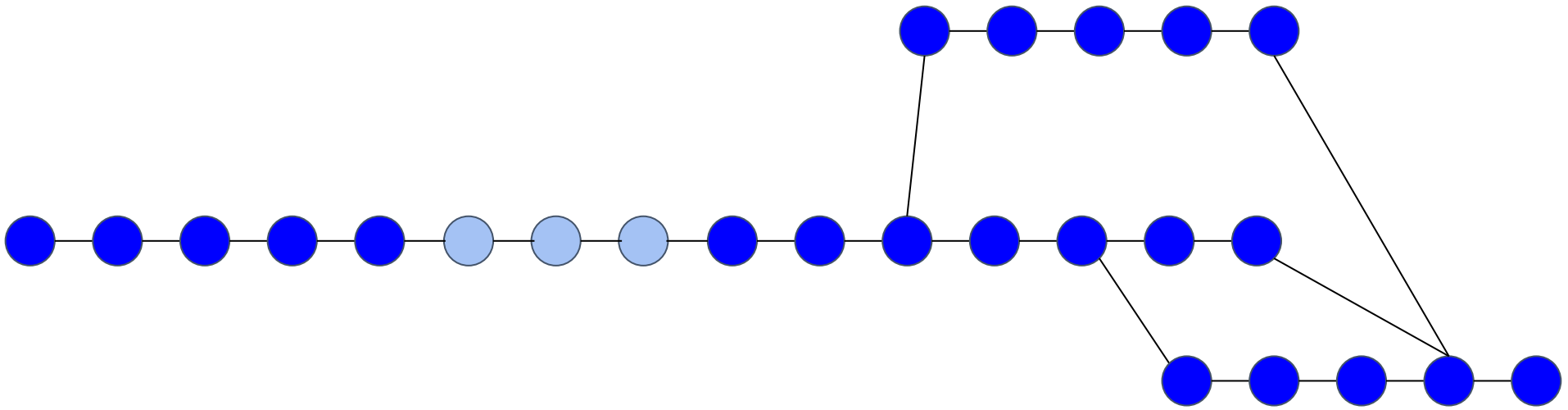
Prune Error Branches



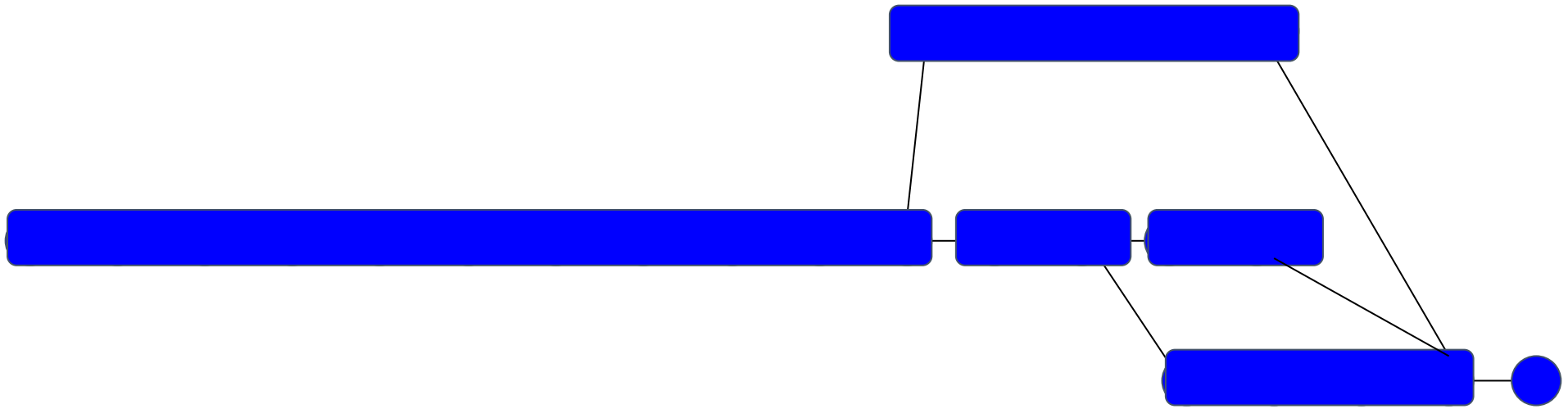
Resolve Bubbles



Resolve Bubbles



Create Contigs

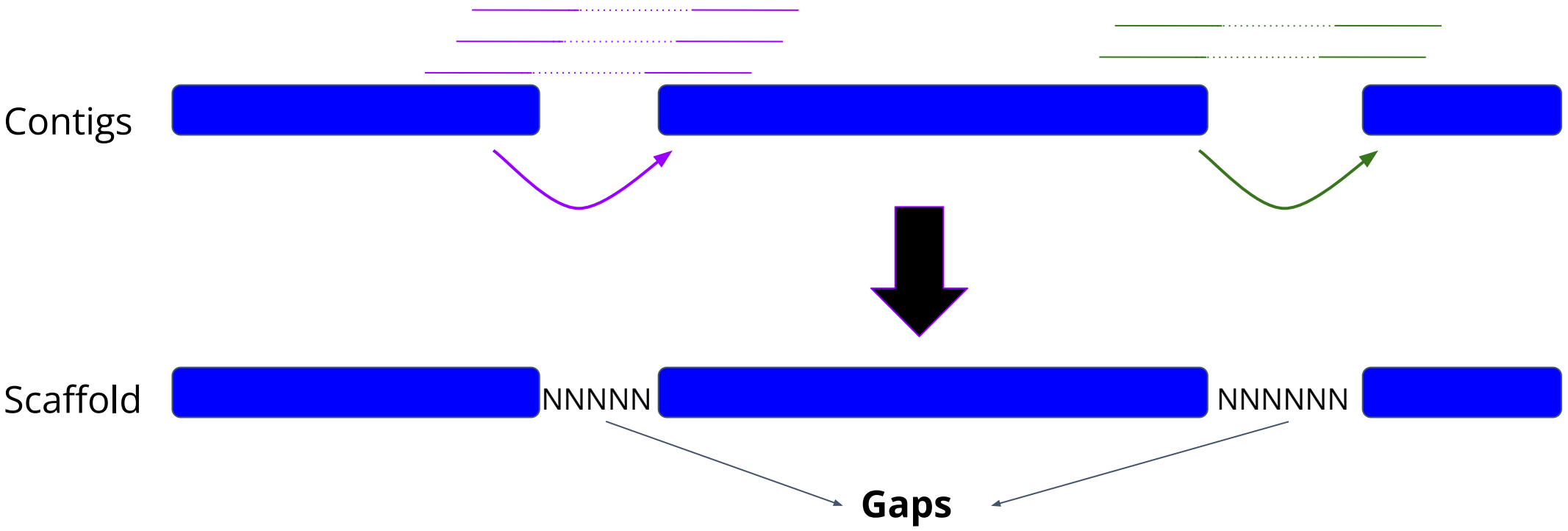


Scaffolding

Mate Pair Sequencing

Use paired end information

Look for evidence of two contigs linked together



De Bruijn Graph

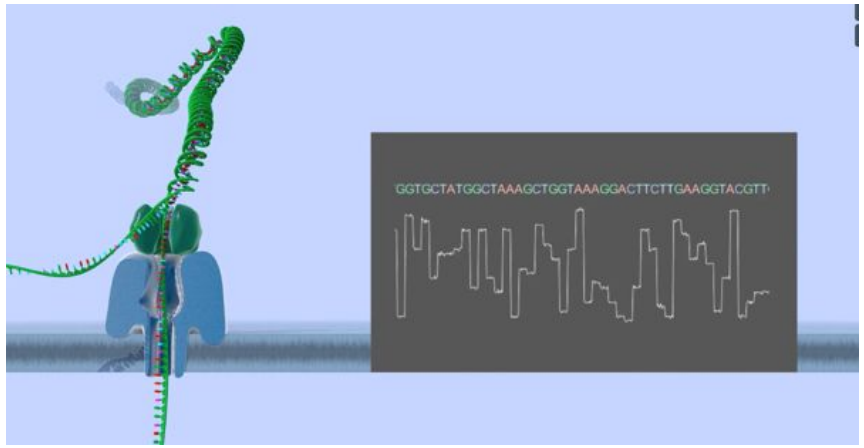
Currently the most popular assembly method

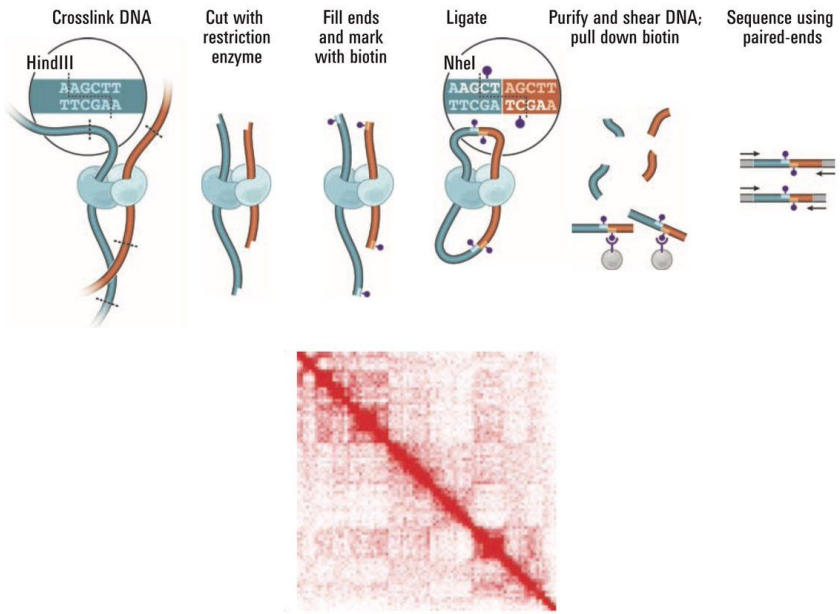
Many software tools use variants of the method

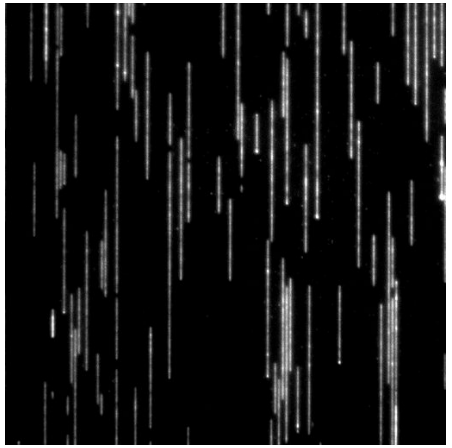
- SPAdes
- SOAPdenovo
- AbySS
- MEGAHIT

SPAdes

Emerging Technologies for Genome Assemblies







Long reads

Hi-C

Optical mapping