DTU

DTU Health Technology
Bioinformatics

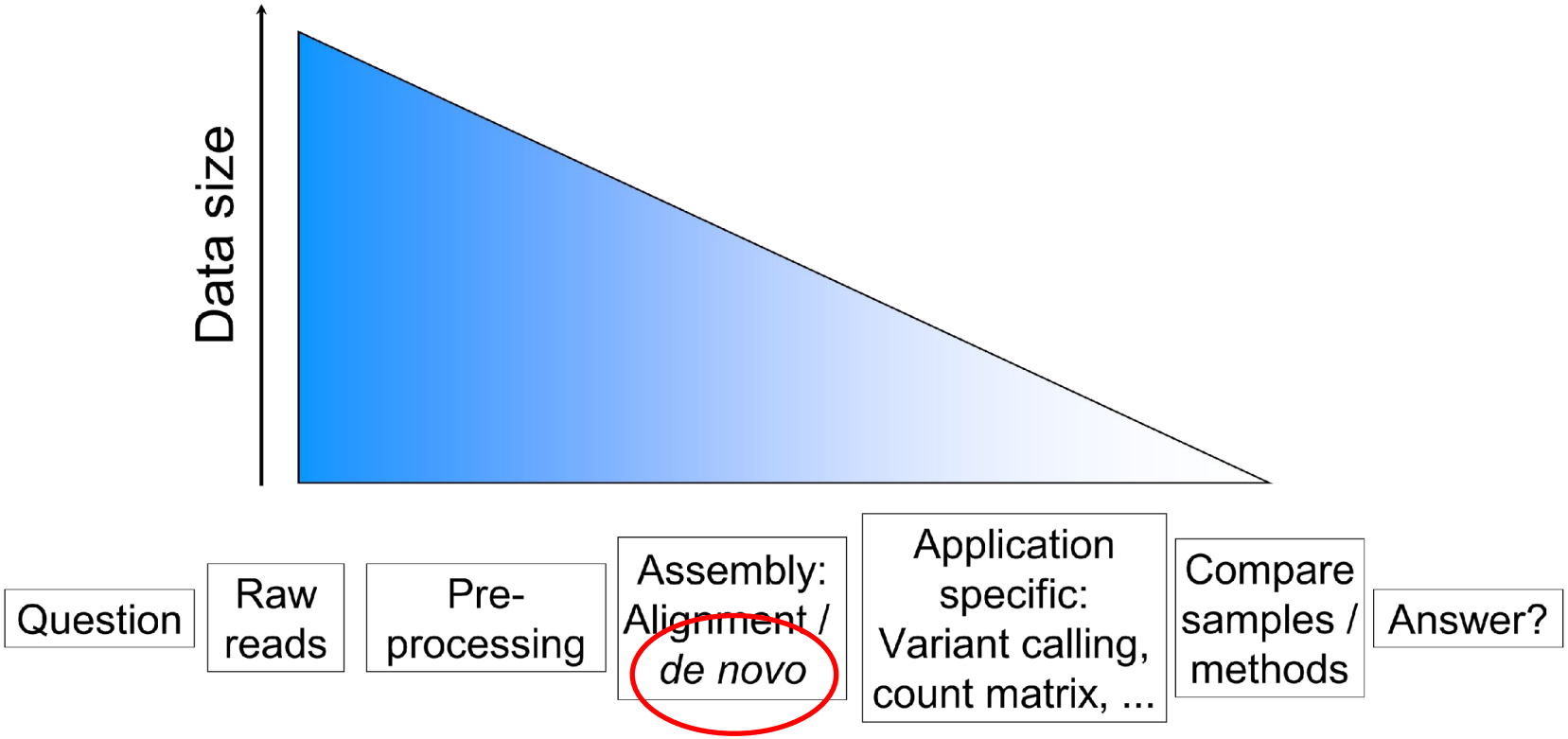
de novo assembly

Gabriel Renaud
Associate Professor
Section of Bioinformatics
Technical University of Denmark
gisves@dtu.dk

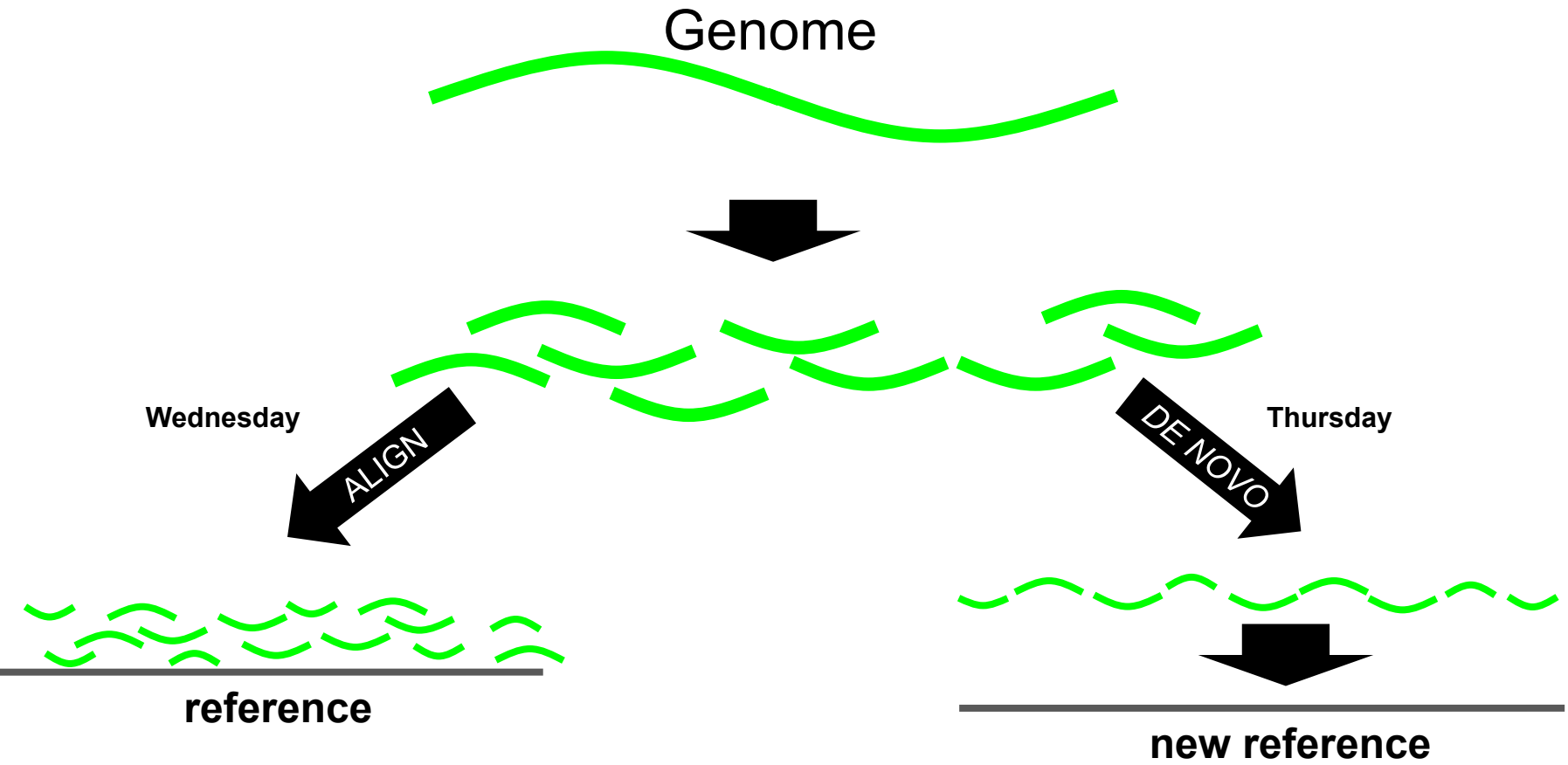
Menu

- Assembly approaches
- Assembly graphs
- Graph postprocessing filtering
- The woes of repetition
- Benchmarking your assembly

Generalized NGS analysis



Whole genome sequencing



Input

Output

Input

```
@MISEQ423_0:+:7218:7278:60-2
GTTACTCGGACTACCCCGATGCATACACCACATGAAACA
T
+
]V]P]]\]]]]]]]\]]]]]]]]]]]]]]]]]]]]]]]]]
]
@MISEQ423_0:-:15245:15305:60-2
AGGGCAAGATGAAGTGAAAGGTAAAGAATCGTGTGAGGG
T
+
]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]
]
@MISEQ423_0:-:242:302:60-2
TTTGGTGGAAATTTTTTGTTATGATGTCTGTGTGGAAAG
T
+
]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]
]
@MISEQ423_0:-:1729:1789:60-2
TGCGGTACTATATCTATTGCGCCAGGTTTCAATTTCTAT
C
+
]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]
```

Output

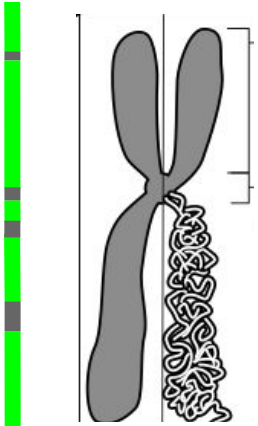
```
>contig#25_0
GATCACAGGTCTATCACCCTATTAACCACTCACGGGAGCTCTCCA
GTATGCACGCGATAGCATTGCGAGACGCTGGAGCCGGAGCACCCT
CTGCCTCATCCTATTATTTATCGCACCTACGTTCAATATTACAGG
ATTAATTAATGCTTGTAGGACATAATAATAACAATTGAATGTCTG
ATAACAAAAAATTTCCACCAAACCCCCCCTCCCCCGCTTCTGGCC
AACCCCAAAAACAAAGAACCCTAACACCAGCCTAACCAGATTTCA
TTTTAACAGTCACCCCCCAACTAACACATTATTTTCCCCTCCCAC
CAACCCCCGCCCATCCTACCCAGCACACACACACCGCTGCTAACC
AAAGACACCCCCCACAGTTTATGTAGCTTACCTCCTCAAAGCAAT
ACATCACCCCATAAACAAATAGGTTTGGTCCTAGCCTTTCTATTA
GCATCCCCGTTCCAGTGAGTTCACCCTCTAAATCACCACGATCAA
AATGCAGCTCAAAACGCTTAGCCTAGCCACACCCCCACGGGAAAC
ACGAAAGTTTAACTAAGCTATACTAACCCCAGGGT
```

Important definitions

Contigs

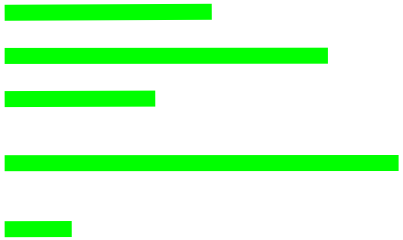
Scaffolds

Chromosome



Important definitions

Contigs



```
>contig#1
GATCACAGGTCTATCACCCTATTAACCACTCACGGGAGCTCTCCA
GTATGCACGCGATAGCATTGCGAGACGCTGGAGCCGGAGCACCCT
CTGCCTCATCCTATTATTTATCGCACCTACGTTCAATATTACAGG
>contig#2
ATTAATTAATGCTTGTAGGACATAATAATAACAATTGAATGTCTG
ATAACAAAAAATTTCCACCAAACCCCCCCTCCCCCGCTTCTGGCC
>contig#3
AACCCCAAAAACAAAGAACCCTAACACCAGCCTAACCAGATTTCA
TTTTAACAGTCACCCCCCAACTAACACATTATTTTCCCCTCCCAC
CAACCCCCGCCCATCCTACCCAGCACACACACACCGCTGCTAACC
AAAGACACCCCCCACAGTTTATGTAGCTTACCTCCTCAAAGCAAT
>contig#4
ACATCACCCCATAAACAAATAGGTTTGGTCCTAGCCTTTCTATTA
GCATCCCCGTTCCAGTGAGTTCACCCTCTAAATCACCACGATCAA
AATGCAGCTCAAAACGCTTAGCCTAGCCACACCCCCACGGGAAAC
ACGAAAGTTTAACTAAGCTATACTAACCCCAGGGT
```

Important definitions

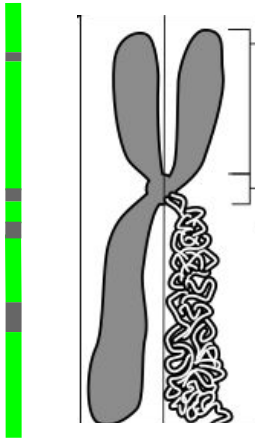
```
>scaffold#1
AACCCCAAAAACAAAGAACCCTAACACCAGCCTAACCAGATTTCA
TTTTAACAGTCACCCCCCAACTAACACATTATTTTCCCCTCCCAC
CAACCCCCGCCCATCCTACCCAGCACACACACACCGCTGCTAACC
AAAGACACCCCCCACAGTTTATGTAGCTTACCTCCTCAAAGCAAT
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNGATCACAGGTCTATC
ACCCTATTAACCACTCACGGGAGCTCTCCA
>scaffold#2
GTATGCACGCGATAGCATTGCGAGACGCTGGAGCCGGAGCACCCT
CTGCCTCATCCTATTATTTATCGCACCTACGTTCAATATTACAGG
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNATTAATTAATGCT
GTAGGACATAATAATAACAATTGAATGTCTGATAACAAAAAATTC
CACCAAACCCCCCCTCCCCCGCTTCTGGCCNNNNNNNNACATCACC
CATAAACAAATAGGTTTGGTCCTAGCCTTTCTATTAGCATCCCCT
TCCAGTGAGTTCACCCTCTAAATCACCACGATCAAAATGCAGCTA
AAACGCTTAGCCTAGCCACACCCCCACGGGAAACACGAAAGTTTA
ACTAAGCTATACTAACCCCAGGGT
```

Scaffolds

Important definitions

```
>chr22
GTATGCACGCGATAGCATTGCGAGACGCTGGAGCCGGAGCACCCT
T
CTGCCTCATCCTATTATTTATCGCACCTACGTTCAATATTACAGG
G
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNATTAATTAATGCT
TGTAGGACATAATAATAACAATTGAATGTCTGATAACAAAAAATT
TCCACCAAACCCCCCCTCCCCCGCTTCTGGCCNNNNNNNACATCA
CCCCATAAACAAATAGGTTTGGTCCTAGCCTTTCTATTAGCATCC
CCGTTCCAGTGAGTTCACCCTCTAAATCACCACGATCAAAATGCA
GCTCAAAACGCTTAGCCTAGCCACACCCCCACGGGAAACACGAAA
GTTTAACTAAGCTATACTAACCCCAGGGTNNNNNNNAACCCCAAA
AACAAAGAACCCTAACACCAGCCTAACCAGATTTCATTTTAACAG
TCACCCCCCAACTAACACATTATTTTCCCCTCCCACCAACCCCCG
CCCATCCTACCCAGCACACACACACCGCTGCTAACCAAAGACACC
CCCCACAGTTTATGTAGCTTACCTCCTCAAAGCAATNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNGATCACAGGTCTATCACCCTATTA
ACCACTCACGGGAGCTCTCCA
```

Chromosome

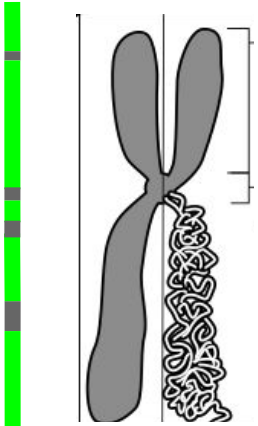


Important definitions

Contigs

Scaffolds

Chromosome



Which approaches?

- Overlap-Layout-Consensus (OLC)
- de Bruijn graphs

Overlap-Layout-Consensus

- Create overlap graph by all-vs-all alignment (Overlap)
- Build graph where each node is a read, edges are overlaps between reads (Layout)

reads:

TCTCAACG

CGATTGTC

ATTGTCTC

CTCAACGT

TGTCTCAA

TTGTCTCA

Overlap-Layout-Consensus

- Create overlap graph by all-vs-all alignment (Overlap)
- Build graph where each node is a read, edges are overlaps between reads (Layout)

overlap of at least 4bp in **bold**

reads:

TCTCAACG

CG**ATTGTC**

ATTGTCTC

CTCAACGT

TGTCTCAA

TTGTCTCA

Overlap-Layout-Consensus

- Create overlap graph by all-vs-all alignment (Overlap)
- Build graph where each node is a read, edges are overlaps between reads (Layout)

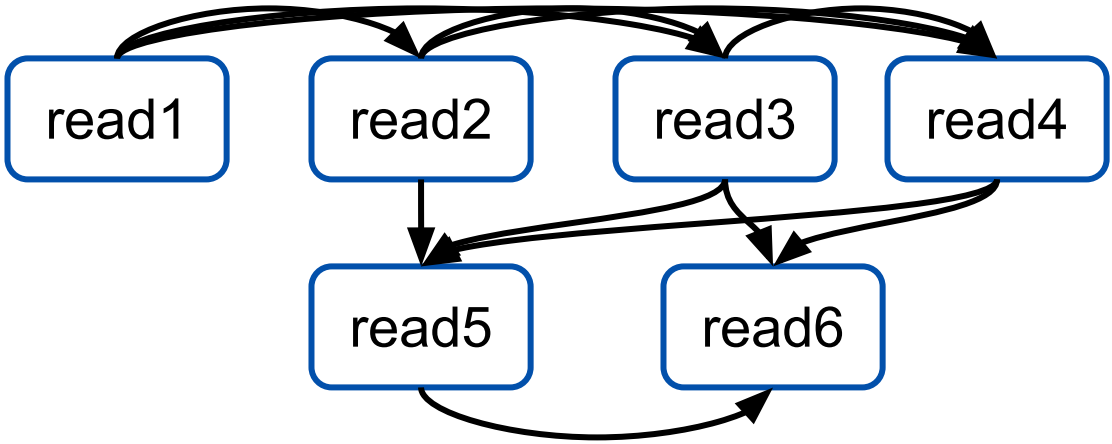
reads:

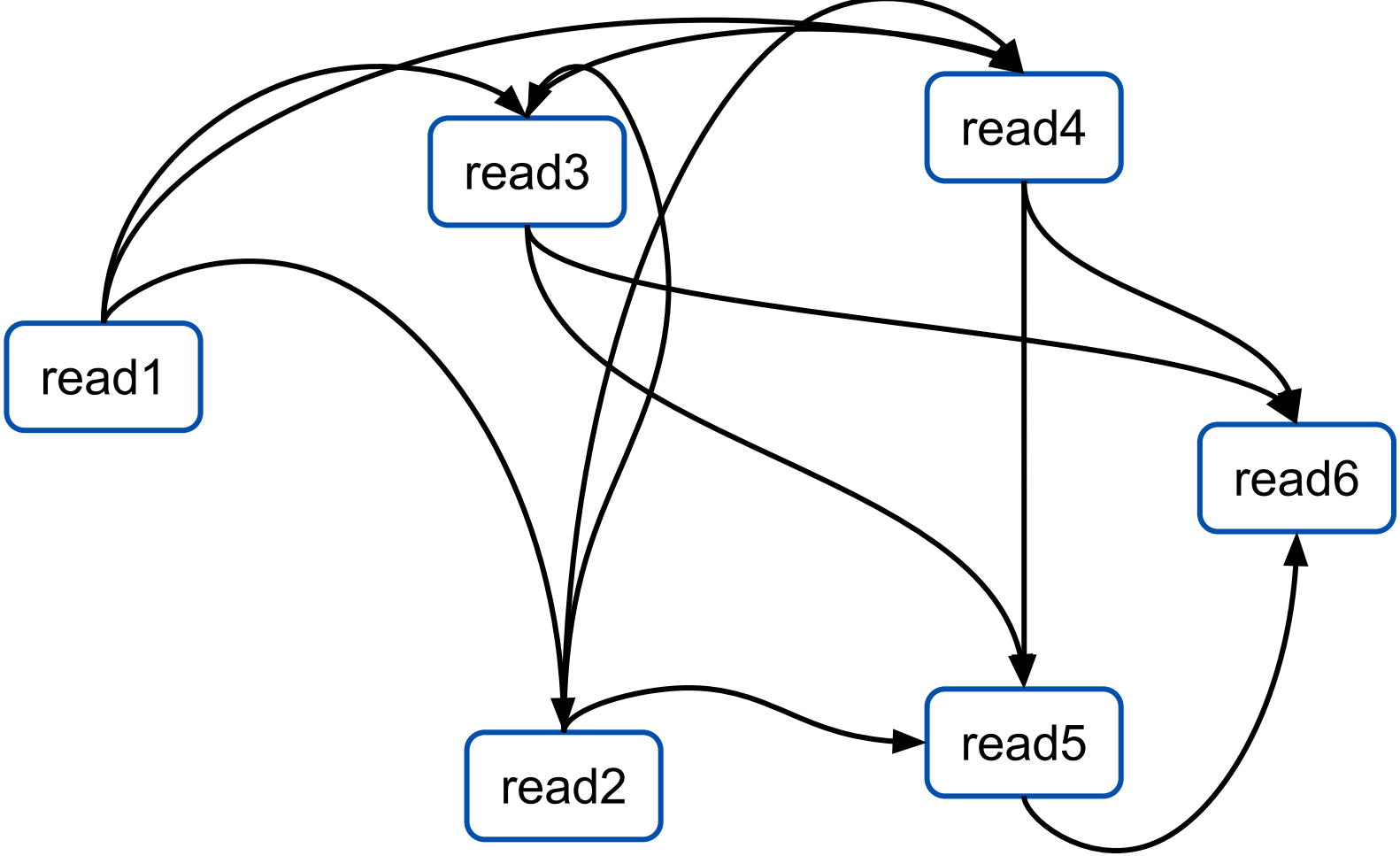
```
read1   CGATTGTC
read2     ATTGTCTC
read3      TTGTCTCA
read4       TGTCTCAA
read5         TCTCAACG
read6          CTCAACGT
```

Overlap-Layout-Consensus

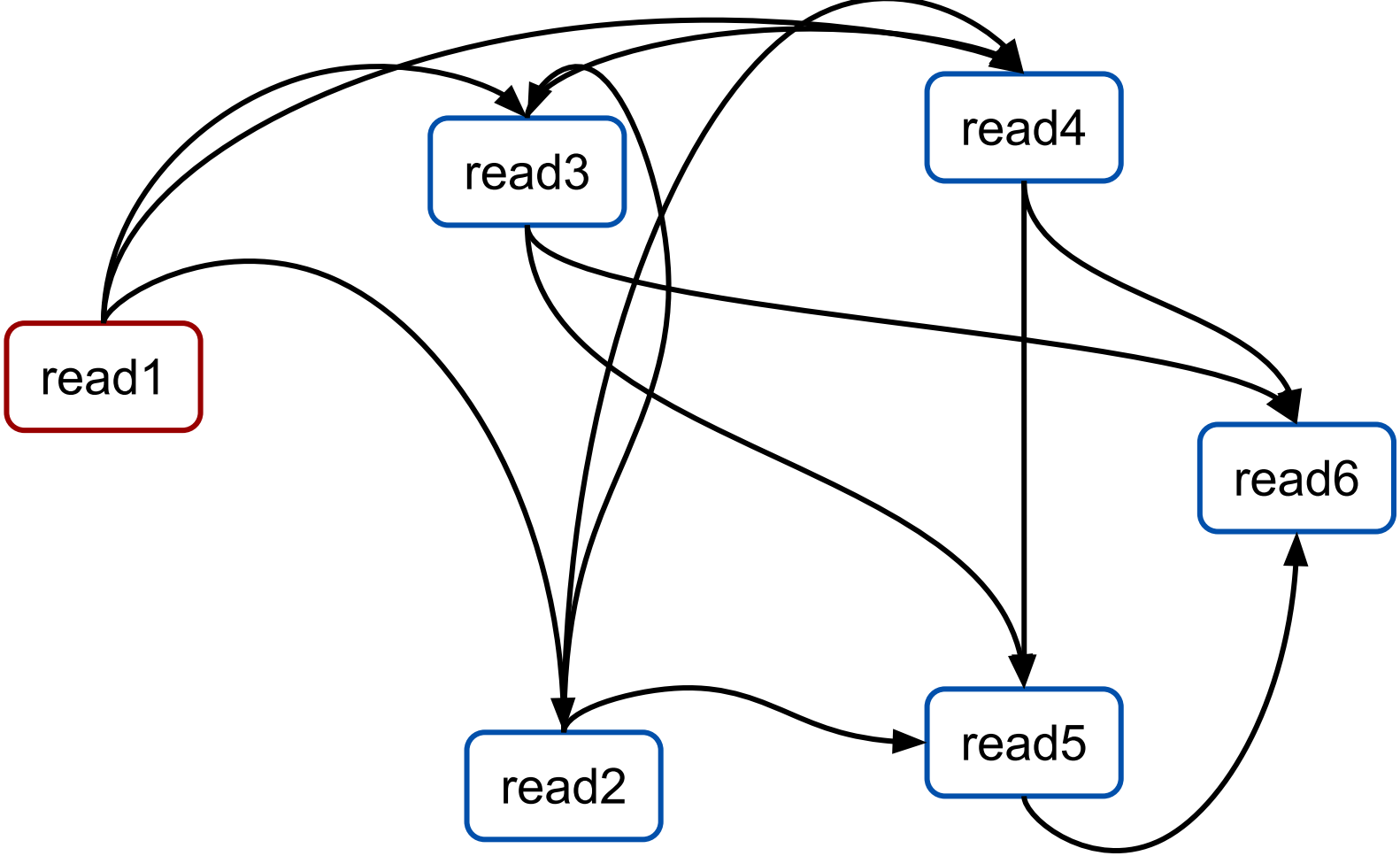
- Create overlap graph by all-vs-all alignment (Overlap)
- Build graph where each node is a read, edges are overlaps between reads (Layout)

```
read1  CGATTGTC
read2    ATTGTCTC
read3     TTGTCTCA
read4      TGTCTCAA
read5        TCTCAACG
read6         CTCAACGT
```

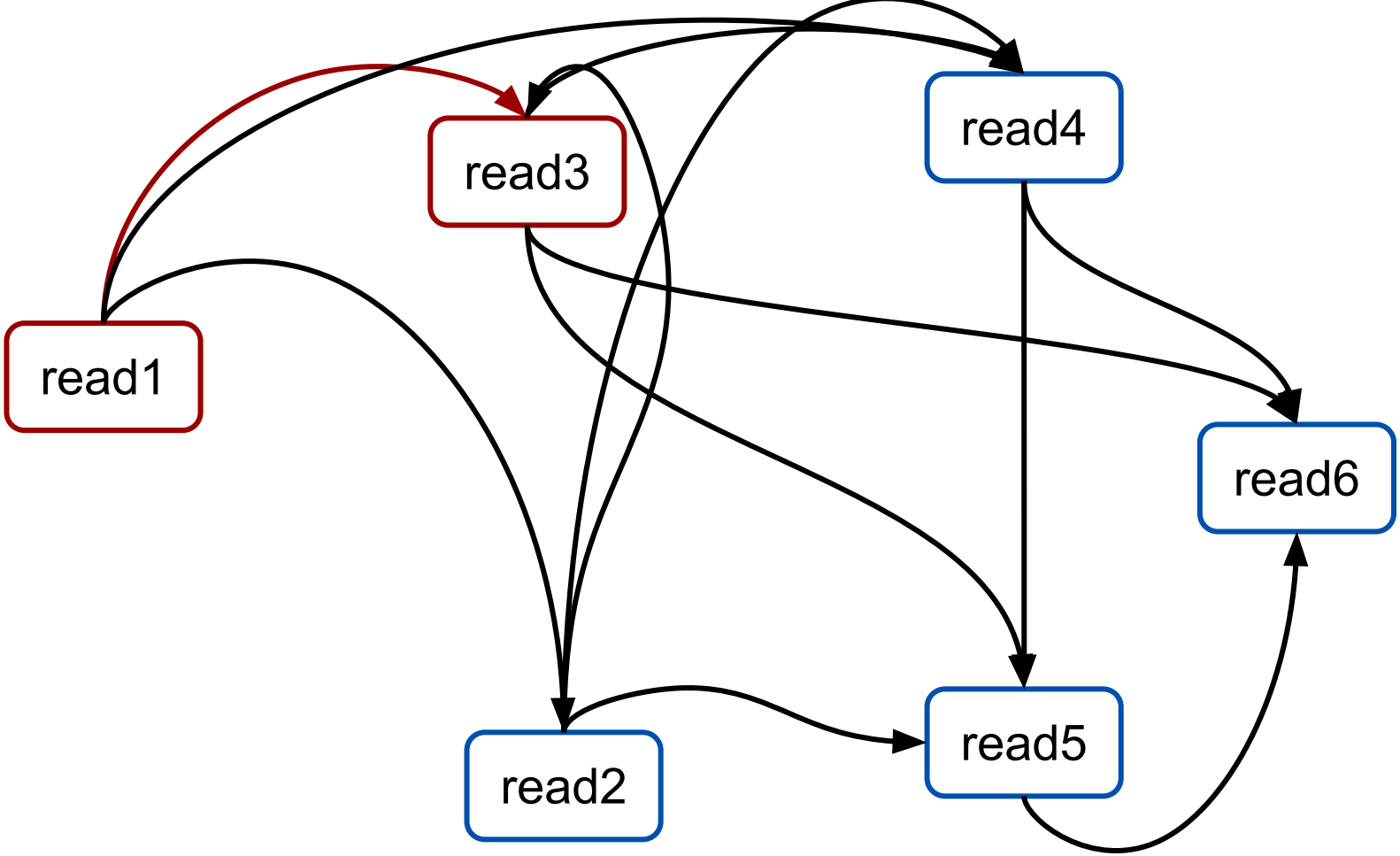




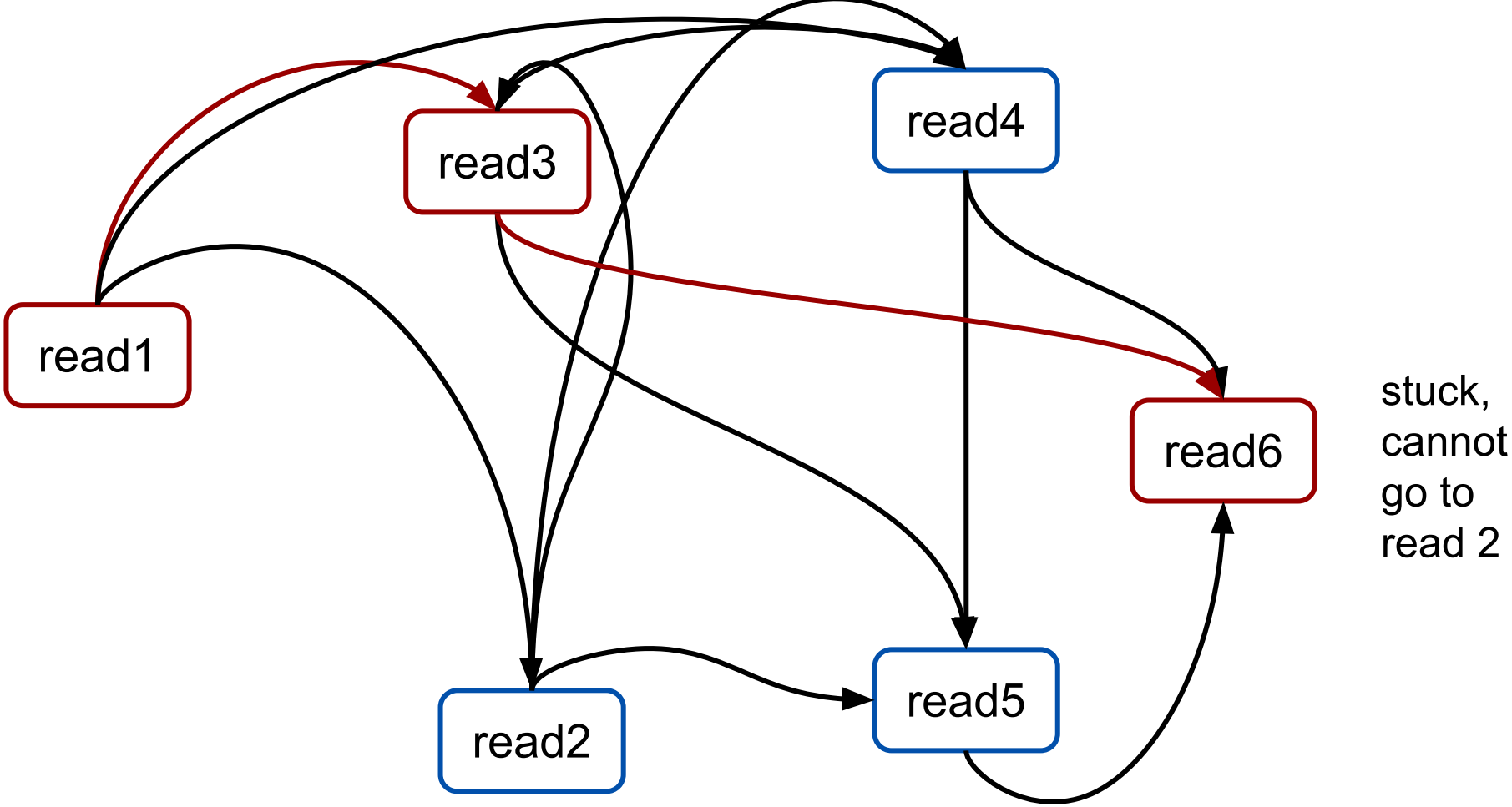
Take a path that goes through each read once



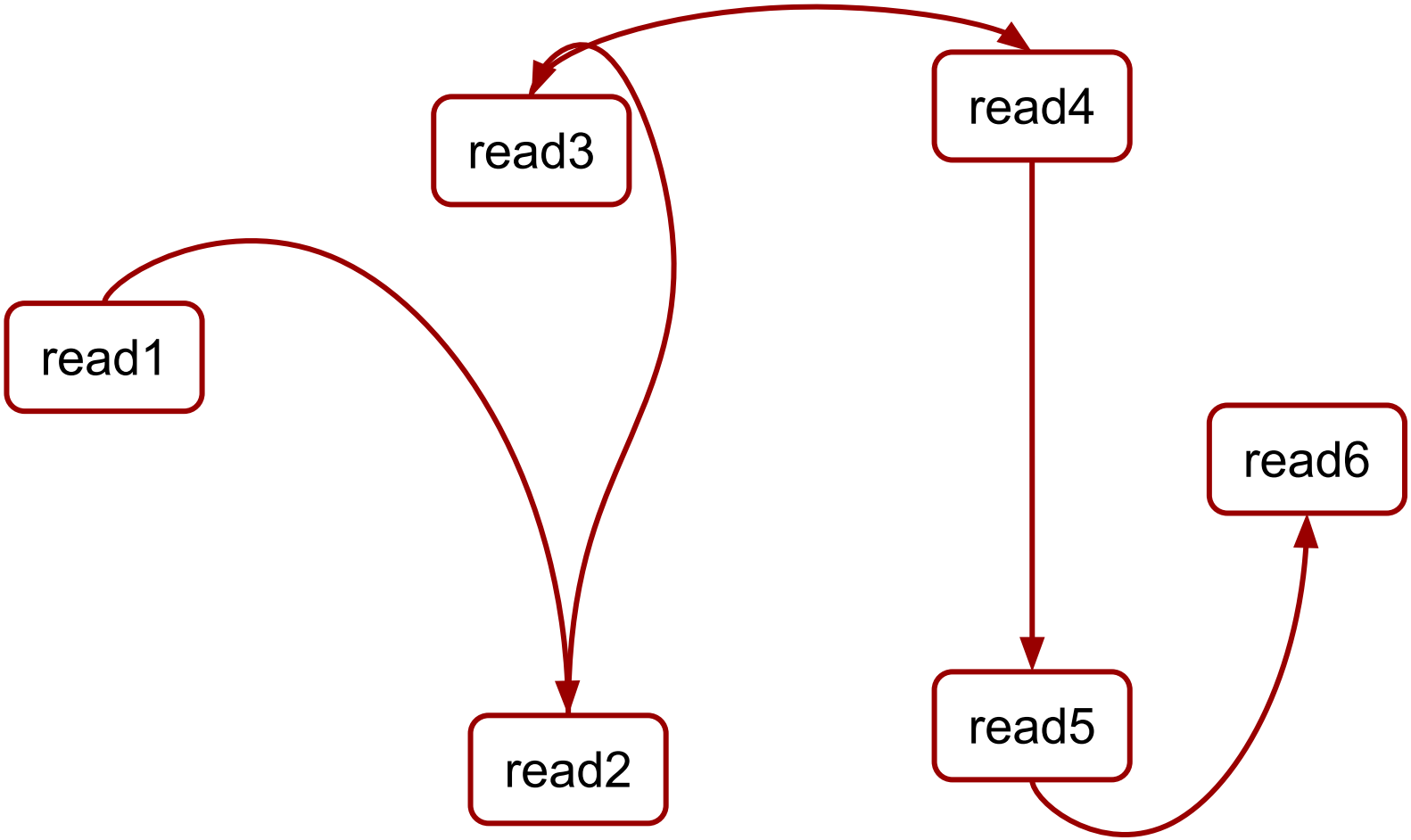
Take a path that goes through each read once



Take a path that goes through each read once



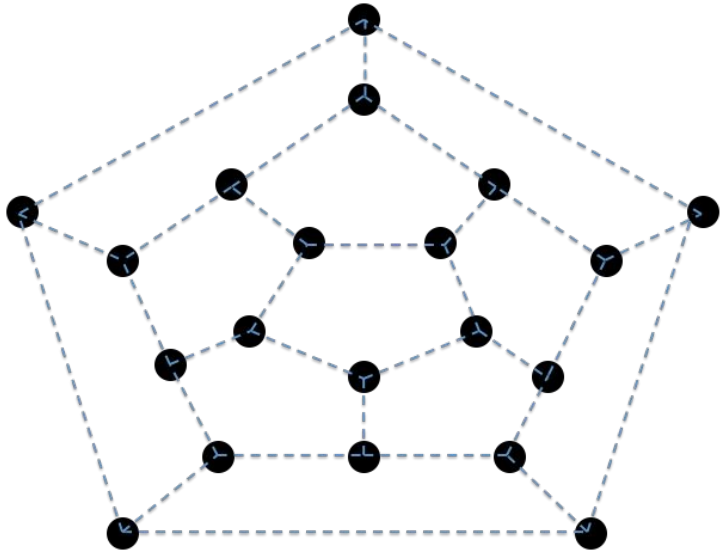
Take a path that goes through each read once



Take a path that goes through each read once

Overlap-Layout-Consensus

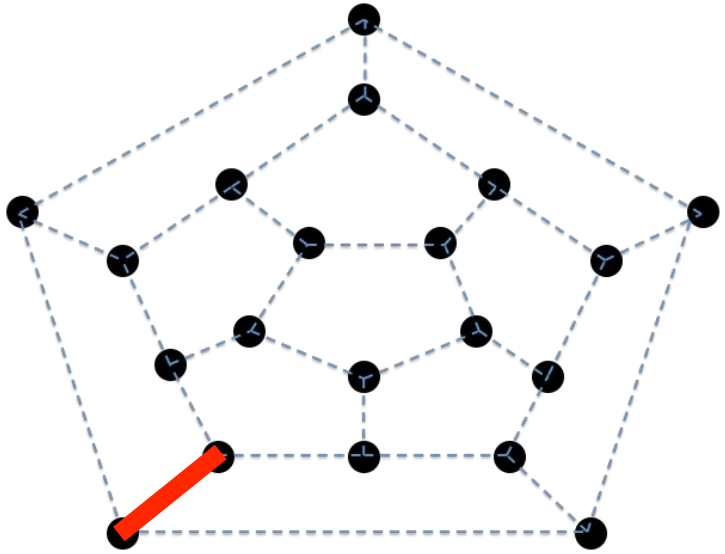
- Create consensus sequence
- We need to use **graph theory** to solve the graph
- Find the *Hamiltonian path*
- i.e. visit each node *exactly once*
- NP-complete



Imagine trying to solve this for a graph of hundred of thousands of nodes (=reads)

Overlap-Layout-Consensus

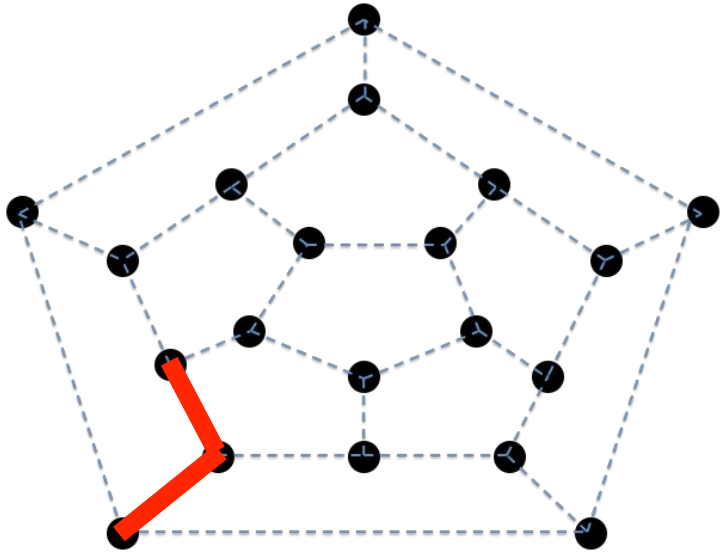
- Create consensus sequence
- We need to use **graph theory** to solve the graph
- Find the *Hamiltonian path*
- i.e. visit each node *exactly once*
- NP-complete



Imagine trying to solve this for a graph of hundred of thousands of nodes (=reads)

Overlap-Layout-Consensus

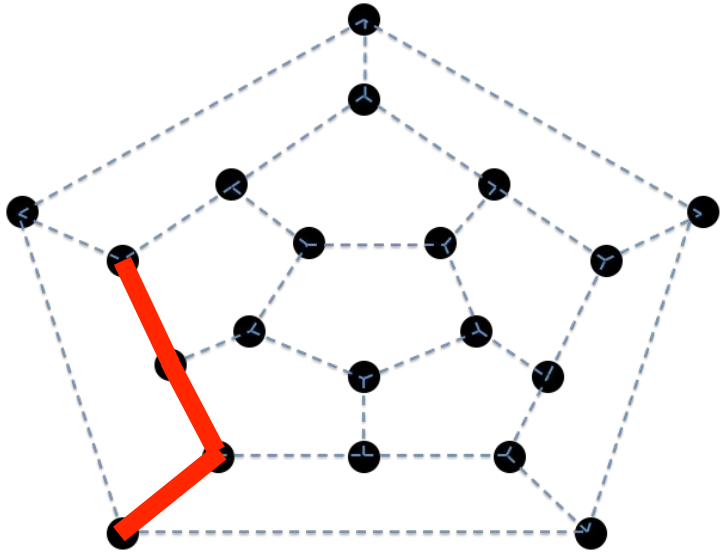
- Create consensus sequence
- We need to use **graph theory** to solve the graph
- Find the *Hamiltonian path*
- i.e. visit each node *exactly once*
- NP-complete



Imagine trying to solve this for a graph of hundred of thousands of nodes (=reads)

Overlap-Layout-Consensus

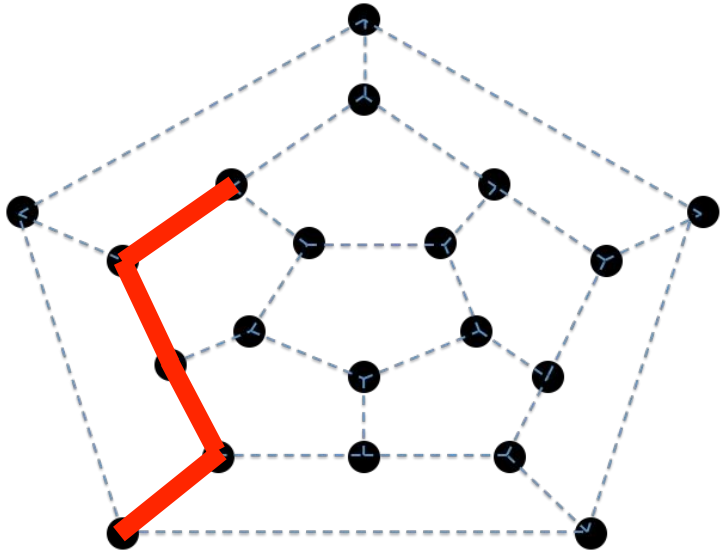
- Create consensus sequence
- We need to use **graph theory** to solve the graph
- Find the *Hamiltonian path*
- i.e. visit each node *exactly once*
- NP-complete



Imagine trying to solve this for a graph of hundred of thousands of nodes (=reads)

Overlap-Layout-Consensus

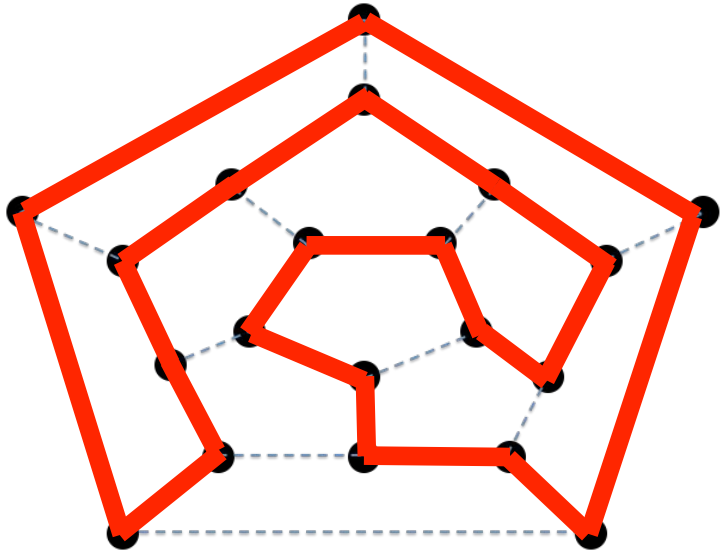
- Create consensus sequence
- We need to use **graph theory** to solve the graph
- Find the *Hamiltonian path*
- i.e. visit each node *exactly once*
- NP-complete



Imagine trying to solve this for a graph of hundred of thousands of nodes (=reads)

Overlap-Layout-Consensus

- Create consensus sequence
- We need to use **graph theory** to solve the graph
- Find the *Hamiltonian path*
- i.e. visit each node *exactly once*
- NP-complete



Imagine trying to solve this for a graph of hundred of thousands of nodes (=reads)

Overlap-Layout-Consensus

- Create overlap graph by all-vs-all alignment (Overlap)
- Build graph where each node is a read, edges are overlaps between reads (Layout)
- Call a consensus

reads:

```
CGATTGTC
  ATTGTCTC
   TTGTCTCA
    TGTCTCAA
      TCTCAACG
       CTCAACGT
```

consensus: CGATTGTCTCAACGT

Overlap-Layout-Consensus

- Not good with many short reads O(n^2) -> lots of alignment!
- With short read lengths, hard to resolve repeats

- Good for large read lengths:
  – PacBio, Oxford Nanopore, 10X Genomics, 454, Ion Torrent, Sanger

- Example assemblers: Canu, Celera, Newbler, PenguiN, CarpeDeam

de Bruijn graph

- Directed graph of overlapping items (here DNA sequences)
- Instead of comparing reads, decompose reads into *k*-mers
  - Graph is created by mapping the *k*-mers to the graph
  - Each *k*-mer only exists once in the graph
  - Problem is reduced to walking Eulerian path (visiting each edge once) - this is a solve-able problem

Drawbacks ...

- Lots of RAM required (**1-1000 GB !**)
- Optimal *k* can not be identified *a priori*, must be experimentally tested for each dataset
- small *k*: very complex graph, large *k*: limited overlap in low coverage areas
- Iterative approach to find best assembly

How is the graph constructed?

- Same 10 reads, extract *k*-mers from reads and map onto graph, *k* = 3:

reads:

TCTCAACG

CGATTGTC

ATTGTCTC

CTCAACGT

TGTCTCAA

TTGTCTCA

How is the graph constructed?

- Same 10 reads, extract *k*-mers from reads and map onto graph, *k* = 3:

reads:

```
TCTCAACG
CGATTGTC
TGTCTCAA
CTCAACGT
ATTGTCTC
TTGTCTCA
```

How is the graph constructed?

- Same 10 reads, extract *k*-mers from reads and map onto graph, *k* = 3:

reads:

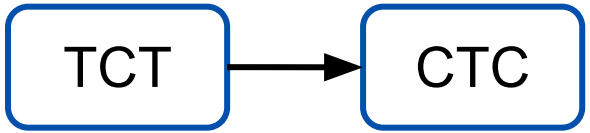
TCTCAACG
CGATTGTC
TGTCTCAA
CTCAACGT
ATTGTCTC
TTGTCTCA

How is the graph constructed?

- Same 10 reads, extract *k*-mers from reads and map onto graph, *k* = 3:

reads:

T**CTC**AACG
CGATTGTC
TGTCTCAA
CTCAACGT
ATTGTCTC
TTGTCTCA



How is the graph constructed?

- Same 10 reads, extract *k*-mers from reads and map onto graph, *k* = 3:

reads:

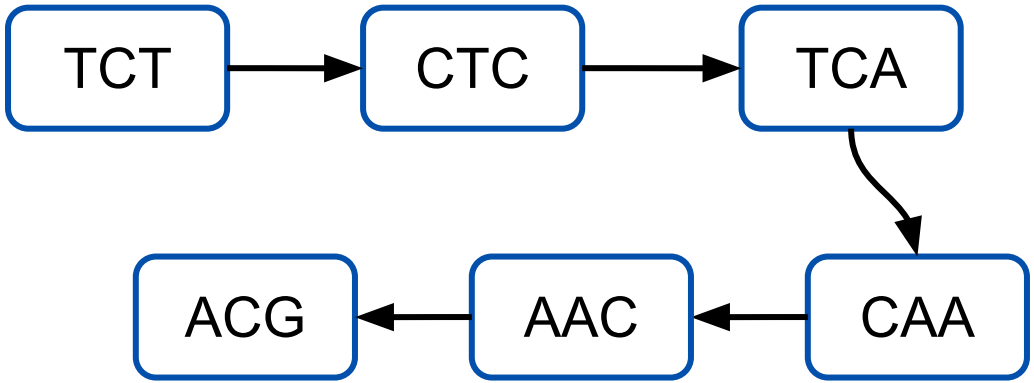
TC**TCA**ACG
CGATTGTC
TGTCTCAA
CTCAACGT
ATTGTCTC
TTGTCTCA

How is the graph constructed?

- Same 10 reads, extract *k*-mers from reads and map onto graph, *k* = 3:

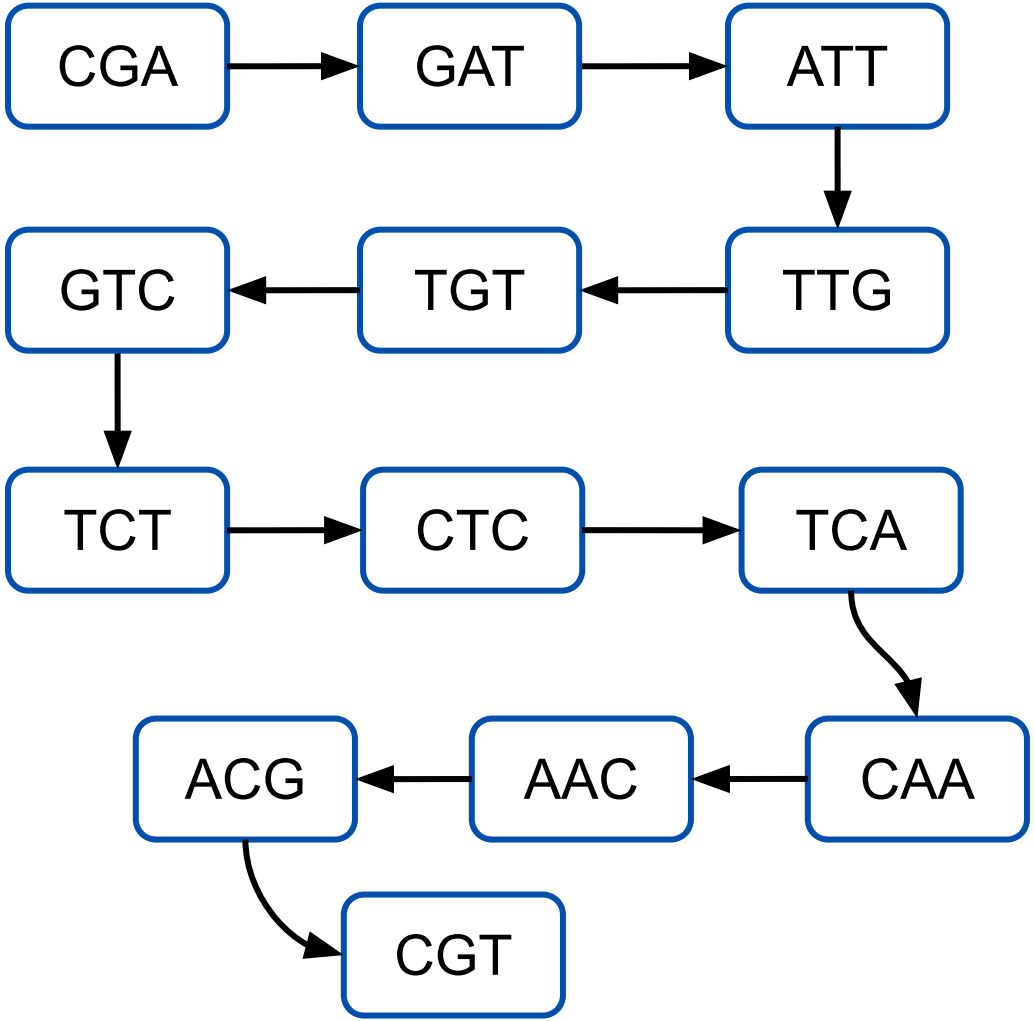
reads:

TCTCAACG
CGATTGTC
TGTCTCAA
CTCAACGT
ATTGTCTC
TTGTCTCA

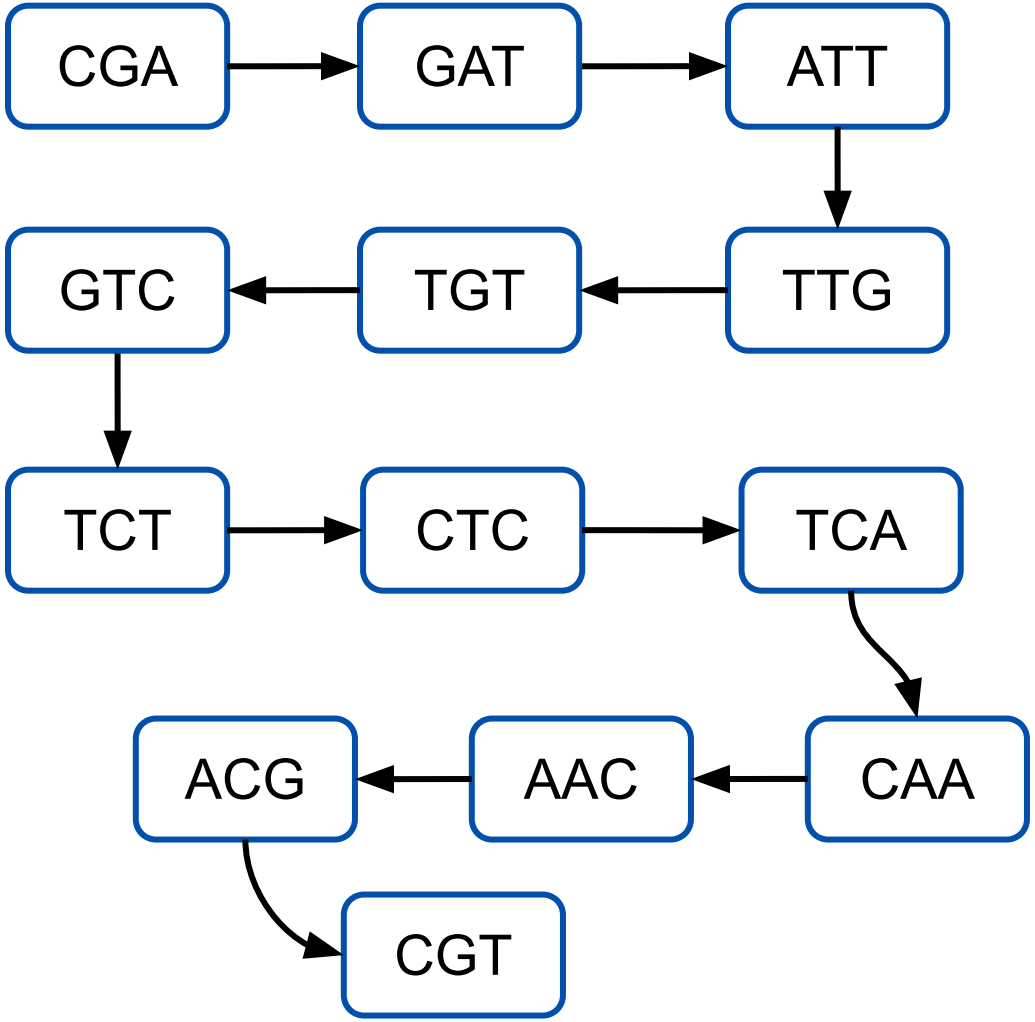


reads:

TCTCAACG
CGATTGTC
TGTCTCAA
CTCAACGT
ATTGTCTC
TTGTCTCA



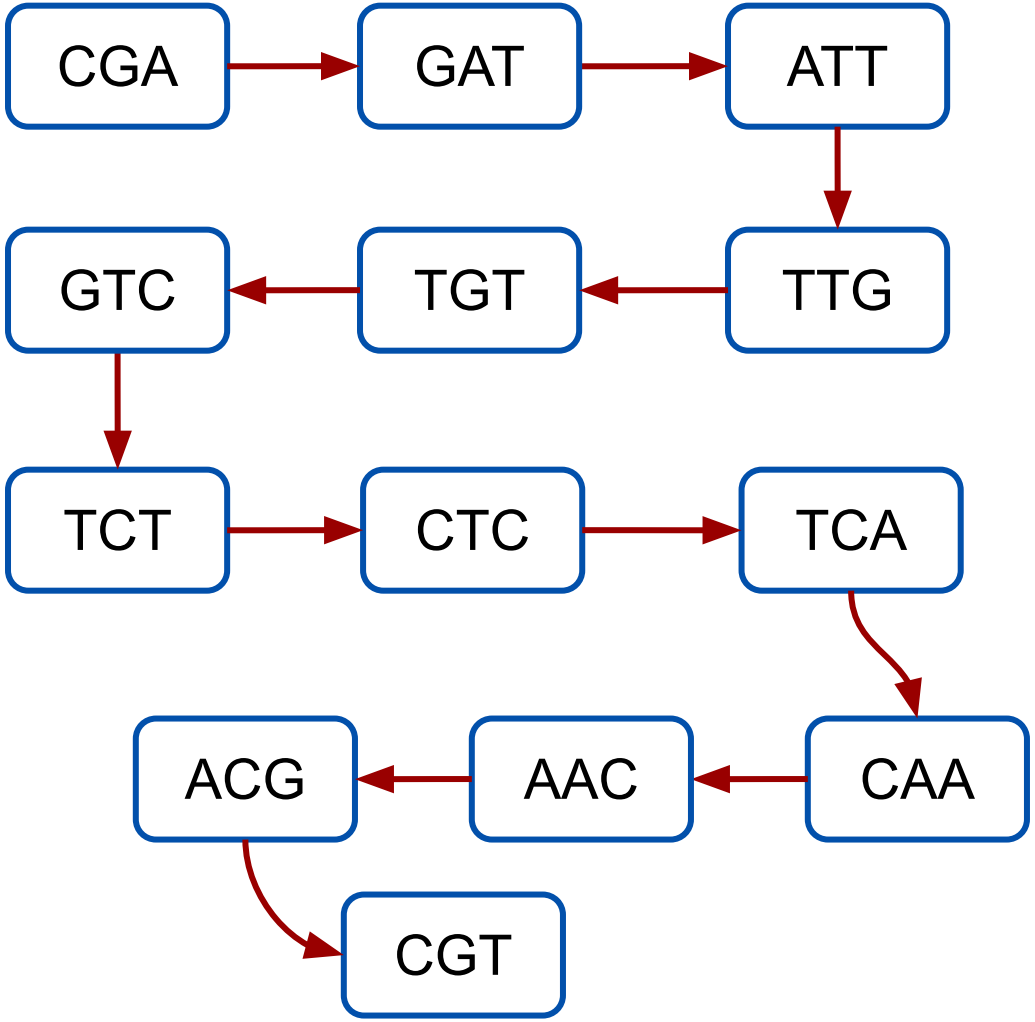
Take a path that goes through each **edge** once



reads:

TCTCAACG
CGATTGTC
TGTCTCAA
CTCAACGT
ATTGTCTC
TTGTCTCA

Take a path that goes through each **edge** once



reads:

TCTCAACG
CGATTGTC
TGTCTCAA
CTCAACGT
ATTGTCTC
TTGTCTCA

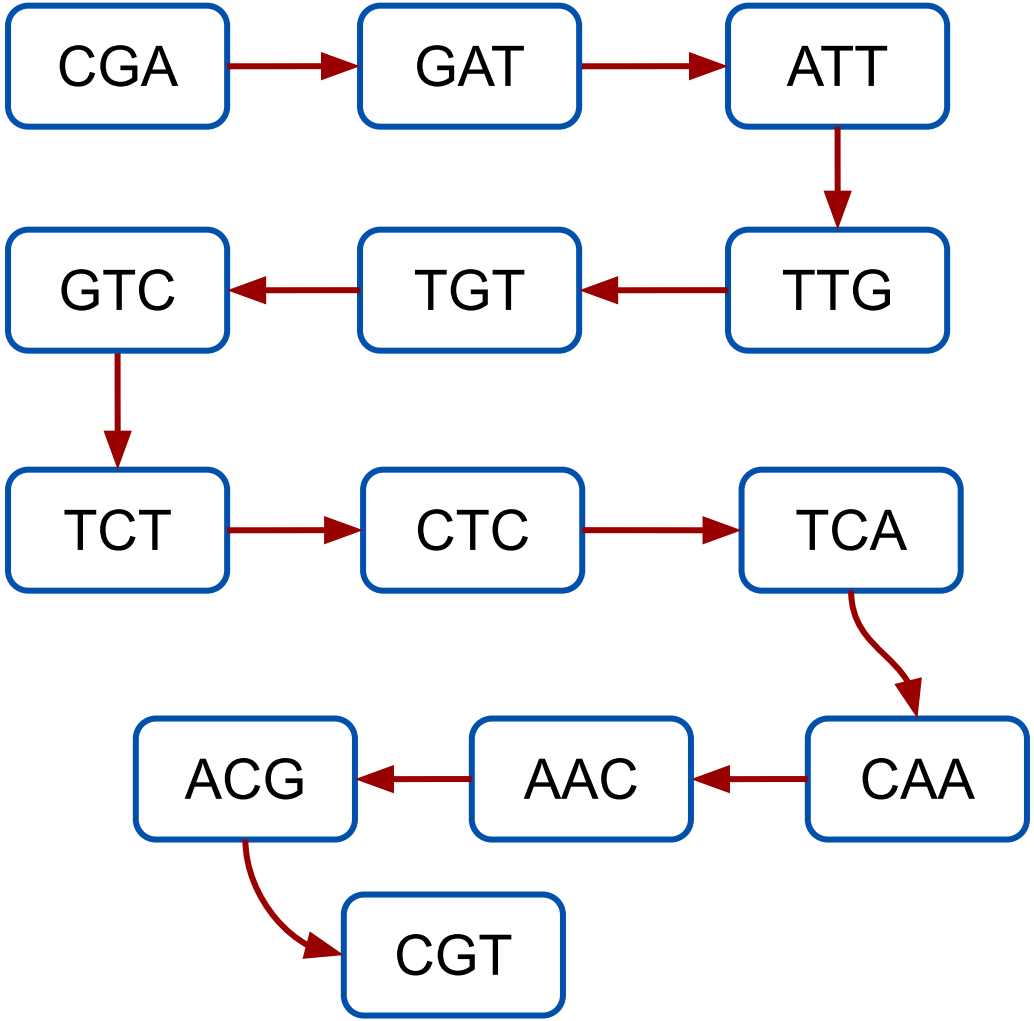
Take a path that goes through each **edge** once

consensus:

CGATTGTCTCAACGT

reads:

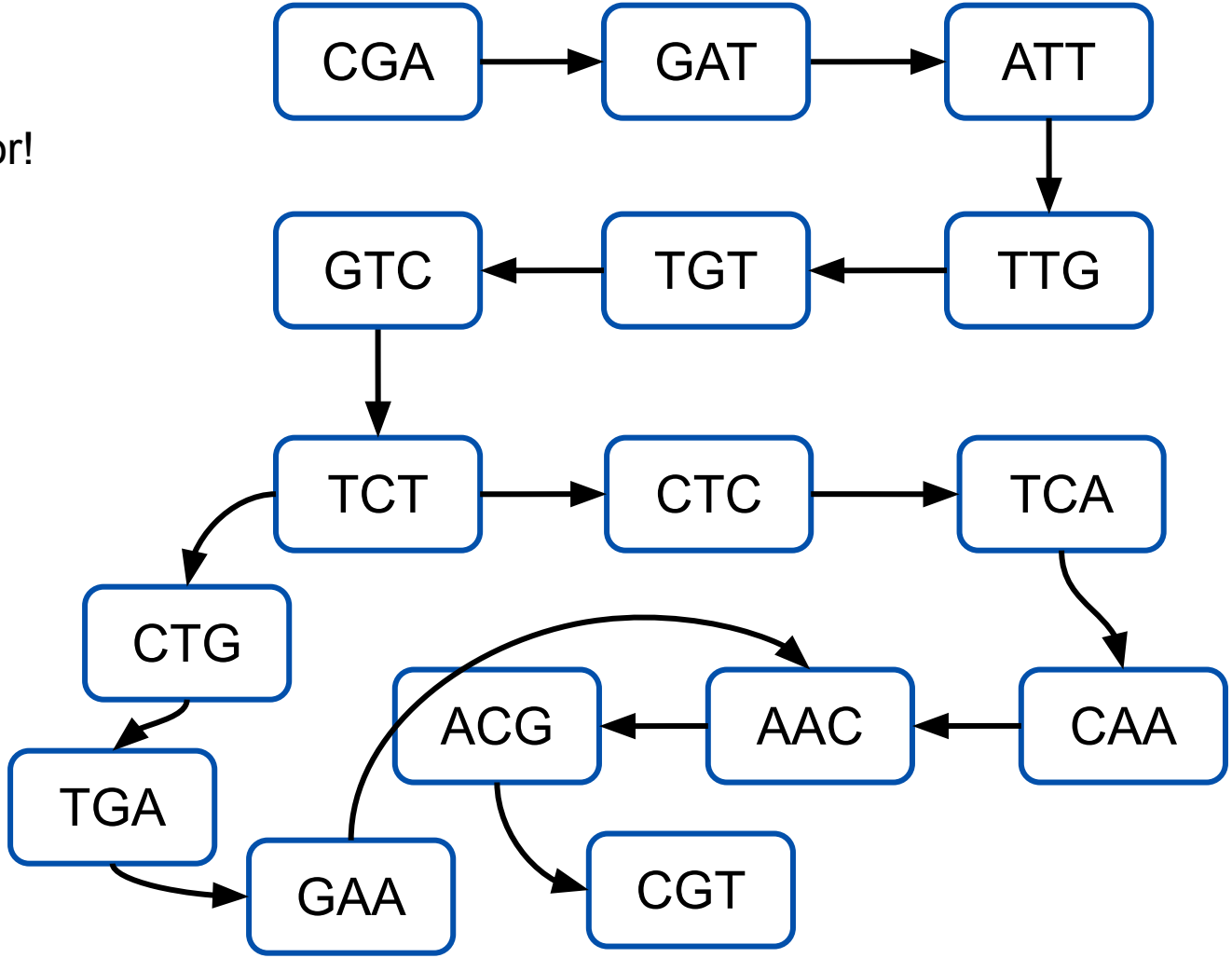
TCTCAACG
CGATTGTC
TGTCTCAA
CTCAACGT
ATTGTCTC
TTGTCTCA



Sequencing error!

reads:

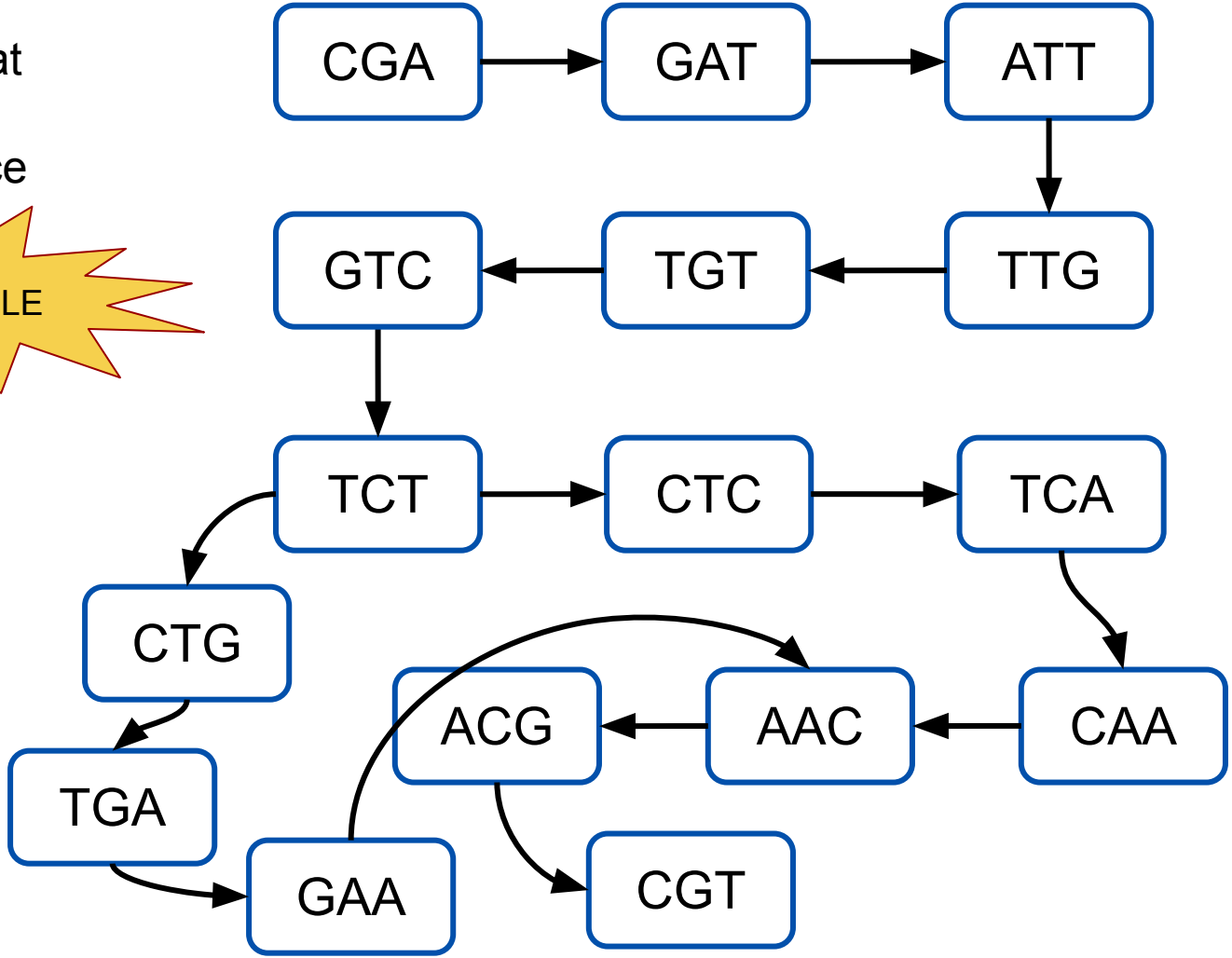
TCT**G**AACG
CGATTGTC
TGTCTCAA
CTCAACGT
ATTGTCTC
TTGTCTCA



Take a path that goes through each **edge** once

reads:

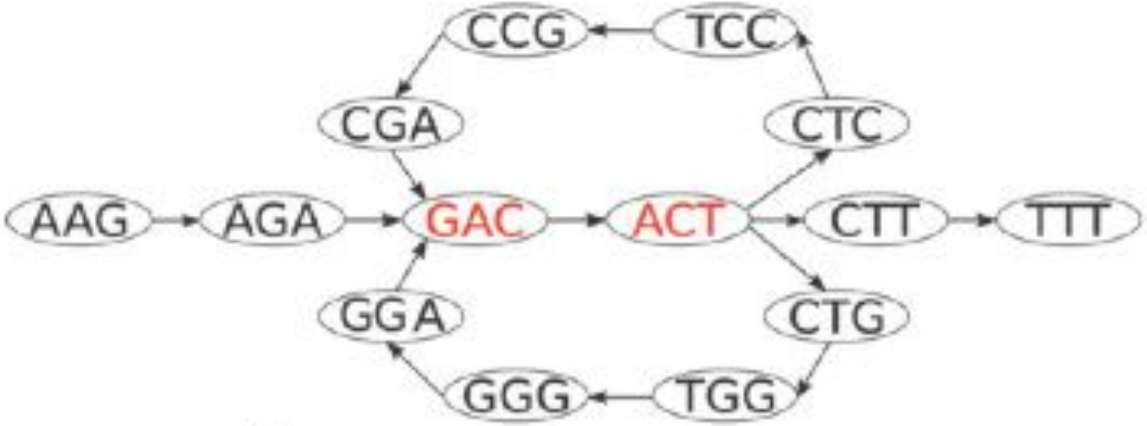
```
TCTGAACG
CGATTGTC
TGTCTCAA
CTCAACGT
ATTGTCTC
TTGTCTCA
```



Create the *de* Bruijn graph of this genome using *k*=3

AAGACTCCGACTGGGACTTT

AAGACTCCGACTGGGACTTT

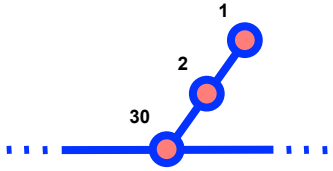


A de Bruijn graph of a sequence

After building: Simplify

Clip tips

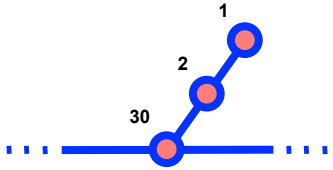
(seq err,end)



After building: Simplify

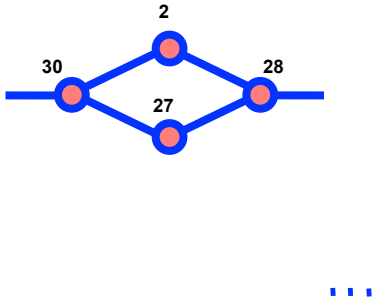
Clip tips

(seq err,end)



Pinch bubbles

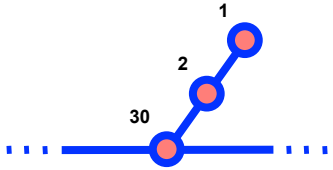
(seq err, middle, SNP)



After building: Simplify

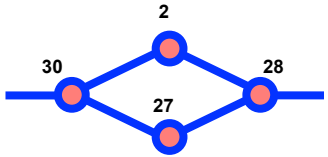
Clip tips

(seq err,end)

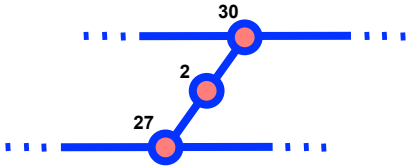


Pinch bubbles

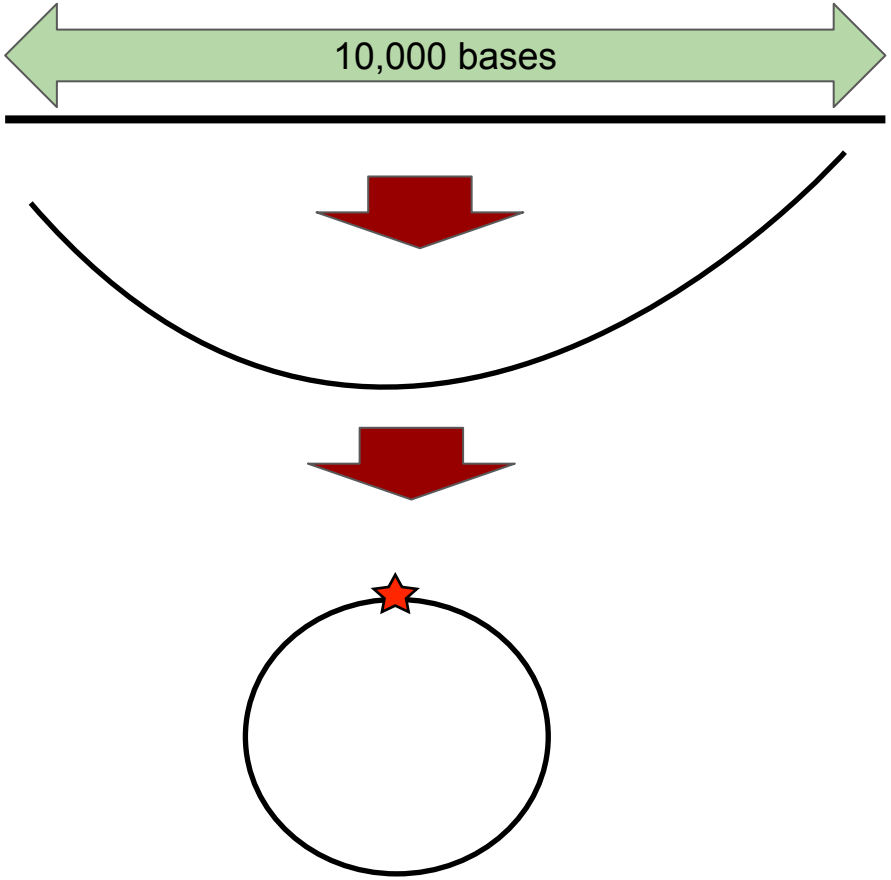
(seq err, middle, SNP)



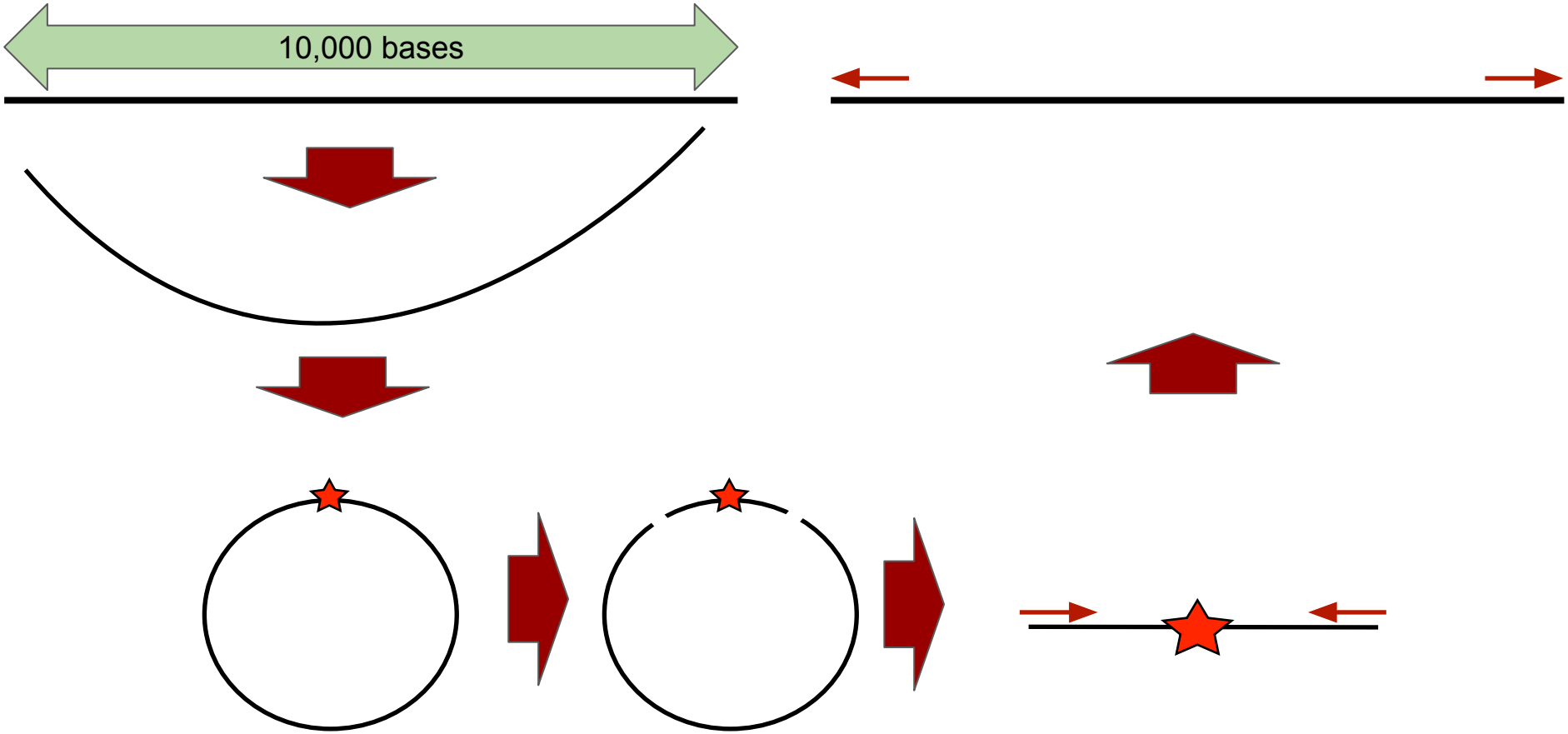
Remove low cov. links



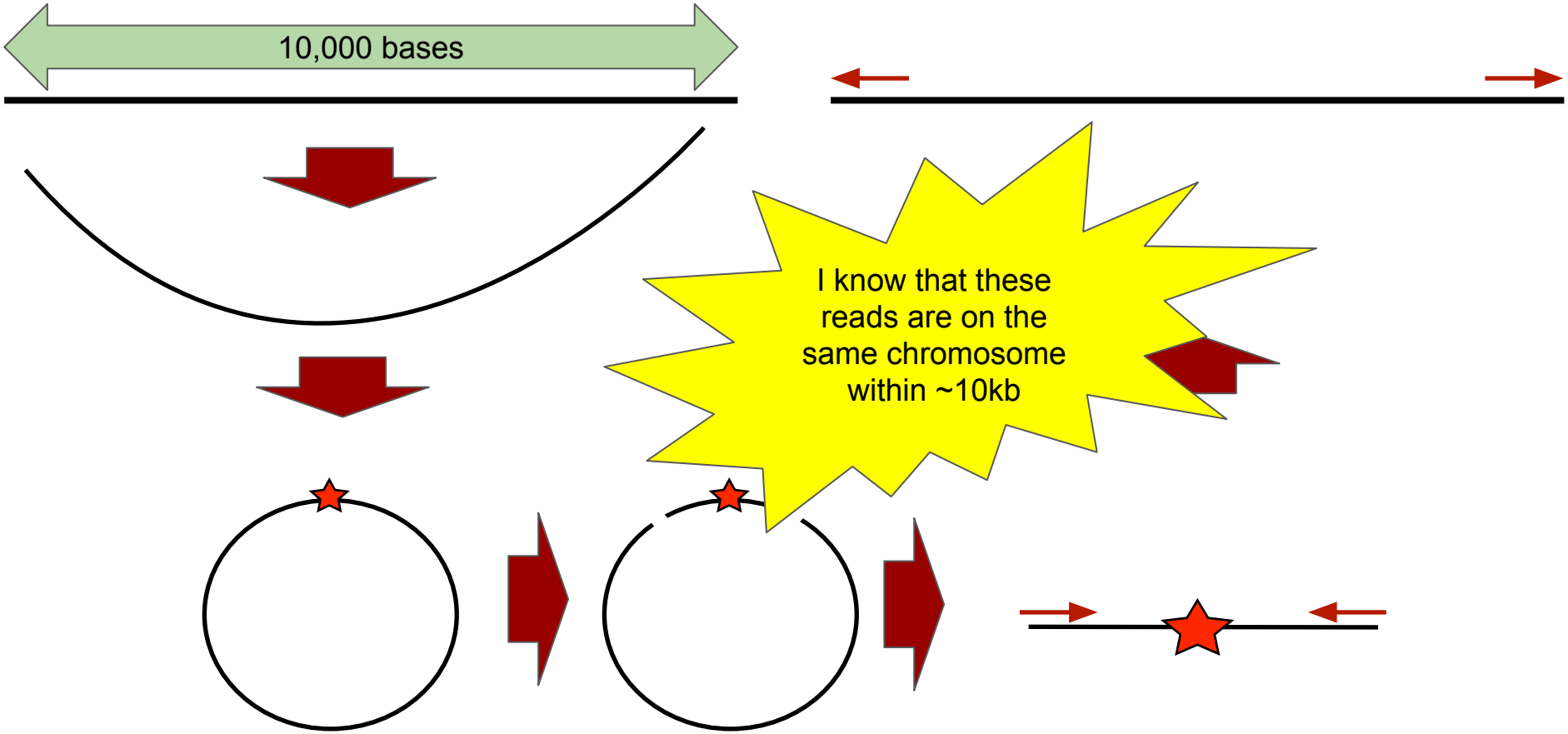
Mate pair reads



Mate pair reads

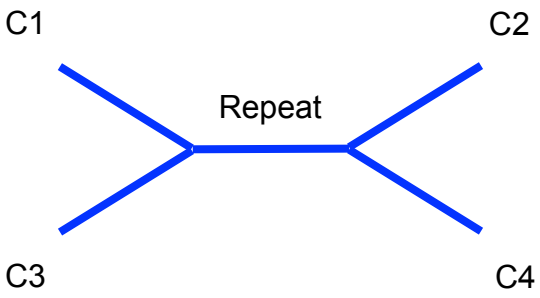


Mate pair reads



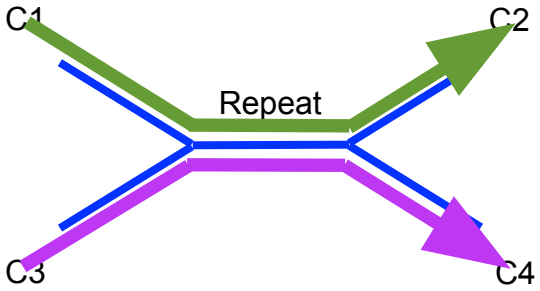
Create contigs and scaffolds

Which goes with which?



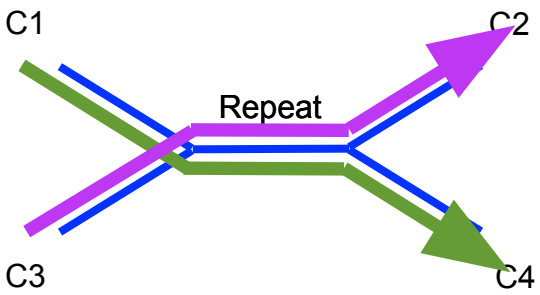
Create contigs and scaffolds

Which goes with which?



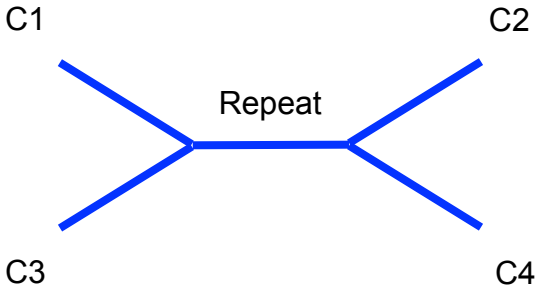
Create contigs and scaffolds

Which goes with which?



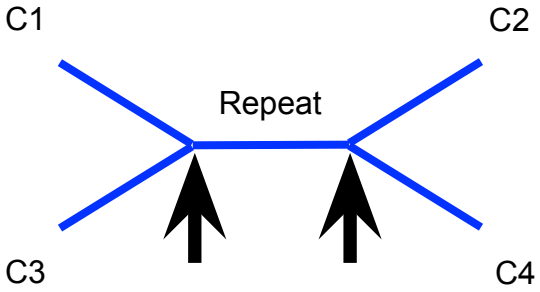
Create contigs and scaffolds

Cut graph at repeat boundaries to create contigs



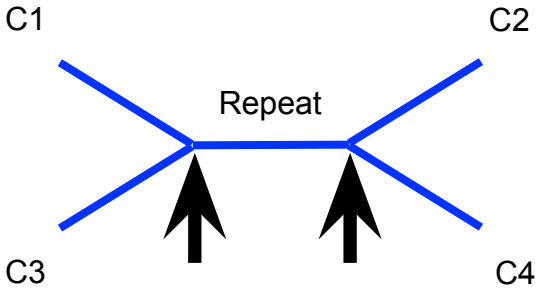
Create contigs and scaffolds

Cut graph at repeat boundaries to create contigs



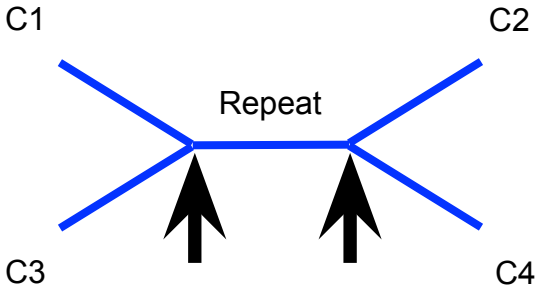
Create contigs and scaffolds

Cut graph at repeat boundaries to create contigs



Create contigs and scaffolds

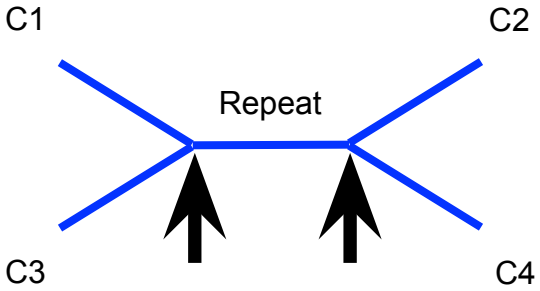
Cut graph at repeat boundaries to create contigs



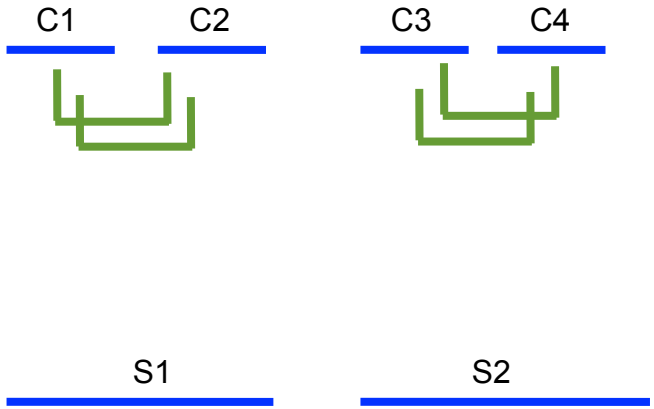
Use paired-end or mate-pair information to resolve repeats and combine to scaffolds

Create contigs and scaffolds

Cut graph at repeat boundaries to create contigs



Use paired-end or mate-pair information to resolve repeats and combine to scaffolds



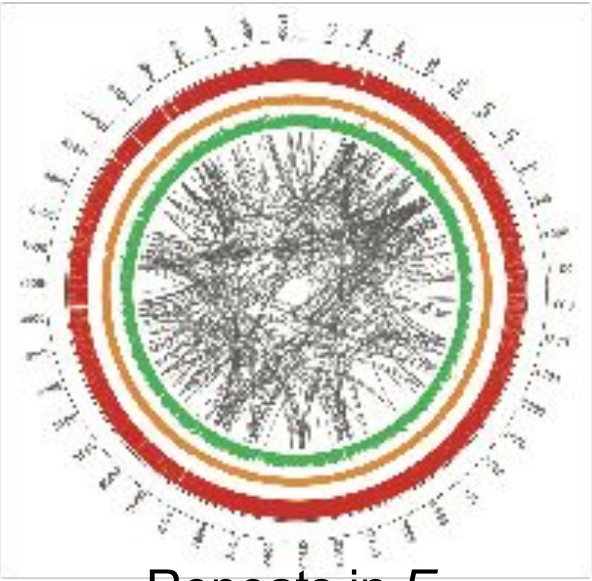
Iterate parameters

- Re-run with different *k*-sizes, find optimum
- Run with multiple k-mers at the same time! (eg. SPAdes)
- Compare assembly statistics such as, assembly length, N50, no. contigs

- Assembly refinement
  - Break contigs not supported by PE/MP reads
  - Analyze assembly using REAPR or QUAST

Successful *de novo* assembly

- Success is a factor of:
  - Genome size, **genomic repeats(!)**, ploidy
  - High coverage, long read lengths, PE/MP libraries



Repeats in *E. coli*

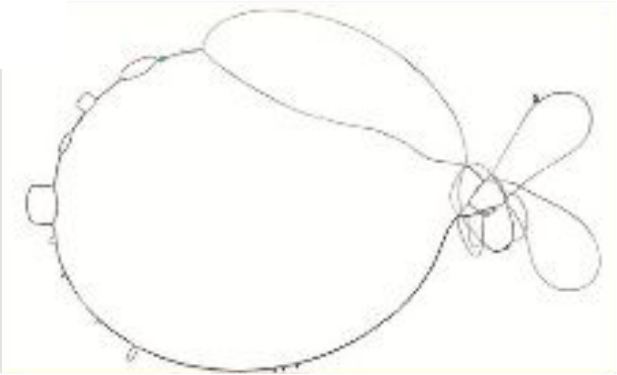
Improving *de novo* assemblies

- Paired-end & Mate-pair for long range continuity
- Hybrid approaches (combine Illumina with PacBio/Oxford Nanopore)

- Synthetic long reads: Illumina Synthetic Reads (Moleculo) or 10X Genomics
- Hi-C contact maps

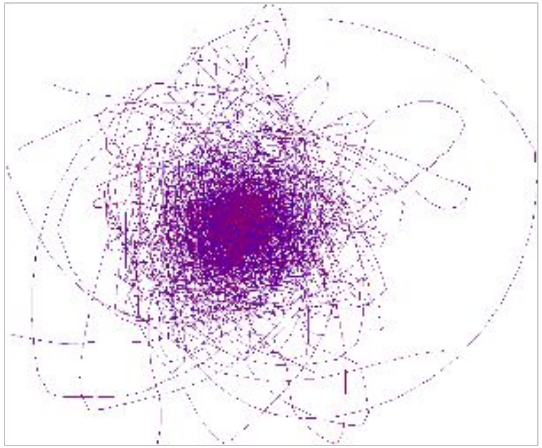
Two bacterial genomes *de Bruijn* graphs

Few repeats



Flicek & Birney, Nat.Methods 2009

"more" repeats

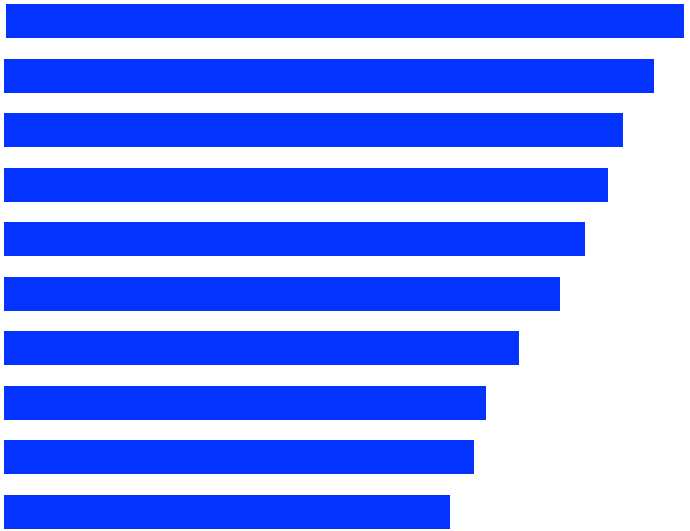


Zerbino, 2009

N50: Assembly quality

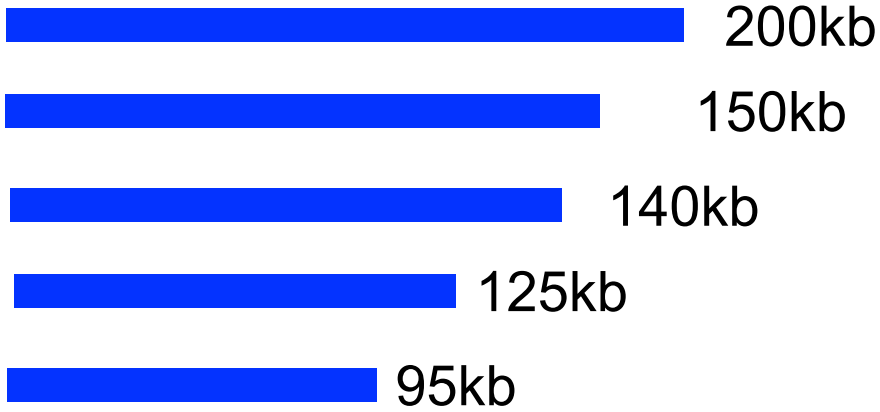
N50: What is the smallest piece in the largest half of the assembly?

- Calculate sum of assembly
- Order contigs by size
- Sum contigs starting by largest
- When half the sum is reached, N50 is the length of the contig



N50 example

5 scaffolds, calculate N50:



Sum: 200+150+140+125+95=710kb

Half: 710 / 2 = 355kb

200kb + 150kb = 350kb

350kb + 140kb = 490kb

490kb > 355kb => **N50: 140kb**

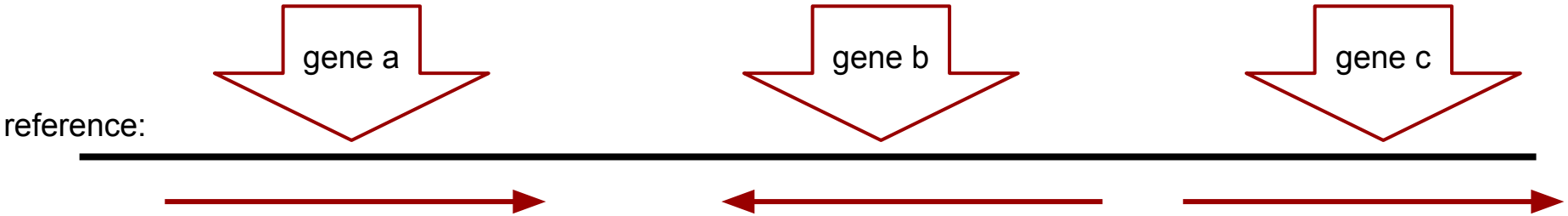
Some assemblers

- OLC: Canu, Newbler, Penguin, CarpeDeam
- de Bruijn: Allpaths-LG, SPAdes, Velvet(best), SOAPdenovo, Megahit (very lean), …
- other: MIRA, SGA, Flye (very good for 3g NGS)

Used in exercises today

You have your assembly, done QC, now what?

gene annotation!



reference:

#ID	start	end	tag
scaffold#292	1000	2000	gene1
scaffold#403	2231	5032	gene2
scaffold#562	1023	4168	gene3

Bioinformatics

Issues | Advance articles | Submit ▾ | Alerts | About ▾ | Bioinformatics

No cover image available

Volume 30, Issue 14
July 2014

JOURNAL ARTICLE

Prokka: rapid prokaryotic genome annotation

Torsten Seemann Author Notes

Bioinformatics, Volume 30, Issue 14, July 2014, Pages 2068–2069,
https://doi.org/10.1093/bioinformatics/btu153
Published: 18 March 2014 **Article history** ▾

PDF | Split View | Cite | Permissions | Share ▾

Article Contents

Abstract

Summary: The multiplex capability and high yield of current day DNA-sequencing instruments has made bacterial whole genome sequencing a routine affair. The subsequent *de novo* assembly of reads into contigs has been well addressed. The final step of annotating all relevant genomic features on those contigs can be achieved slowly using existing web- and email-based systems, but these are not applicable for sensitive data or integrating into computational

JOURNAL ARTICLE

BRAKER2: automatic eukaryotic genome annotation with GeneMark-EP+ and AUGUSTUS supported by a protein database

Tomáš Brůna, Katharina J Hoff, Alexandre Lomsadze, Mario Stanke, Mark Borodovsky
Author Notes

NAR Genomics and Bioinformatics, Volume 3, Issue 1, March 2021, lqaa108,
https://doi.org/10.1093/nargab/lqaa108
Published: 06 January 2021 **Article history** ▾

PDF | Split View | Cite | Permissions | Share ▾

Abstract

The task of eukaryotic genome annotation remains challenging. Only a few genomes could serve as standards of annotation achieved through a tremendous investment of human curation efforts. Still, the correctness of all alternative isoforms, even in the best-annotated genomes, could be a good subject for further investigation. The new BRAKER2 pipeline generates and integrates external protein support into the iterative process of training and gene prediction by GeneMark-EP+ and AUGUSTUS. BRAKER2 continues the line started by BRAKER1 where self-training GeneMark-ET and AUGUSTUS made

Research | Open access | Published: 31 August 2023

Galba: genome annotation with miniprot and AUGUSTUS

Tomáš Brůna, Heng Li, Joseph Guhlin, Daniel Honsel, Steffen Herbold, Mario Stanke, Natalia Nenasheva, Matthis Ebel, Lars Gabriel & Katharina J. Hoff

BMC Bioinformatics **24**, Article number: 327 (2023) | Cite this article

Abstract

Background

How to store annotations?

The BED (Browser Extensible Data) format:

chromosome/
scaffold ID

first base -1

last base -1

```
chr2    3924893    3929684
chr2    4483132    4493023
chr3    5060342    5064066
...
```

How to store annotations?

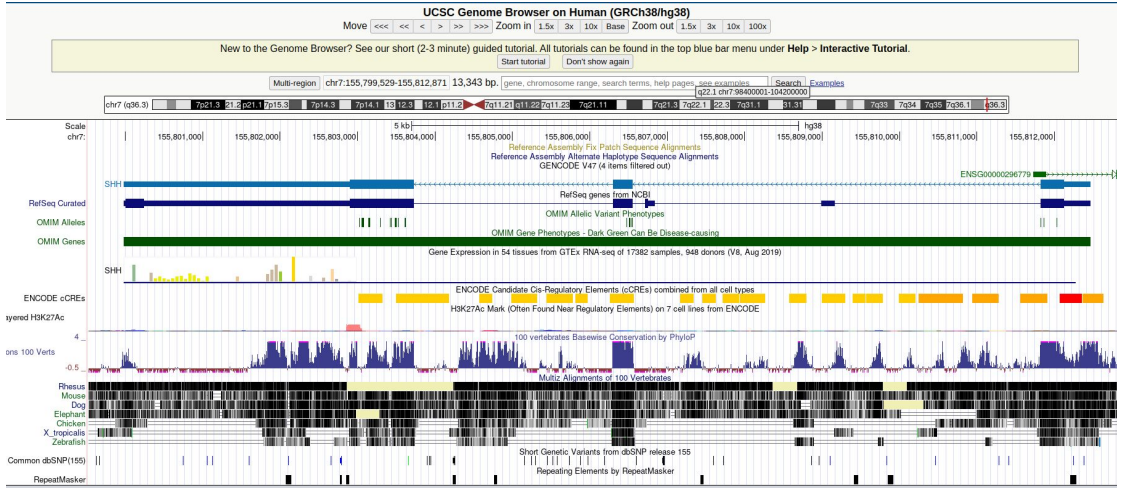
The GFF (General Feature Format) format:

```
browser position chr22:10000000-10025000
browser hide all
track name=regulatory description="TeleGene(tm) Regulatory Regions" visibility=2
chr22   TeleGene        enhancer        10000000        10001000        500     +       .       touch1
chr22   TeleGene        promoter        10010000        10010100        900     +       .       touch1
chr22   TeleGene        promoter        10020000        10025000        800     -       .       touch2
```

source: https://genome.ucsc.edu/FAQ/FAQformat.html#format3

Other types of genome annotations

- Repeats: LINE1 etc
- Comparitive genomics: conserved elements
- Transcription regulation



https://genome.ucsc.edu/

Bedtools is a fast, flexible toolset for genome arithmetic.

Bedtools links

- Issue Tracker
- Source @ GitHub
- Old Releases @ Google Code
- Mailing list @ Google Groups
- Queries @ Biostar
- Quinlan lab @ UU

Sources

bedtools: *a powerful toolset for genome arithmetic*

Collectively, the **bedtools** utilities are a swiss-army knife of tools for a wide-range of genomics analysis tasks. The most widely-used tools enable *genome arithmetic*: that is, set theory on the genome. For example, **bedtools** allows one to *intersect*, *merge*, *count*, *complement*, and *shuffle* genomic intervals from multiple files in widely-used genomic file formats such as BAM, BED, GFF/GTF, VCF. While each individual tool is designed to do a relatively simple task (e.g., *intersect* two interval files), quite sophisticated analyses can be conducted by combining multiple bedtools operations on the UNIX command line.

bedtools is developed in the Quinlan laboratory at the University of Utah and benefits from fantastic contributions made by scientists worldwide.

Tutorial

- We have developed a fairly comprehensive tutorial that demonstrates both the basics, as well as some more advanced examples of how bedtools can help you in your research. Please have a look.
- Robert Aboukhalil has developed sandbox.bio an excellent, web-based playground for the bedtools tutorial and other widely-used genomics tools.

Important notes

Exercise time!