

# UBC Bioinformatics Class

Topic 5: de novo assembly

# Outcomes

- Identify the difference between de novo assembly and reference guided alignment
- Evaluate two different approaches to de novo genome assembly
- Describe how repetitive elements can hamper proper assembly and compare approaches that can overcome this problem
- Describe approaches for transcriptome/GBS de novo assembly

# Introduction



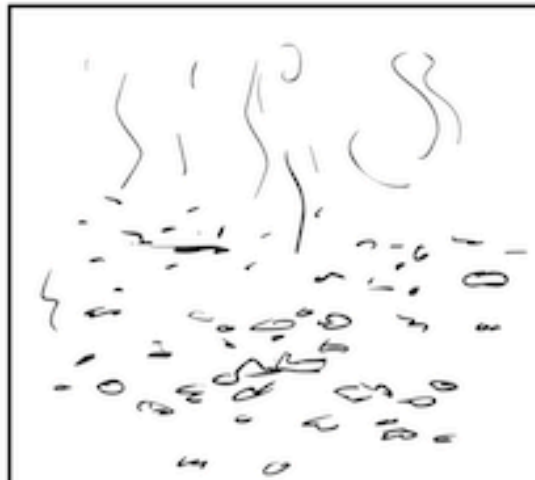
stack of NY Times, June 27, 2000



stack of NY Times, June 27, 2000  
on a pile of dynamite



this is just hypothetical



so, what did the June 27, 2000 NY  
Times say?

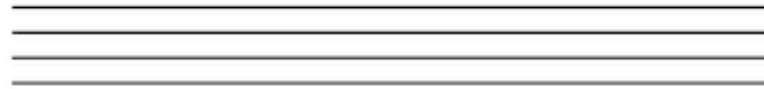
# Introduction

atshirt, approximately  
e have not yet named a  
information is welc

shirt, approximately 6'2" 18  
t yet named any suspects  
is welcomed. Please call

# Introduction

Multiple identical  
copies of a genome



Shatter the genome  
into reads



Sequence the reads



Assemble the  
genome using  
overlapping reads



## Alignment vs assembly

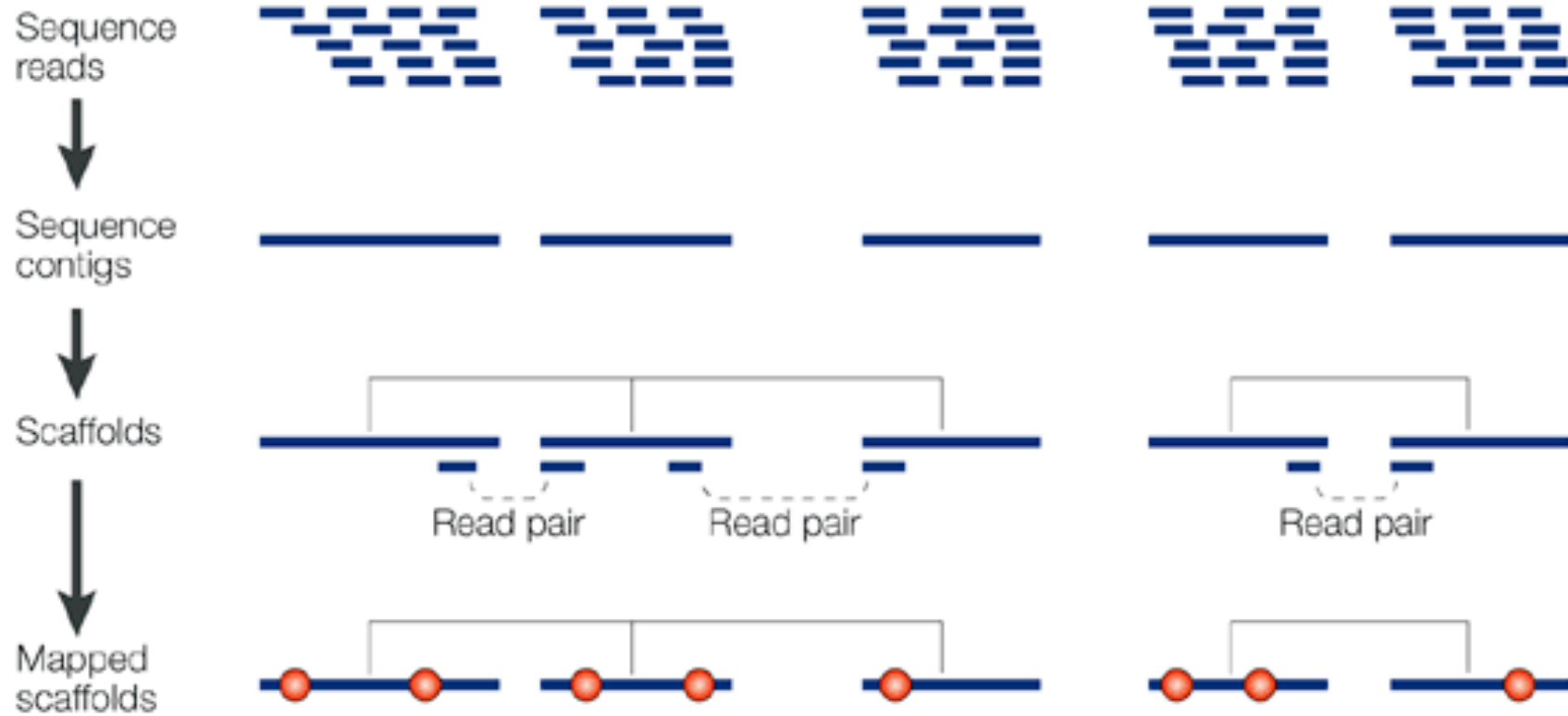
Aligning to a reference:

- Reference guided alignments: align the reads to a reference genome and looks for differences

Building a reference:

- De novo assembly: no previous genome assembly is used
- Comparative genome assembly: assemble a newly sequenced genome by mapping it on to a reference
- Hybrid approach: reference-guided and de novo for unused reads or de novo and then reference guided alignments

# Introduction

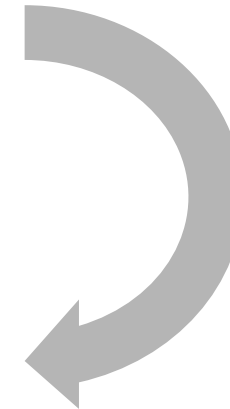


# Introduction

Original sequence

GATAGAAGGGTCCGCTCGCTCAGCTACCGGTTTTTATAGATCTA

GATAGAAGGGTCCGCT  
AGAAGGGTCCGCTC  
GGGTCCGCTCGCTCA  
CCGCTCGCTCAGC  
CTCGCTCAGCTACC  
TCAGCTACCGGTTT  
CTACCGGTTTTT  
AGCTACCGGTTTTTAT  
TTTTTATAGATCTA



fragmented sequences  
from sequencer  
(reads)



# Introduction

**assembled** TTTTTATAGATCTA  
AGCTACCGGTTTTTAT  
fragmented sequences CAGCTACCGGTTTTT  
from sequencer TCAGCTACCGGTTT  
(reads) CTCGCTCAGCTACC  
CCGCTCGCTCAGC  
GGGTCCGCTCGCTCA  
AGAAGGGTCCGCTC  
GATAGAAGGGTCCGCT  
  
GATAGAAGGGTCCGCTCGCTCAGCTACCGGTTTTTATAGATCTA

We want to reconstruct this from the reads

# Introduction

## Simplified scenario

- Single strand
- Error free
- Complete coverage

```
                TTTTTATAGATCTA
            AGCTACCGGTTTTTAT
        CAGCTACCGGTTTTT
    TCAGCTACCGGTTT
        CTCGCTCAGCTACC
            CCGCTCGCTCAGC
                GGGTCCGCTCGCTCA
                    AGAAGGGTCCGCTC
                        GATAGAAGGGTCCGCT

GATAGAAGGGTCCGCTCGCTCAGCTACCGGTTTTTATAGATCTA
```

# Introduction

Coverage: reads “covering” a position in the genome  
(average or at a single base or region)

```

                                TTTTATAGATCTA
                                AGCTACCGGTTTTAT
                                CAGCTACCGGTTTT
                                TCAGCTACCGGTTT
                                CTCGCTCAGCTACC
                                CCGCTCGCTCAGC
                                GGGTCCGATCGCTCA
                                AGAAGGGTCCGCTC
                                GATAGAAGGGTCCGCT
                                ↓
                                44 bases in the “genome”
                                GATAGAAGGGTCCGCTCGCTCAGCTACCGGTTTTTATAGATCTA
```

What is our average coverage?

What is the coverage at the arrow?

# Introduction

```
                TTTTTATAGATCTA
              AGCTACCGGTTTTTAT
             CAGCTACCGGTTTTT
            TCAGGTACCGGTTT
           CTCGCTCAGCTACC
          CCGCTCGCTCAGC
         GGGTCCGATTGCTCA
        AGAAGGGTCCGCTC
       GATAGAAGGGTCCGCT
```

GATAGAAGGGTCCGCTCGCTCAGCTACCGGTTTTTATAGATCTA

Why might there be differences among reads covering the same position?

# Introduction

CCGCTCGCTCAGC  
TCAGCTACCGGTTT  
CTCGCTCAGCTACC  
CAGCTACCGGTTTTT  
AGAAGGGTCCGCTC  
GATAGAAGGGTCCGCT  
AGCTACCGGTTTTTAT  
TTTTTATAGATCTA  
GGGTCCGCTCGCTCA

How would you go about “assembling” these reads when you have no reference?

## Code break

Write a one liner to find all the overlaps exactly 4 bp in length between **CTCTAGGCC** and a list of other sequences in the file `/home/biol525d/Topic_5/data/overlaps.fa`

# Overlap-layout-consensus

Overlap: make an overlap graph

Layout: find the path through the graph

Consensus: find the most likely contig sequence

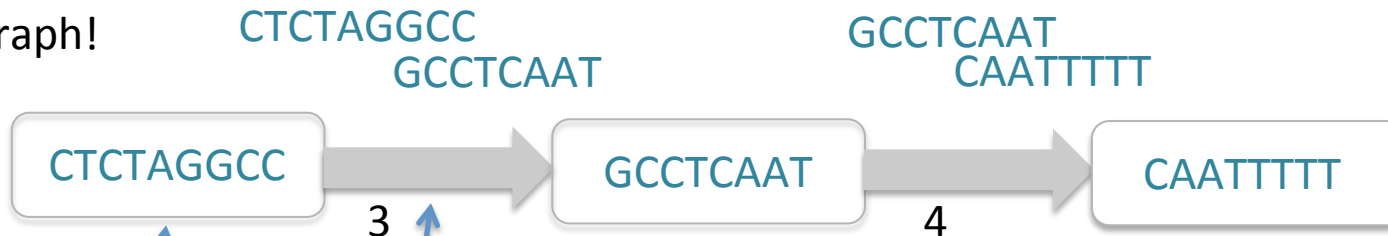
OLC programs:

ARACHNE, PHRAP, CAP, TIGR, CELERA

# Overlap-layout-consensus

Reads: CTCTAGGCC      GCCTCAAT      CAATTTTT

This is a graph!  
(directed)



Node or vertex=  
read

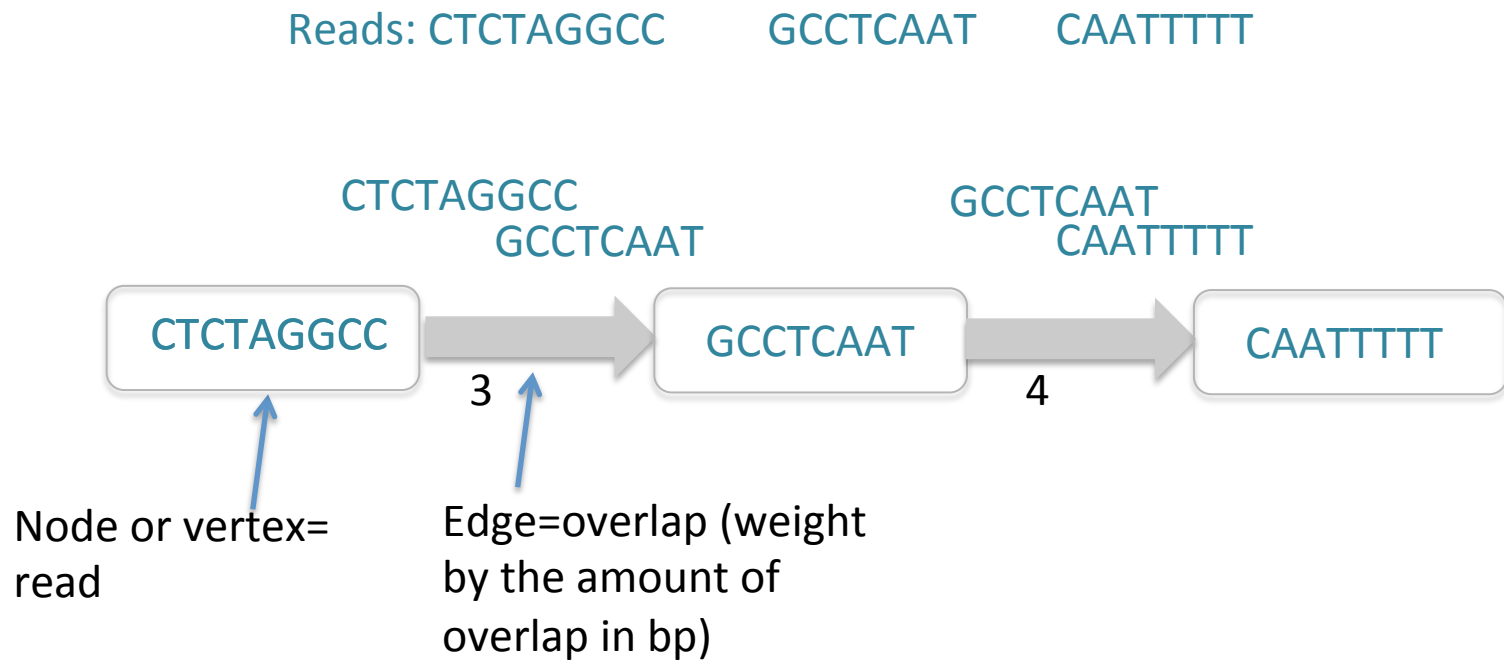
Edge=overlap (weight  
by the amount of  
overlap in bp)

Can pick a minimum overlap length (e.g. 3 bp)

Finding overlaps can be computationally challenging  
when you have millions of reads!



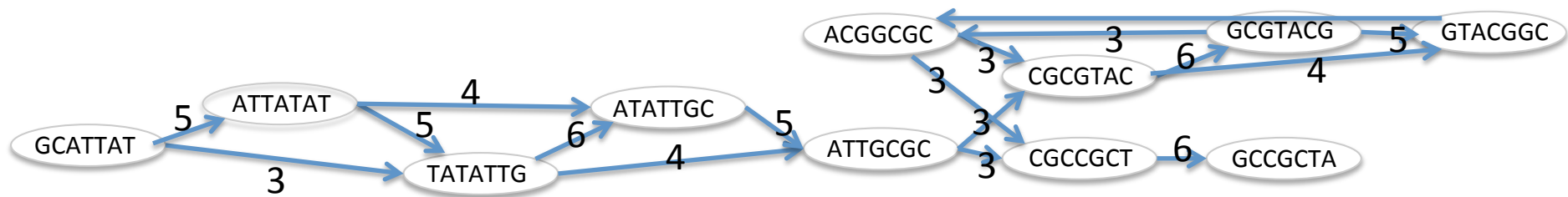
# Overlap-layout-consensus



Here we have only one path through the graph

# Overlap-layout-consensus

These graphs get complicated!



Minimum  
overlap = 3  
Read length = 7

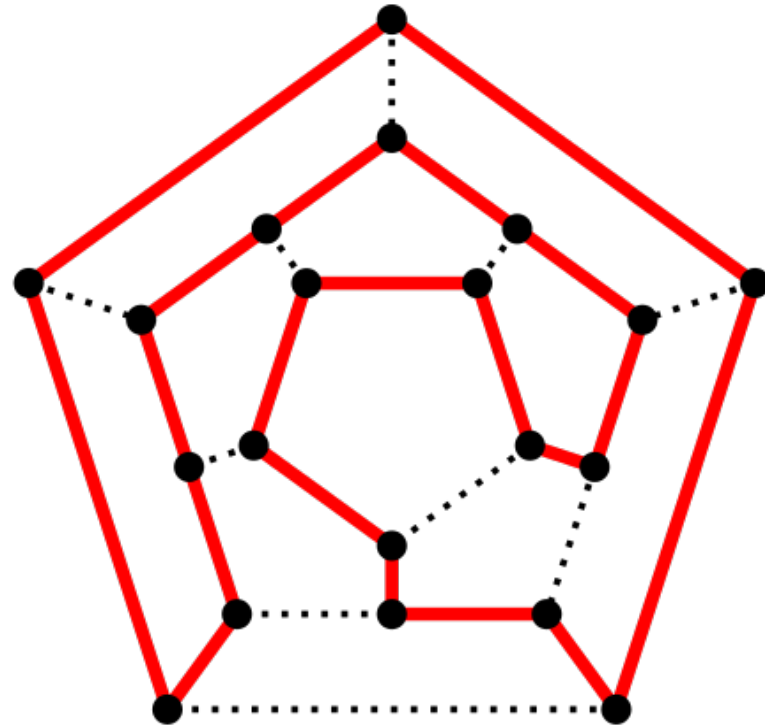
GCATTATATATTGCGCGTACGGCGCCGCTACA

Original sequence

How can we find the best path?

# Overlap-**layout**-consensus

Hamiltonian path: hit each node (read) once  
–no quick way to figure it out (NP-complete)  
–not practical and not implemented



# Overlap-**layout**-consensus

Shortest superstring: find the shortest final sequence (greatest overlap between reads)  
-hit each node (read) once  
-NP-hard

# Overlap-**layout**-consensus

Greedy algorithm (example)

- 1) Pairwise alignments between all fragments
- 2) Pick the two with the largest overlap
- 3) Merge chosen fragments
- 4) Repeat



the greedy one

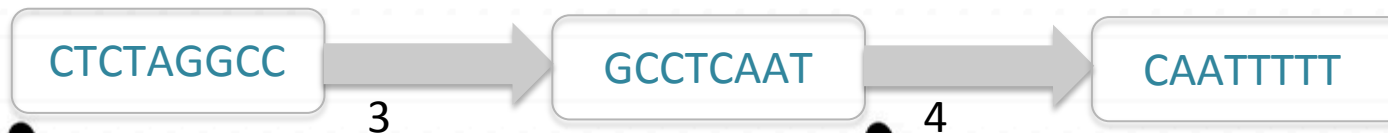
# Overlap-layout-**consensus**

Join sequences together into one sequence

Reads: CTCTAGGCC      GCCTCAAT      CAATTTTT

CTCTAGGCC  
GCCTCAAT

GCCTCAAT  
CAATTTTT



CTCTAGGCC  
GCCTCAAT  
CAATTTTT

CTCTAGGCCTCAATTTTT



# Overlap-layout-**consensus**

## Limitations of OLC

- require overlaps to be scored between all possible pairs of reads. This is a problem when you have millions of reads
- finding the best path through the graph with a huge number of nodes (reads) is computationally challenging

Is there a faster way to assemble many short reads?

# De Bruijn graphs

What are all the 5-mers (5 bp fragments) in these reads?

2 reads of 9 bp

read 1

ATGGGGAAC

read 2

GGGAACCCC

ATGGG

GGGAA

TGGGG

GGAAC

GGGGA

GAACC

GGGAA

AACCC

GGAAC

ACCCC

If a read is  $L$  bp long, how many kmers of size  $k$  can you make?



# Code break

Find all the unique 9mers in a fasta sequence and sort them alphabetically [/home/biol525d/Topic\\_5/data/kmer.fa](/home/biol525d/Topic_5/data/kmer.fa)

1. Find all the kmers in this fasta sequence.

Hints: test out the following commands

```
cut -c2- kmer.fa
```

```
cut -c1-4 kmer.fa
```

```
for num in {1..10}
```

```
do
```

```
echo $num >> file.txt
```

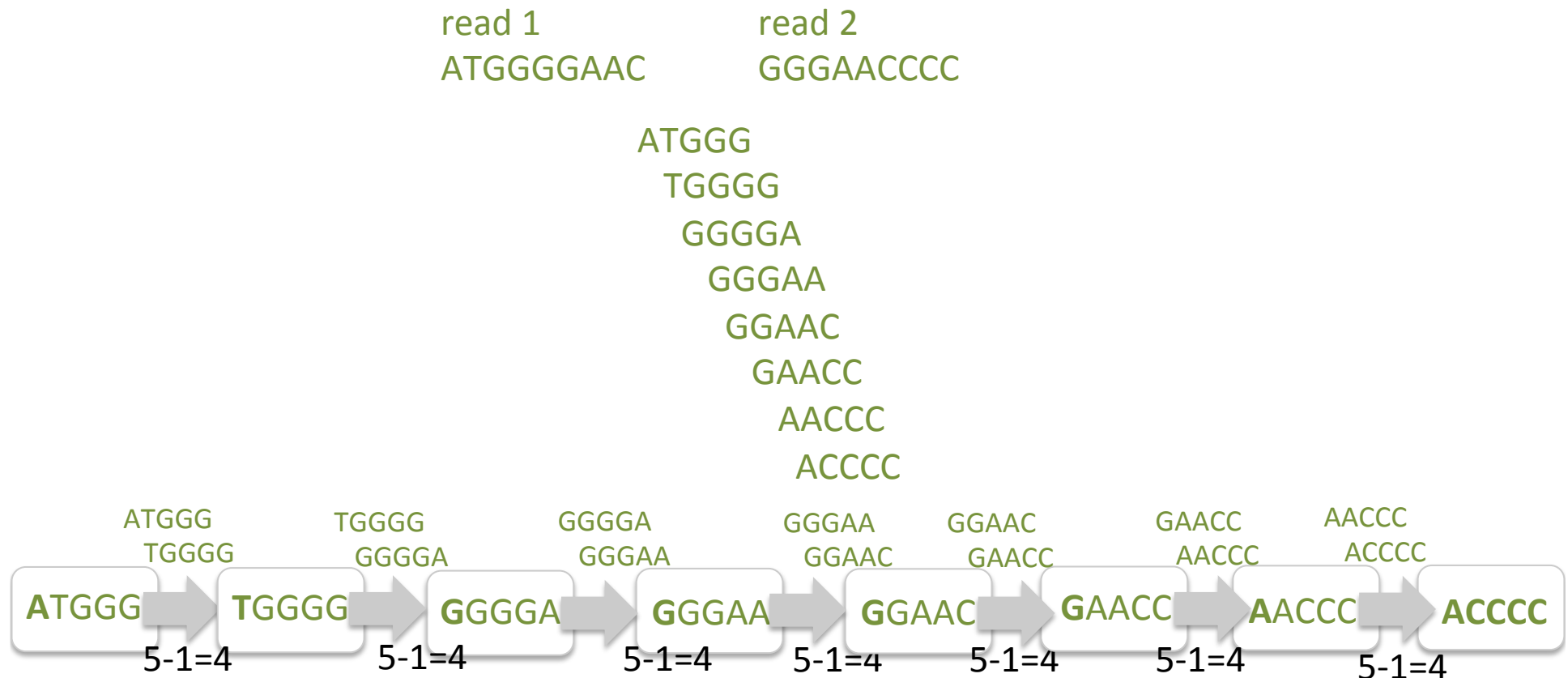
```
done
```

2. Sort them and keep the unique ones

Hint: try `sort`

# De Bruijn graphs

- Join up all the k-mers (length = k bp) into a graph with an overlap of k-1 (here k=5)



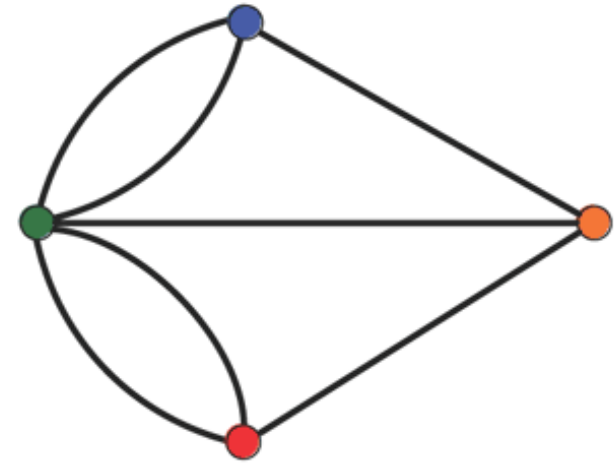
- Traverse through the graph
- The first base of each node spells out the sequence

# De Bruijn graphs

a

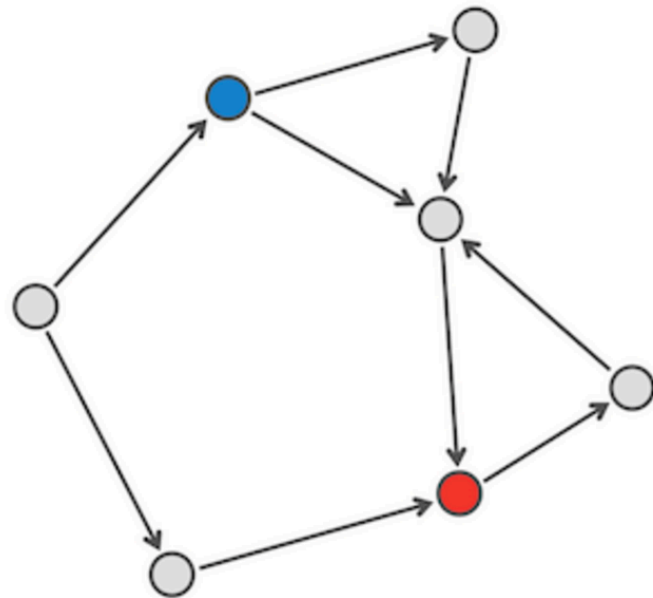
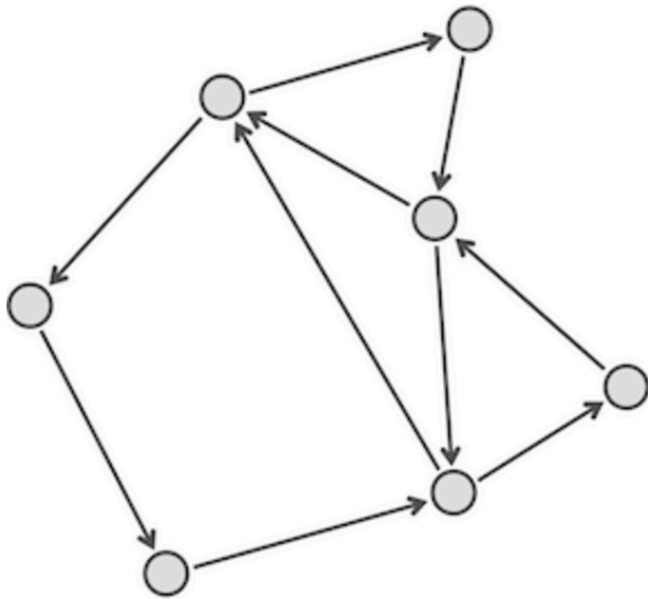


b

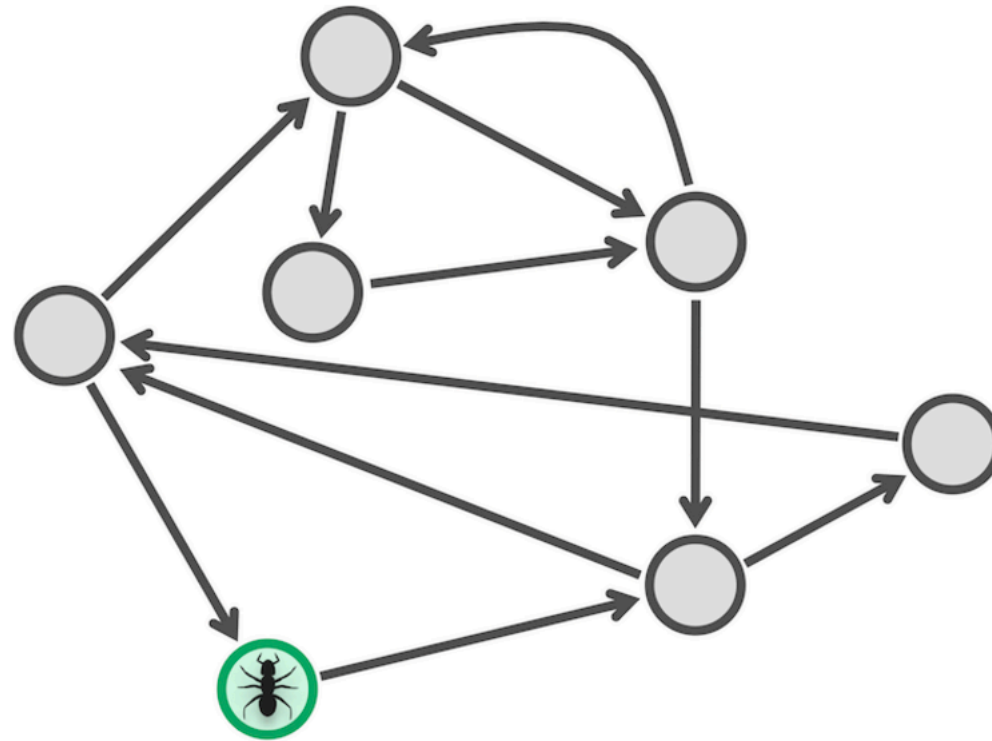


# De Bruijn graphs

Eulerian graph must be both balanced and strongly connected

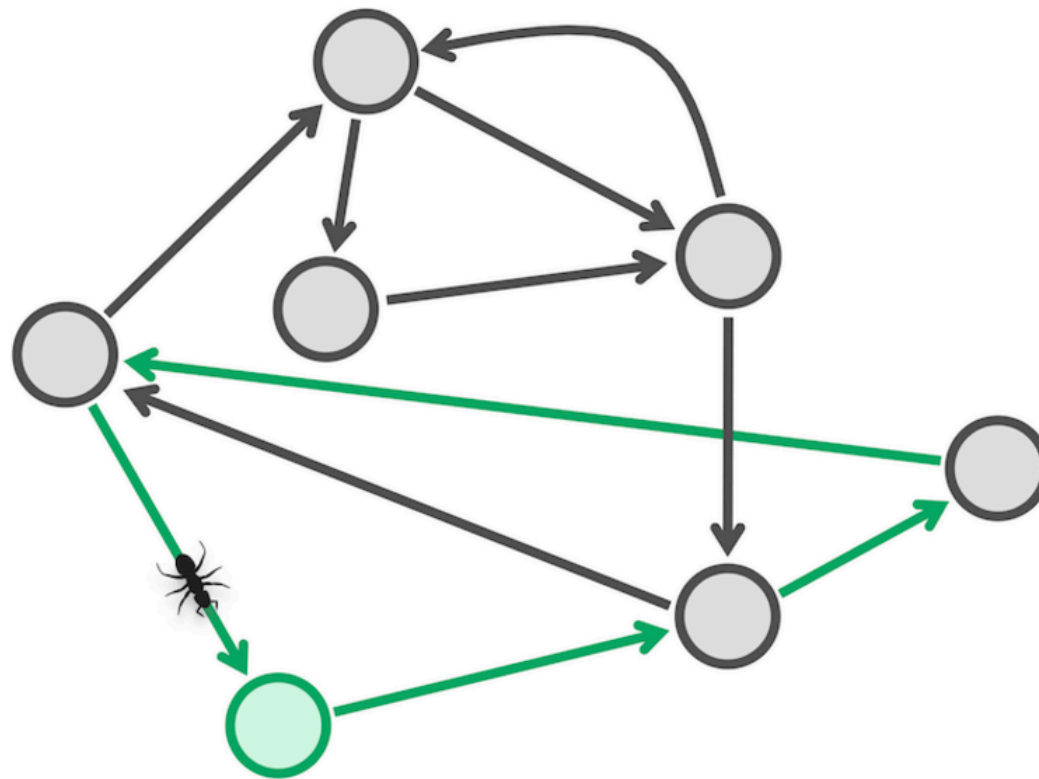


# De Bruijn graphs



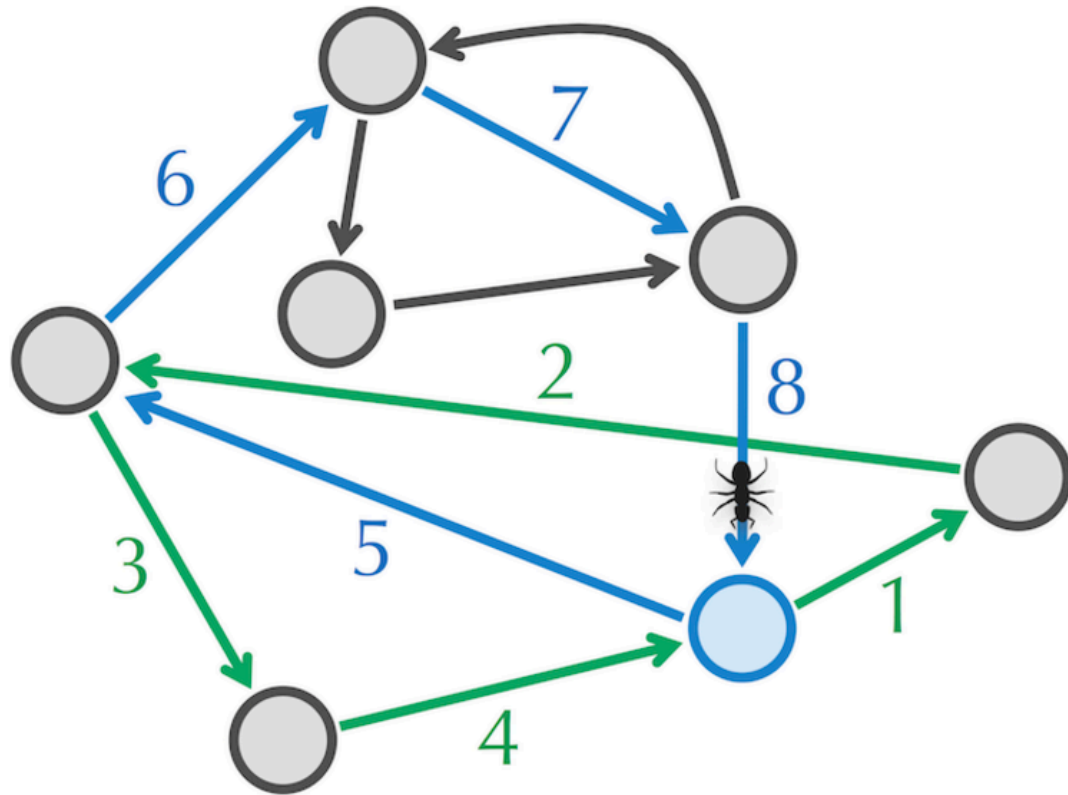
Algorithm to find a path through an Eulerian graph

# De Bruijn graphs



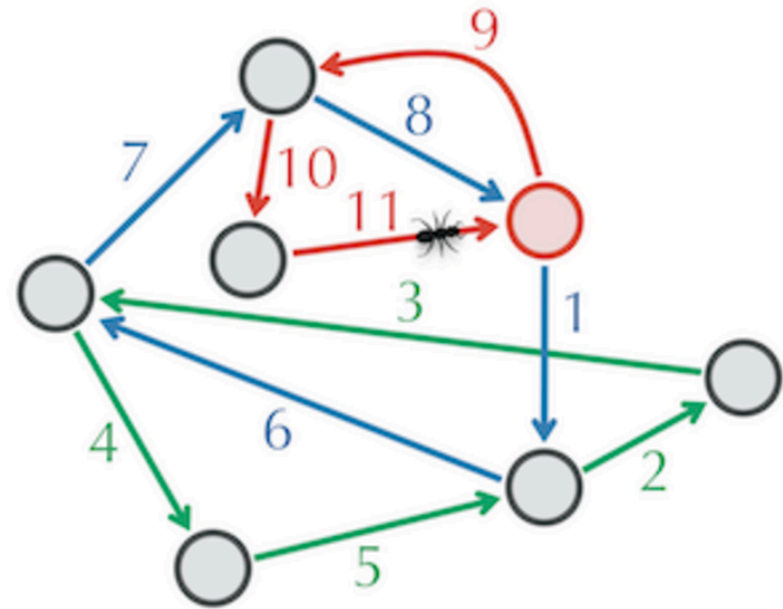
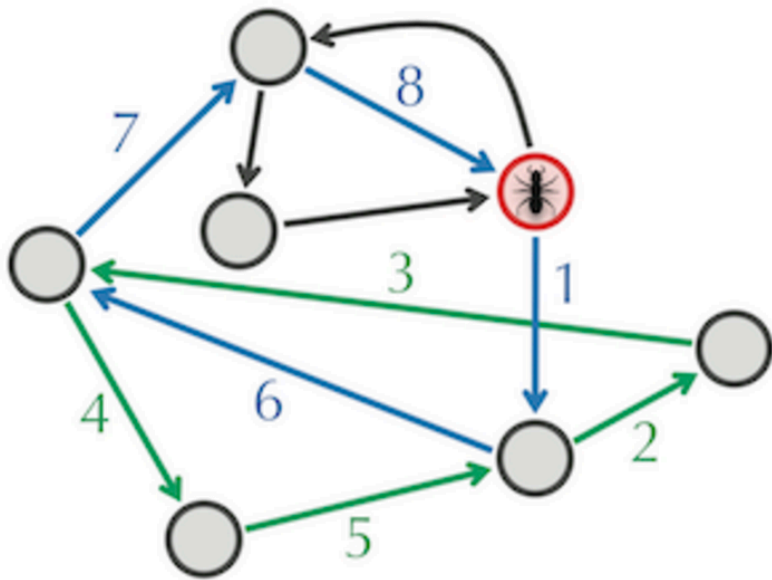
Algorithm to find a path through an Eulerian graph

# De Bruijn graphs



Algorithm to find a path through an Eulerian graph

# De Bruijn graphs



Algorithm to find a path through an Eulerian graph



# De Bruijn graphs

Limitations of the Eulerian path:

- With “perfect” genomic data there are usually many Eulerian tours
- Data is not perfect (areas of low coverage, errors, repeats, etc.)

# De Bruijn graphs

TAGTCGAGGCTTTAGATCCGATGAGGCTTTAGAGACAG

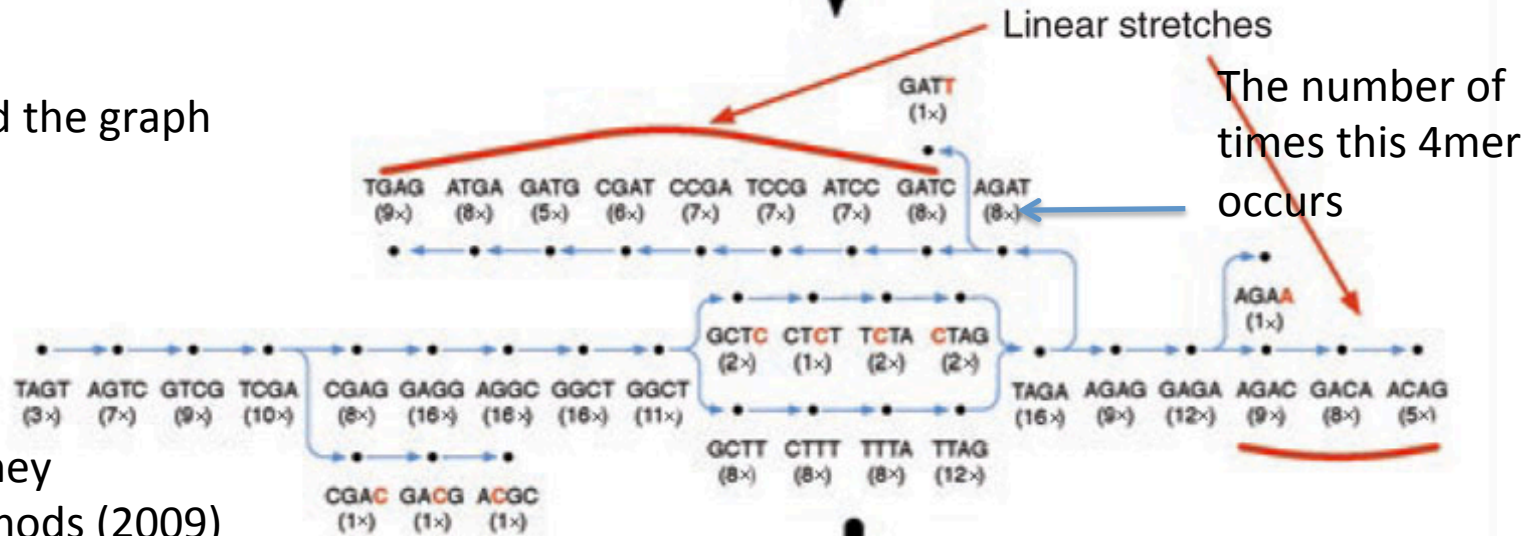
1. Sequence

AGTCGAG	CTTTAGA	CGATGAG	CTTTAGA
GTCGGG	TTAGATC	ATGAGGC	GAGACAG
GAGGCTC	ATCCGAT	AGGCTTT	GAGACAG
AGTCGAG	TAGATCC	ATGAGGC	TAGAGA
TAGTCGA	CTTTAGA	CCGATGA	TTAGAGA
CGAGGCT	AGATCCG	TGAGGCT	AGAGACA
TAGTCGA	GCTTTAG	TCCGATG	GCTCTAG
TCGACGC	GATCCGA	GAGGCTT	AGAGACA
TAGTCGA	TTAGATC	GATGAGG	TTTAGAG
GTCGAGG	TCTAGAT	ATGAGGC	TAGAGAC
AGGCTTT	ATCCGAT	AGGCTTT	GAGACAG
AGTCGAG	TTAGATT	ATGAGGC	AGAGACA
GGCTTTA	TCCGATG	TTTAGAG	
CGAGGCT	TAGATCC	TGAGGCT	GAGACAG
AGTCGAG	TTTAGATC	ATGAGGC	TTAGAGA
GAGGCTT	GATCCGA	GAGGCTT	GAGACAG

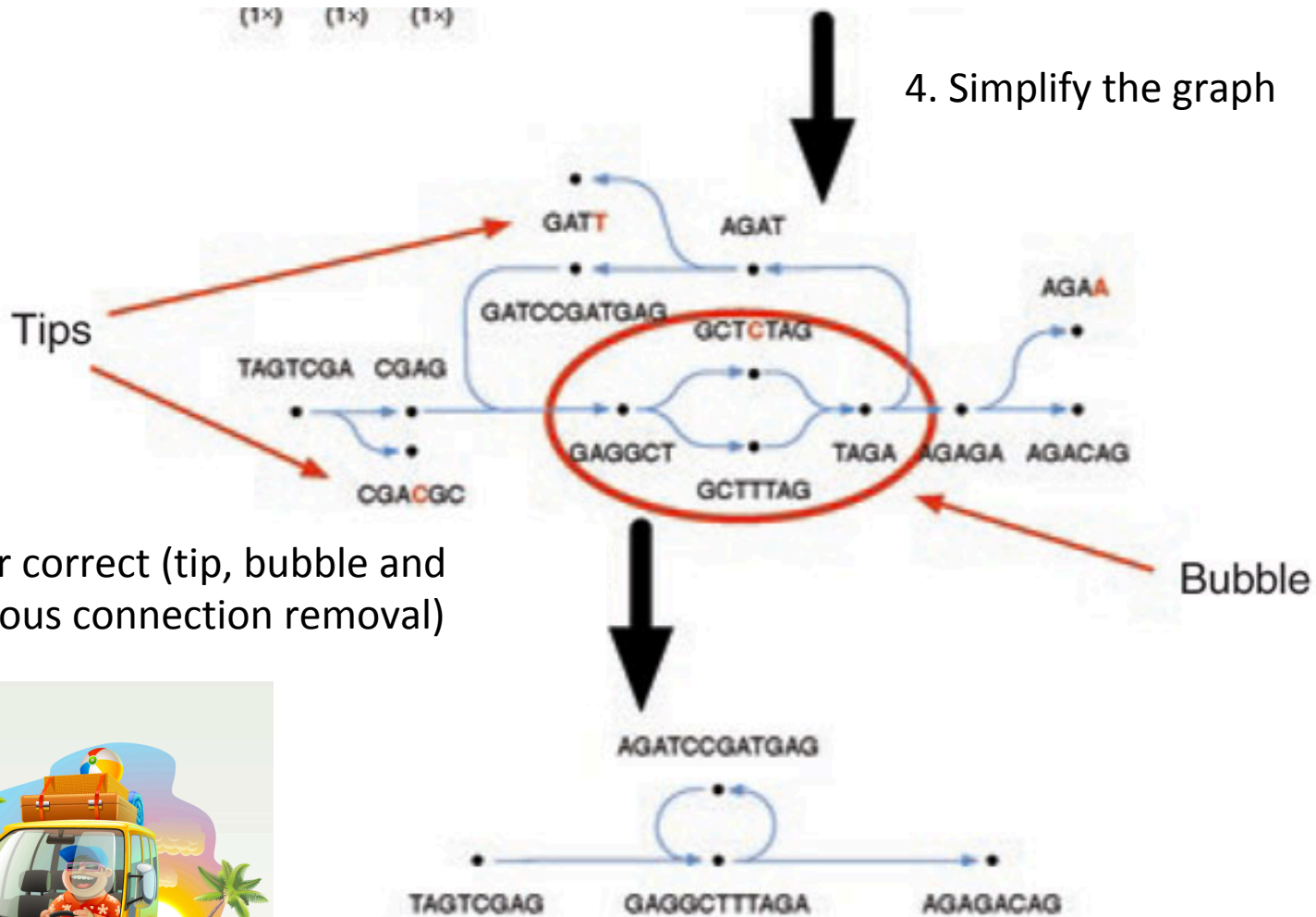
Sequencing errors

2. Find the kmers

3. Build the graph



# De Bruijn graphs



# De Bruijn graphs

Advantages:

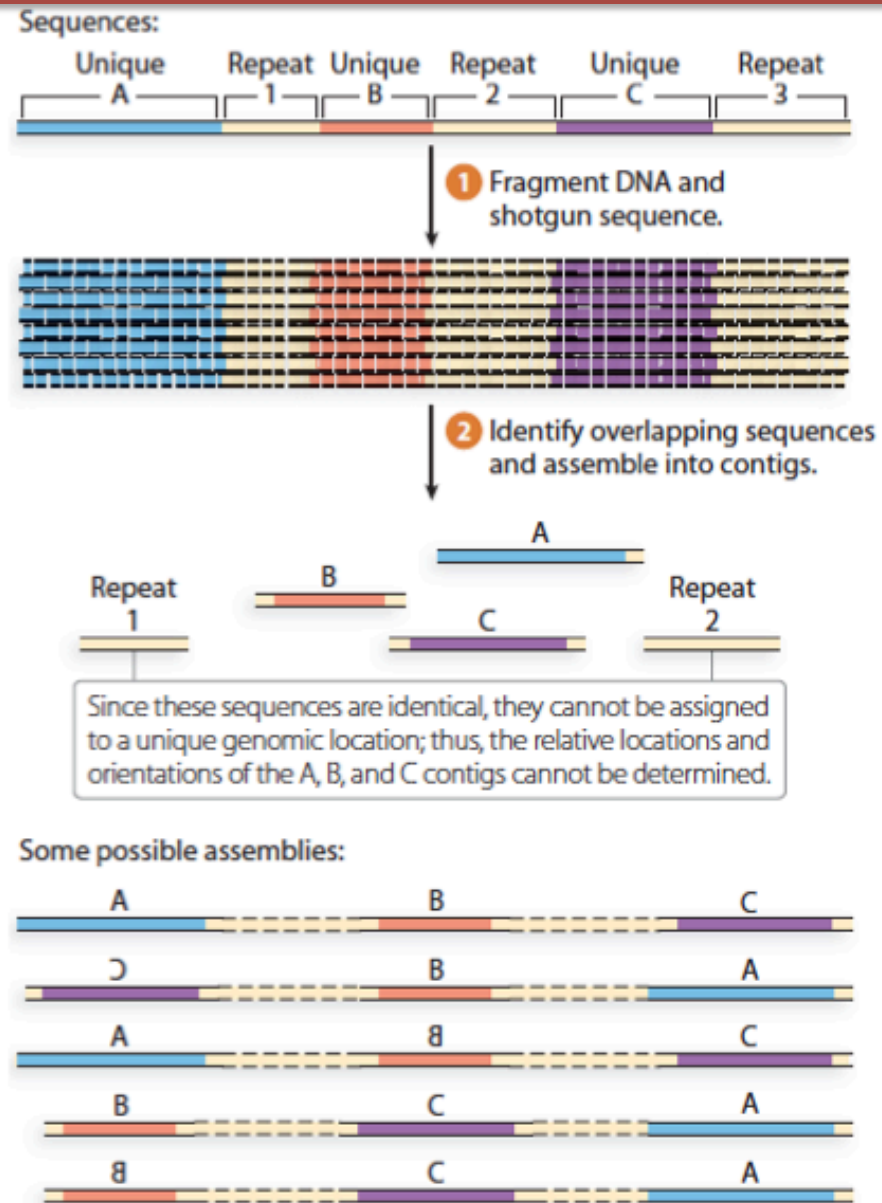
- 1) Set node length (no overlap algorithm)
- 2) Easy approaches for traversing through the graph
- 3) Simpler representation of repeats in the graph

Disadvantages:

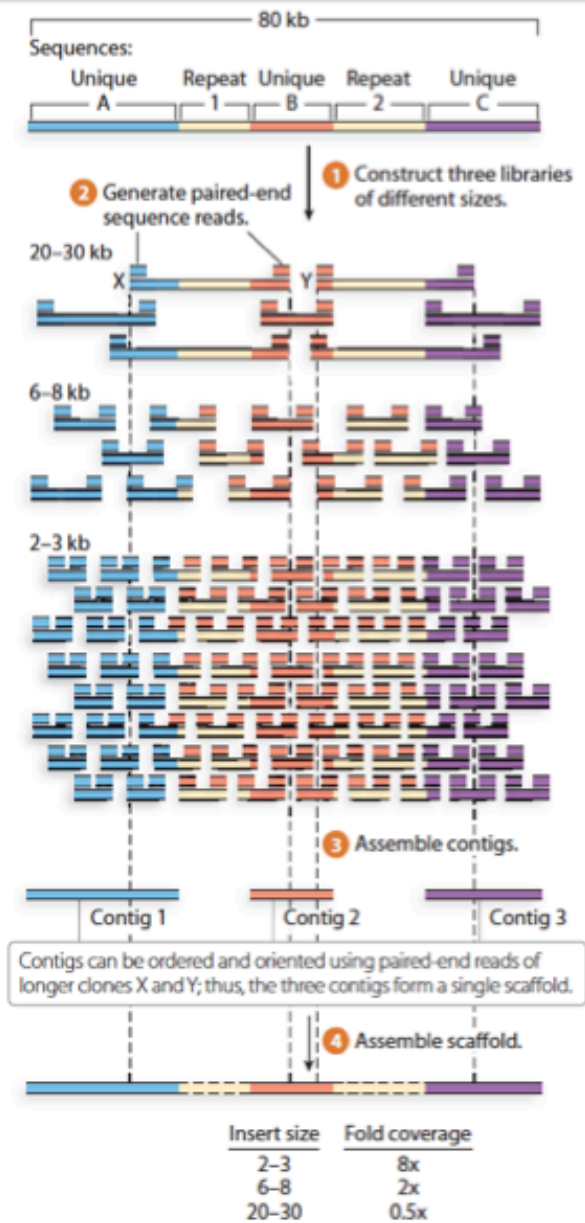
- 1) Lose information
- 2) Shorter contigs

For PacBio and other long read sequences, what type of assembly strategy would you use?

# Repetitive regions



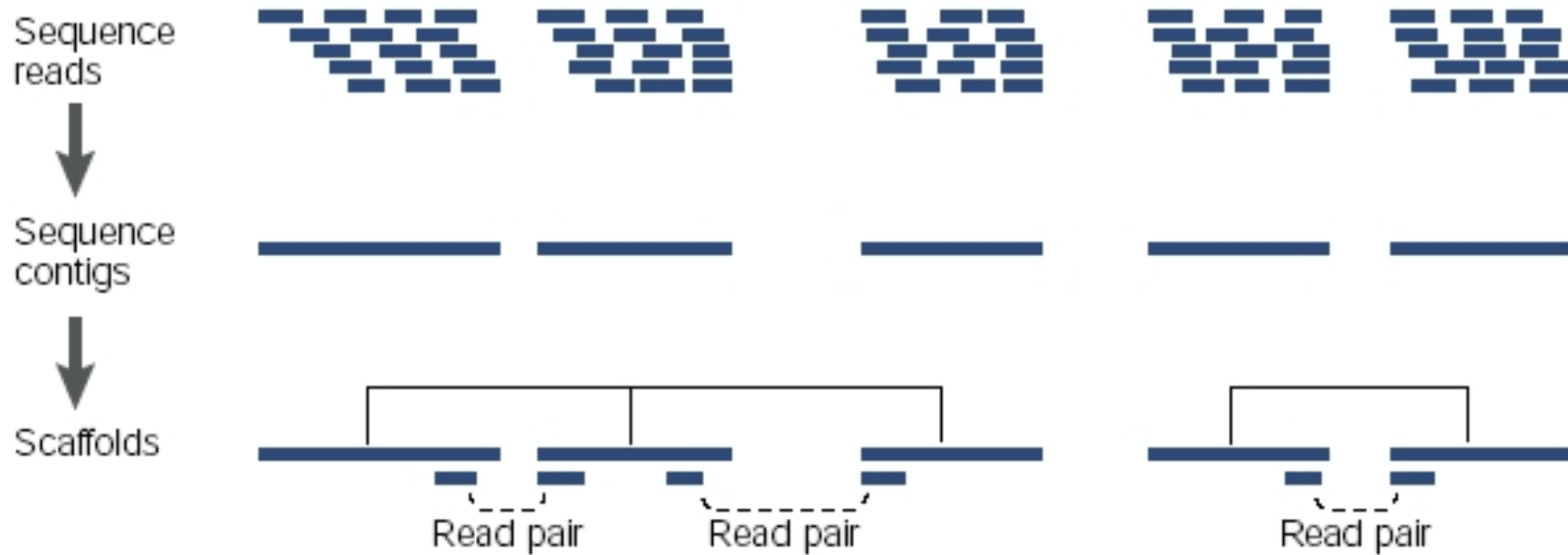
# Repetitive regions



# Finishing genomes

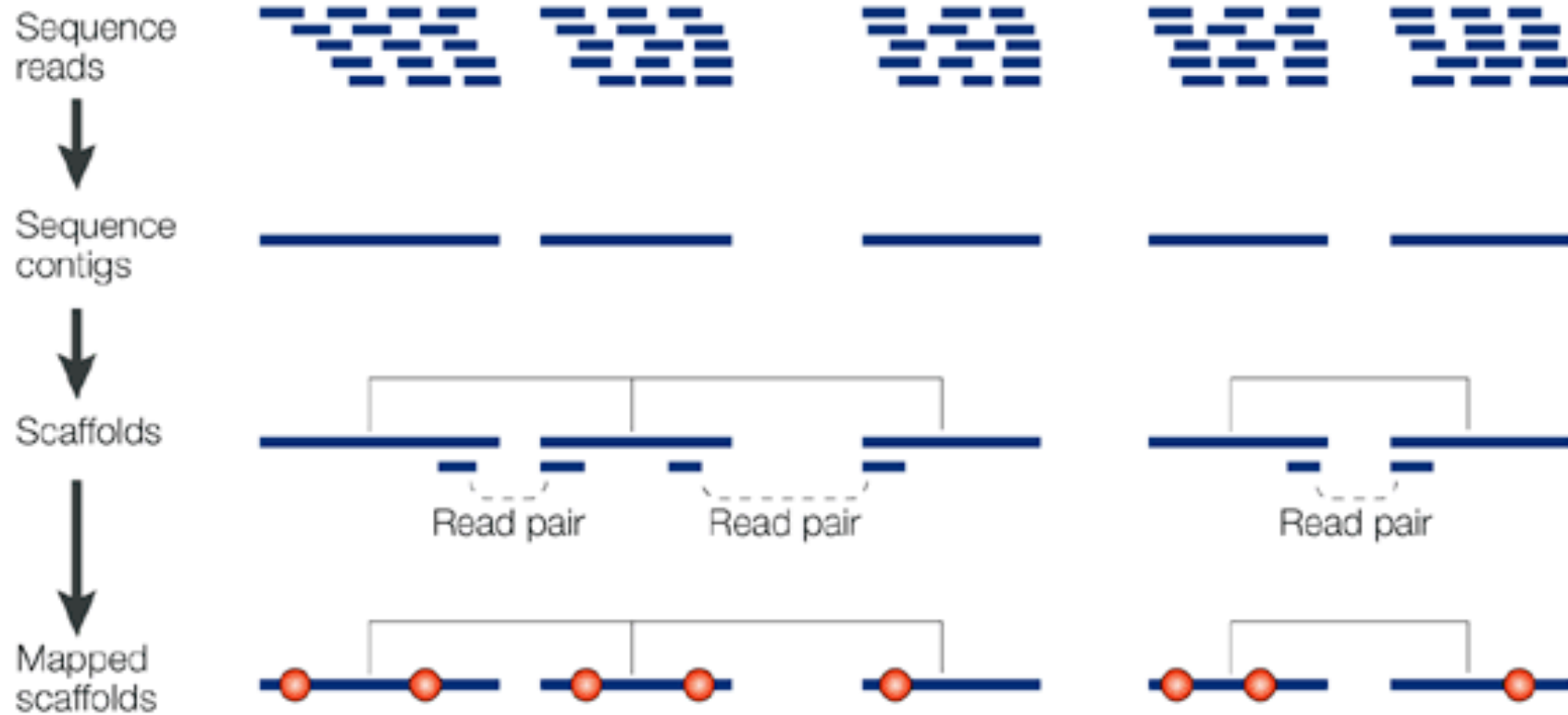
- Finishing eukaryotic genome assemblies can be challenging because much of the genome is repetitive
- This repetitive DNA breaks up the assembly and obscures the order and orientation of the assembled contigs
- Even well studied model organisms can have poorly assembled regions of their genome

# Finishing genomes





# Finishing genomes

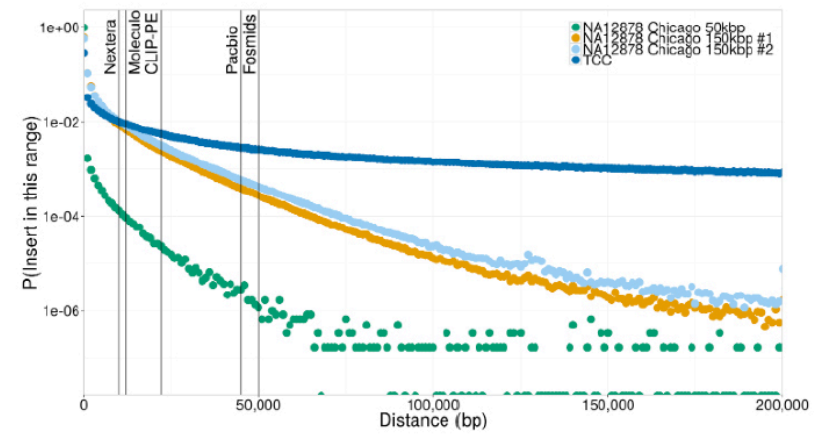
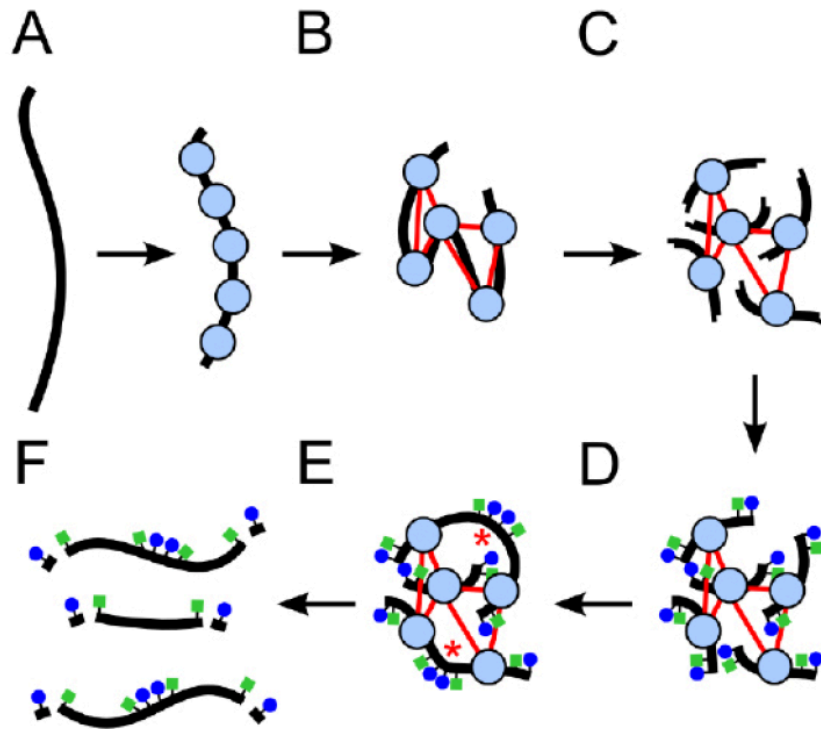


## Finishing genomes

1. Long read sequencing (e.g. PacBio)
2. Optical mapping (<https://vimeo.com/116090215>)
3. Genetic and physical maps
4. Jumping libraries

# Finishing genomes

## Ragweed genome project: Chicago library



# Finishing genomes

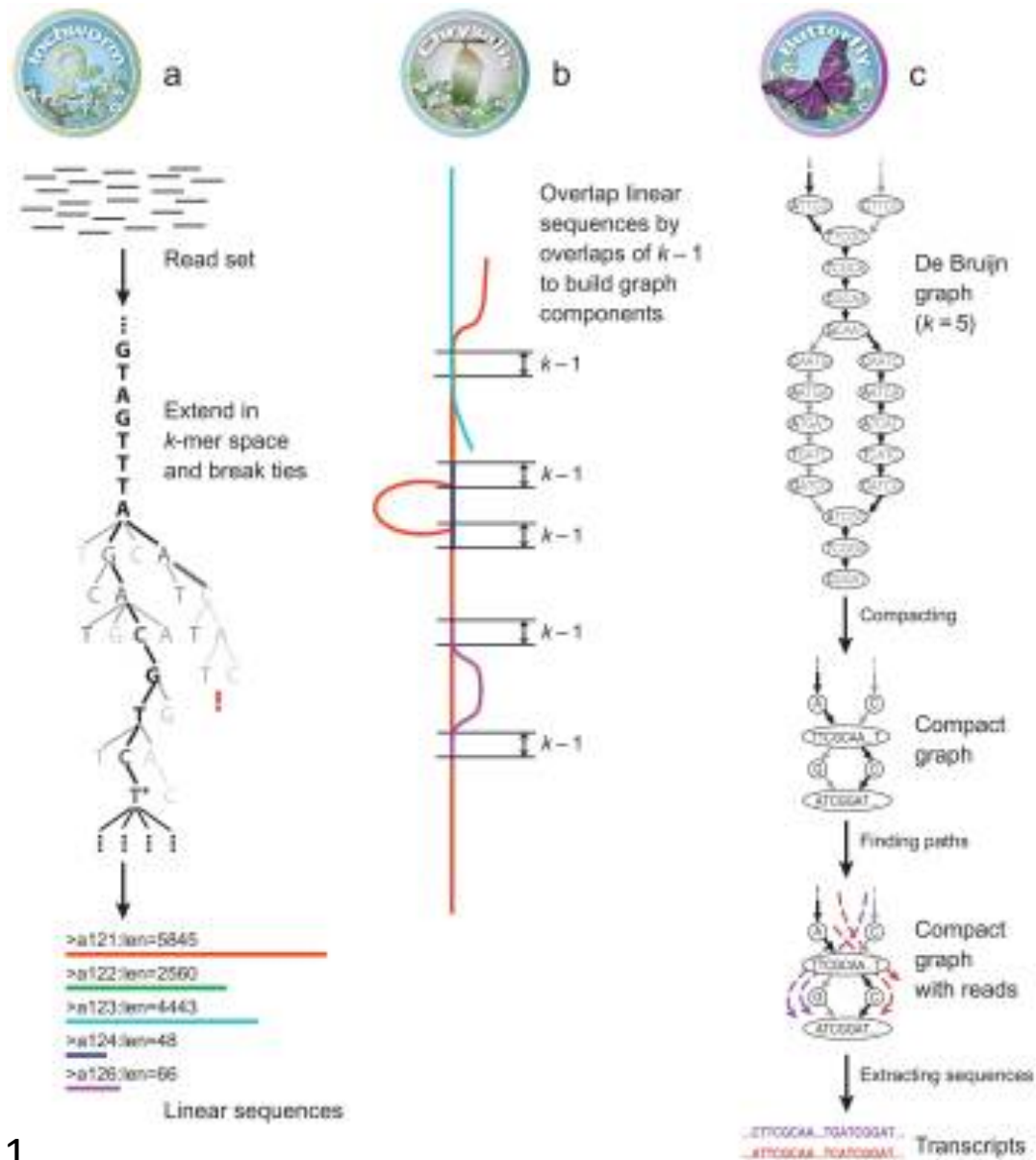
	Before HiRise	After HiRise	Fold Increase
N50	30 Kb	522 Kb	17.5X
#of Scaffolds	12,793	807	-
N90	3 Kb	88Kb	29.3X
#of Scaffolds	65,658	3,190	-

# Other types of de novo assembly

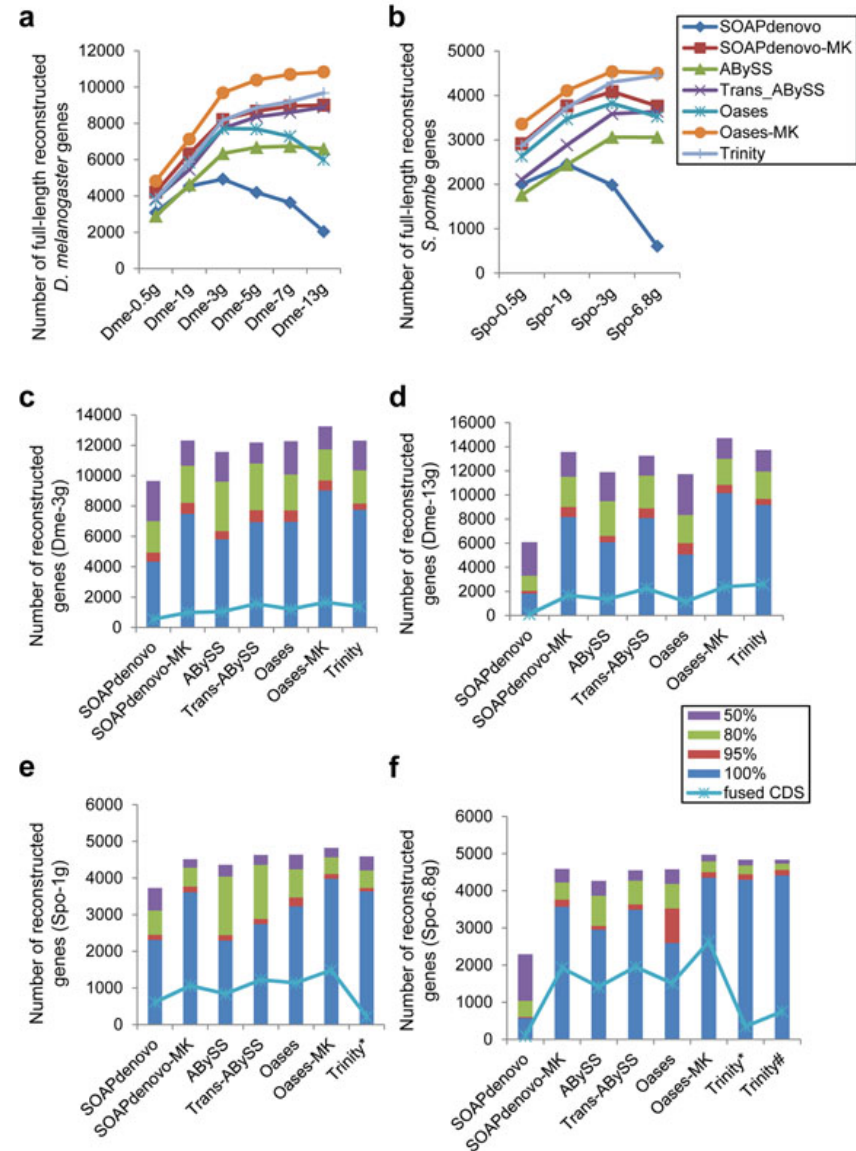
## Transcriptome

- Variable coverage among genes/isoforms
- Alternative splicing  
promoters, exons, and poly(A)

# Other types of de novo assembly

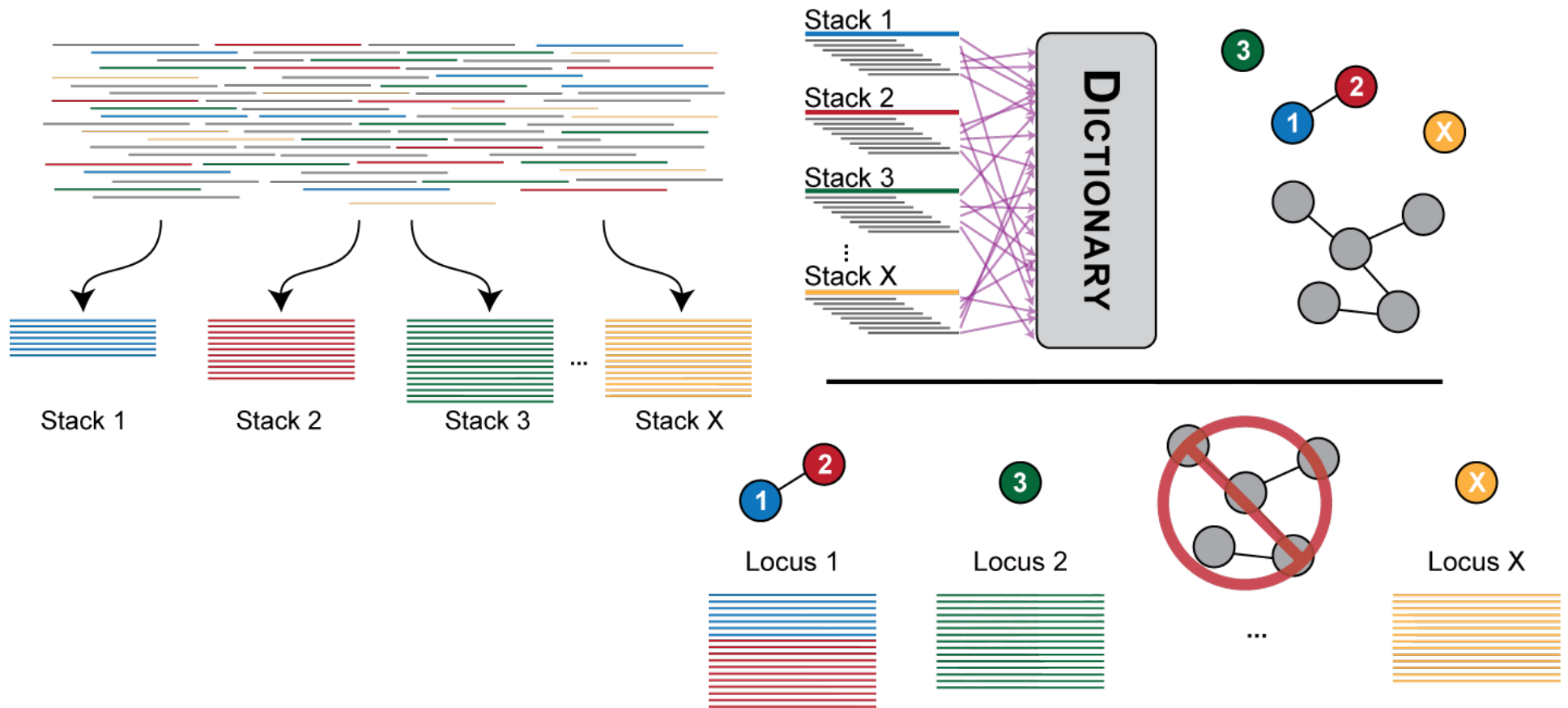


# Other types of de novo assembly



# Other types of de novo assembly

## De novo assembly of GBS reads: Stacks





# Further Reading

Flicek, P., & Birney, E. (2009). Sense from sequence reads: methods for alignment and assembly. *Nature methods*, 6, S6-S12.

Zerbino DR, Birney E. Velvet: Algorithms for de novo short read assembly using de Bruijn graphs. *Genome Research*. 2008;18(5):821-829. doi:10.1101/gr.074492.107.

<http://computing.bio.cam.ac.uk/local/doc/velvet.pdf>

Li, Z., Chen, Y., Mu, D., Yuan, J., Shi, Y., Zhang, H., ... & Yang, B. (2012). Comparison of the two major classes of assembly algorithms: overlap–layout–consensus and de-bruijn-graph. *Briefings in functional genomics*, 11(1), 25-37.

Grabherr MG, Haas BJ, Yassour M, et al. Trinity: reconstructing a full-length transcriptome without a genome from RNA-Seq data. *Nature biotechnology*. 2011;29(7):644-652. doi:10.1038/nbt.1883.

<https://github.com/trinityrnaseq/trinityrnaseq/wiki>

J. Catchen, A. Amores, P. Hohenlohe, W. Cresko, and J. Postlethwait. Stacks: building and genotyping loci de novo from short-read sequences. *G3: Genes, Genomes, Genetics*, 1:171-182, 2011.

<http://catchenlab.life.illinois.edu/stacks/>

Today you will use the genome assembly program Velvet to assemble a bacterial genome.

Velvet overview:

1. Hash k-mers
2. Construct the graph
3. Correct for errors
4. Resolve the repeats

Refer to Github page or open `/home/biol525d/Topic_5/README_assembly.txt` and follow the instructions

- 1) Given the above information, what is the expected coverage?
- 2) For a k-mer of 21 what is the k-mer coverage for this genome assembly?
- 3) Can you think of other ways to assess assembly quality? What might be the trouble with only focusing on maximizing N50? Discuss this with your group.
- 4) Quantify the assembly metrics for your first assembly that you ran without any options. In your group of four, each person should pick different sets of parameters to run. Compare the resulting assemblies with one another and discuss which ones seemed to have improved the assembly and why that might be. Be prepared to share your findings with the class.

## For next class

- Make sure R and Rstudio are installed and working on your computer
- Go over Greg's short R tutorial (Topic 2) if you are not familiar with R

1. **ABYSS (Assembly By Short Sequencing) (Biroi et al)**: A *de novo* assembler for short read sequence data which uses a distributed representation of a de Bruijn graph, allowing parallel computation of the assembly algorithm across a network of commodity computers. Developed at Canada's Michael Smith Genome Sciences Centre.
2. **ALLPATHS-LG (Gnerre et al)**: a de Bruijn graph-based *de novo* assembler for large (and small) genomes. ALLPATHS-LG is being developed by scientists at the Broad Institute.
3. **Bambus2**: The second generation Bambus scaffolder relies on a combination of a novel method for detecting genomic repeats and algorithms that analyze assembly graphs to identify biologically meaningful genomic variants. Bambus2 compares favorably to existing scaffolds generated by CABOG, Newbler and SOAPdenovo with respect to contiguity and error rate. While Bambus 2 was specifically designed for polymorphic and metagenomic scaffolding, its modular and efficient algorithm allows it to be used to scaffold mammalian genomes and used a drop-in replacement scaffolder for CABOG, Newbler, and SOAPdenovo. Bambus2 is being primarily developed by [Sergey Koren](#) and [Mihai Pop](#), with input from [Todd Treangen](#),
4. **Celera Assembler**: an Overlap-Layout-Consensus based *de novo* whole-genome shotgun (WGS) DNA sequence assembler. It reconstructs long sequences of genomic DNA from fragmentary data produced by whole-genome shotgun sequencing. Celera Assembler has enabled many advances in genomics, including the first whole genome shotgun sequence of a multi-cellular organism (Myers 2000) and the first diploid sequence of an individual human (Levy 2007). Celera Assembler was developed at Celera Genomics starting in 1999. It was released to SourceForge in 2004 as the wgs-assembler under the GNU General Public License. The pipeline revised for 454 data was named CABOG (Miller 2008).
5. **MSR-CA** (pronounced "MizerKa") is a new technique that pre-processes the short read data and then performs the final assembly using a modified version of Celera Assembler. MSR-CA stands for Maryland Super-Reads + Celera Assembler. The pre-processing steps include error correction and subsequent coverage reduction by creating "super-reads," which are produced using a de Bruijn graph. The algorithm then groups together the reads that map to the same sets of nodes and edges, and for each set replaces them by a single super-read that contains these nodes and edges. This can reduce the number of reads by a factor of 50 or more, resulting in the data set that is much easier to manage.
6. **SGA (Simpson et al)**: stands for String Graph Assembler. Experimental *de novo* assembler based on string graphs. SGA is being developed by scientists at the Wellcome Trust Sanger Institute.
7. **SOAPdenovo (Li et al)**: is the short-read assembler that was used for the panda genome, the first mammalian genome assembled entirely from Illumina reads, and for several human genomes and other genomes subsequently. It is being developed by scientists at BGI.
8. **Velvet (Zerbino et al)**: Velvet is a *de novo* genome assembler specially designed for short read sequencing technologies, particularly Illumina reads, and was one of the first short-read assemblers to be published. It was developed by Daniel Zerbino and Ewan Birney at the European Bioinformatics Institute (EMBL-EBI), near Cambridge, England.

Assembler	Contigs				Scaffolds			
	Num	N50 (kb)	Errors	N50 corr. (kb)	Num	N50 (kb)	Errors	N50 corr. (kb)
ABySS	302	29.2	19	24.8	246	34	1	28
Allpaths-LG	<b>60</b>	96.7	20	<b>66.2</b>	<b>12</b>	1,092	<b>0</b>	<b>1,092</b>
Bambus2	109	50.2	190	16.7	17	1,084	<b>0</b>	1,084
CABOG	Could not run: incompatible read lengths in one library							
MSR-CA	94	59.2	34	48.2	17	<b>2,412</b>	3	1,022
SGA	1252	4.0	<b>10</b>	4.0	456	208	1	208
SOAPdenovo	107	<b>288.2</b>	65	62.7	99	332	<b>0</b>	288
Velvet	162	48.4	42	41.5	45	762	17	126