Genome Assembly Introduction

Sebastian Schmeier
s.schmeier@gmail.com
http://sschmeier.github.io/bioinf-workshop/

10.08.2015

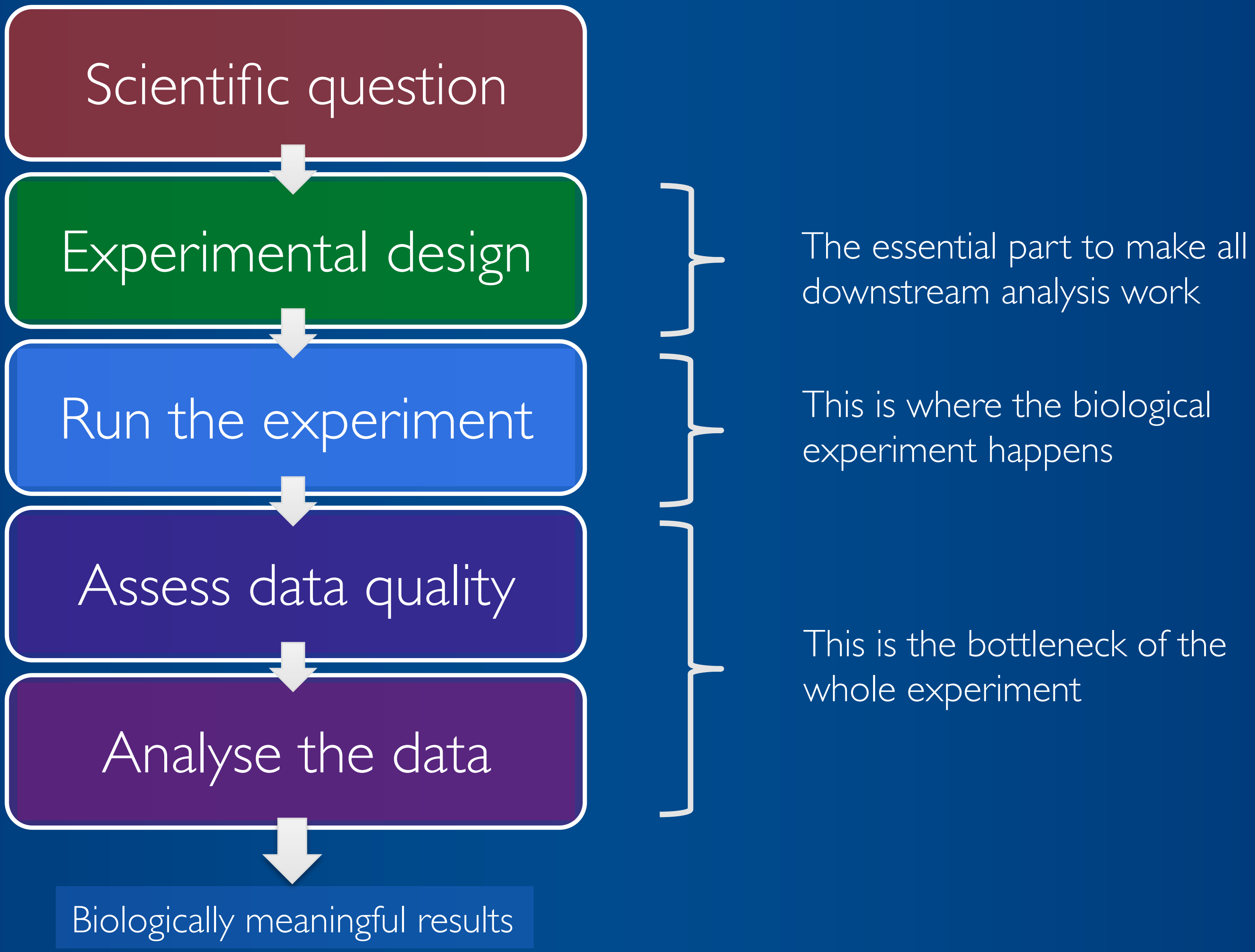
Topic overview

- Genome assembly
  - Learning outcomes:
    - Being able to compute, investigate and evaluate the quality of sequence data from a sequencing experiment.
    - Be able to describe the concepts regarding genome assembly.
    - Be able to compute and investigate a whole genome assembly.
    - Being able to interpret and judge the quality of a genome assembly.

Overview

- DNA sequencing technologies
- Quality assessment of a sequencing run
- Genome assemblies and *de Bruijn* graphs

Typical workflow of a genomics experiment


Scientific question
Experimental design
Run the experiment
Assess data quality
Analyse the data
Biologically meaningful results
The essential part to make all downstream analysis work
This is where the biological experiment happens
This is the bottleneck of the whole experiment

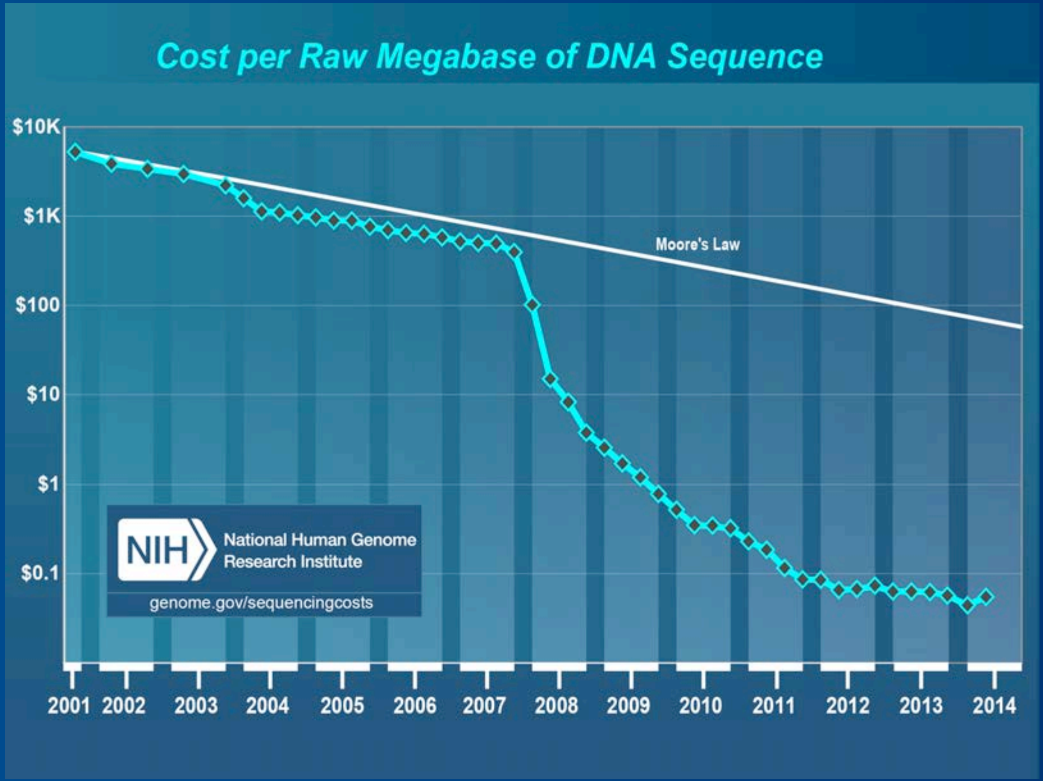
Genome versus transcriptome

- Genome
  - The entirety of an organism's ancestral information. It is encoded either in DNA or, for many types of viruses, in RNA.
- Transcriptome
  - The set of all RNA molecules, including messenger RNA, ribosomal RNA, transfer RNA, and other non-coding RNA produced in one or a population of cells

Name	Base Pairs	
HIV	9,749	9.7kb
E.Coli	4,600,000	4.6MB
Yeast	12,100,000	12.1Mb
Drosophila	130,000,000	130MB
Homo sapiens	**3,200,000,000**	**3.2GB**
marbled lungfish	130,000,000,000	130Gb
"Amoeba" dubia	670,000,000,000	670Gb

disputed

DNA sequencing



- **DNA sequencing** is the process of determining the nucleotide order of a given DNA fragment.
  - **First-generation sequencing**:
    - 1977 Sanger sequencing method development (chain-termination method)
    - *2001*, Sanger method produced a draft sequence of the human genome
  - **Next-generation sequencing** (NGS)
    - Demand for low-cost sequencing has driven the development of high-throughput sequencing (or NGS) technologies that parallelize the sequencing process, producing thousands or millions of sequences concurrently
    - *2004* 454 Life Sciences marketed a parallelized version of pyrosequencing

Result of a sequencing run

- Short read sequences
  - The result of NGS technology are a collection of short nucleotide sequences (reads) of varying length (~40-400nt) depending on the technology used to generate the reads
  - Usually a reads quality is good at the beginning of the read and errors accumulate the longer the read gets → IMPORTANT

Illumina sequencing

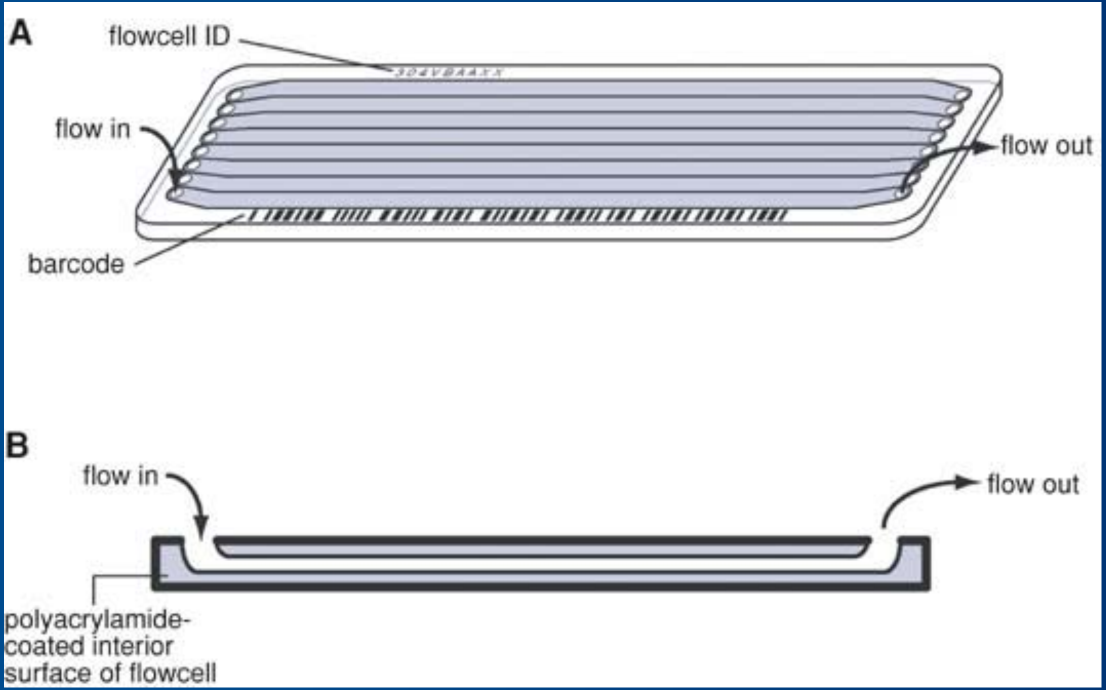
- MiSeq:
  - Bench-top sequencer
  - Produces around 30 million reads/run
  - Reads are up to 250nt
- HiSeq:
  - Large-scale sequencer
  - 4 billion reads/run
  - Reads up to 150nt
- The Illumina systems accumulate errors towards the end of the read sequence.

http://www.illumina.com/

Illumina sequencing

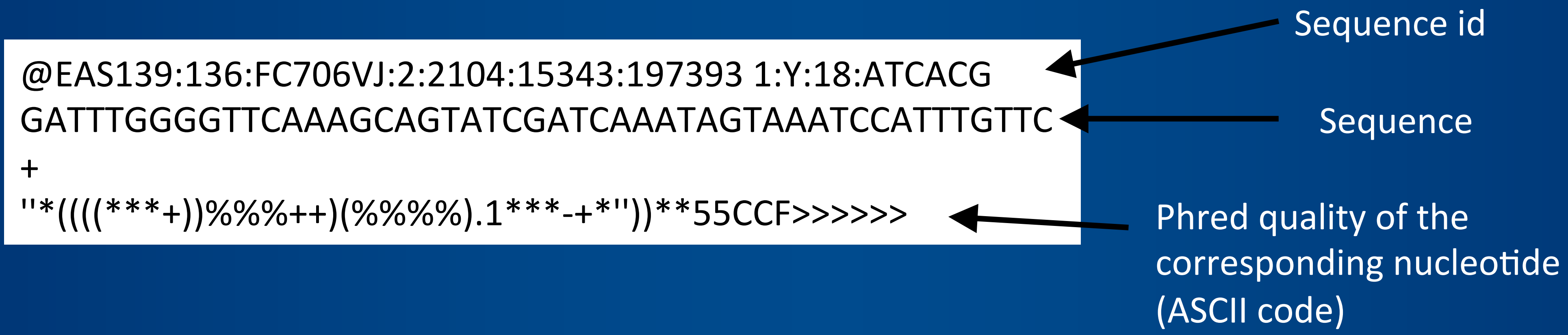
- An Illumina flowcell is a surface to which seq. adaptors are covalently attached.
- DNA with complementary adaptors is attached, clonally amplified, and then sequenced by synthesis
- Each flowcell is subdivided into hundreds of tiles

http://www.illumina.com/



How does the output look like?

- The file-format that you will encounter soon is called FastQ



```
@EAS139:136:FC706VJ:2:2104:15343:197393 1:Y:18:ATCACG
GATTTGGGGTTCAAAGCAGTATCGATCAAATAGTAAATCCATTTGTTC
+
''*((((***+))%%%++)(%%%%).1***-+*''))**55CCF>>>>>>
```

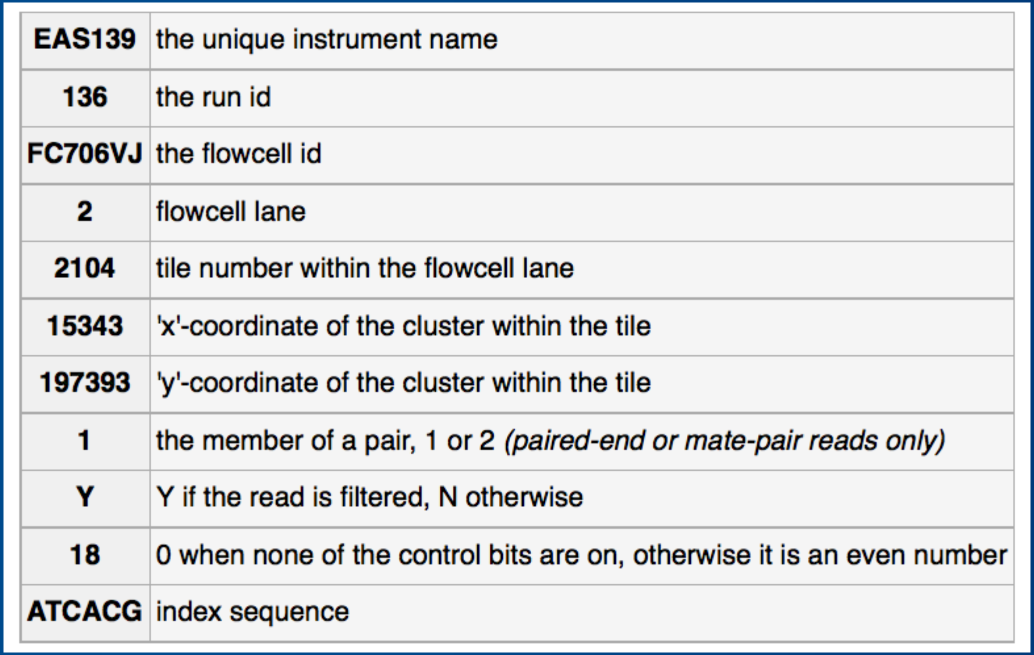
How does the output look like?

- FastQ: Identifier

Sequence id

```
@EAS139:136:FC706VJ:2:2104:15343:197393 1:Y:18:ATCACG
GATTTGGGGTTCAAAGCAGTATCGATCAAATAGTAAATCCATTTGTTC
+
''*((((***+))%%%++)(%%%%).1***-+*''))**55CCF>>>>>>
```

Casava 1.8 the format



EAS139	the unique instrument name
136	the run id
FC706VJ	the flowcell id
2	flowcell lane
2104	tile number within the flowcell lane
15343	'x'-coordinate of the cluster within the tile
197393	'y'-coordinate of the cluster within the tile
1	the member of a pair, 1 or 2 *(paired-end or mate-pair reads only)*
Y	Y if the read is filtered, N otherwise
18	0 when none of the control bits are on, otherwise it is an even number
ATCACG	index sequence

How does the output look like?

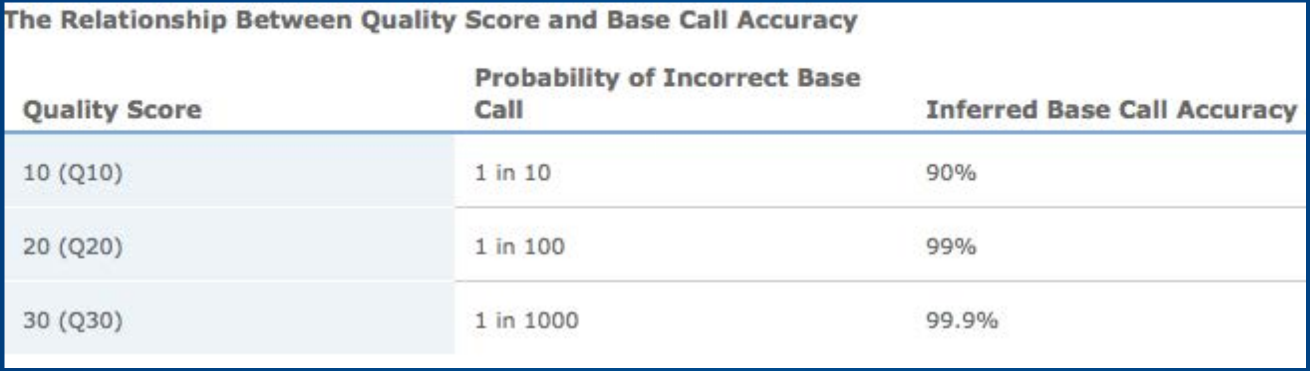
- FastQ: Phred base quality

```
@EAS139:136:FC706VJ:2:2104:15343:197393 1:Y:18:ATCACG
GATTTGGGGTTCAAAGCAGTATCGATCAAATAGTAAATCCATTTGTTC
+
"*((((***+))%%%++)(%%%%).1***-+*"))**55CCF>>>>>>
```

Phred quality of the corresponding nucleotide (ASCII code)

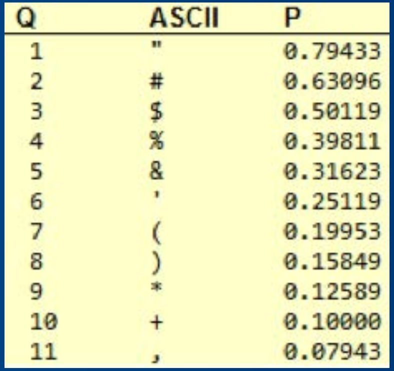
- One ASCII character per nucletide.
- Encodes for a quality Q = -10*$\log_{10}$(P), where P is the error probability

-10*$\log_{10}(0.1)$ =



The Relationship Between Quality Score and Base Call Accuracy

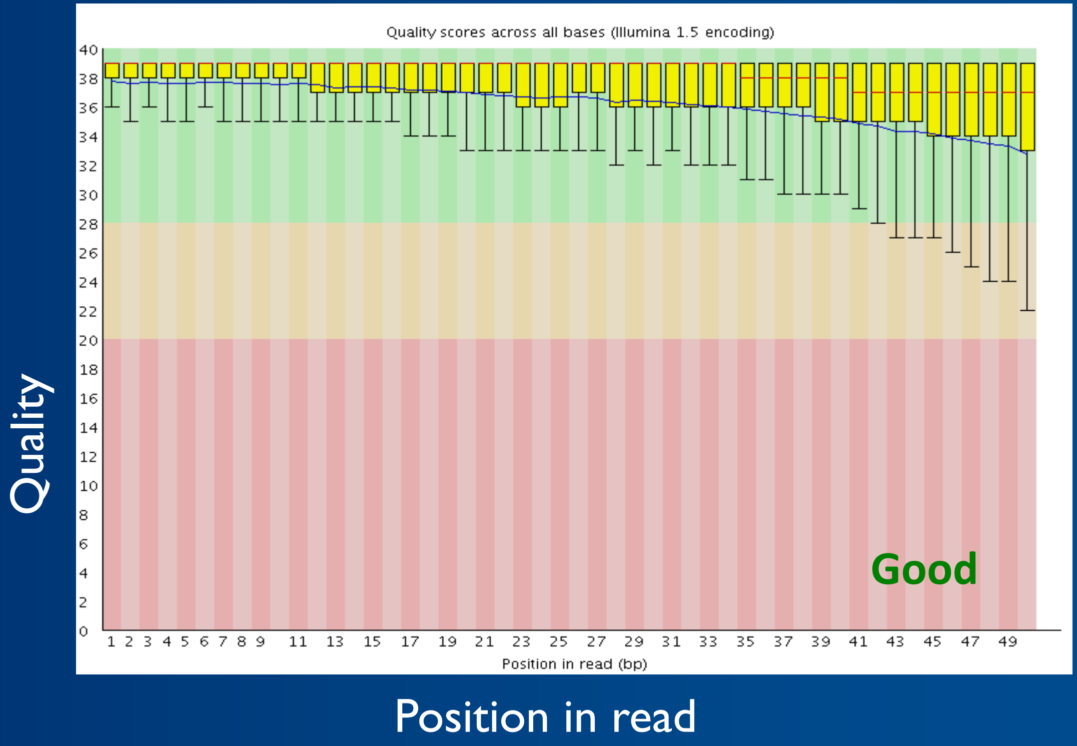
Quality Score	Probability of Incorrect Base Call	Inferred Base Call Accuracy
10 (Q10)	1 in 10	90%
20 (Q20)	1 in 100	99%
30 (Q30)	1 in 1000	99.9%

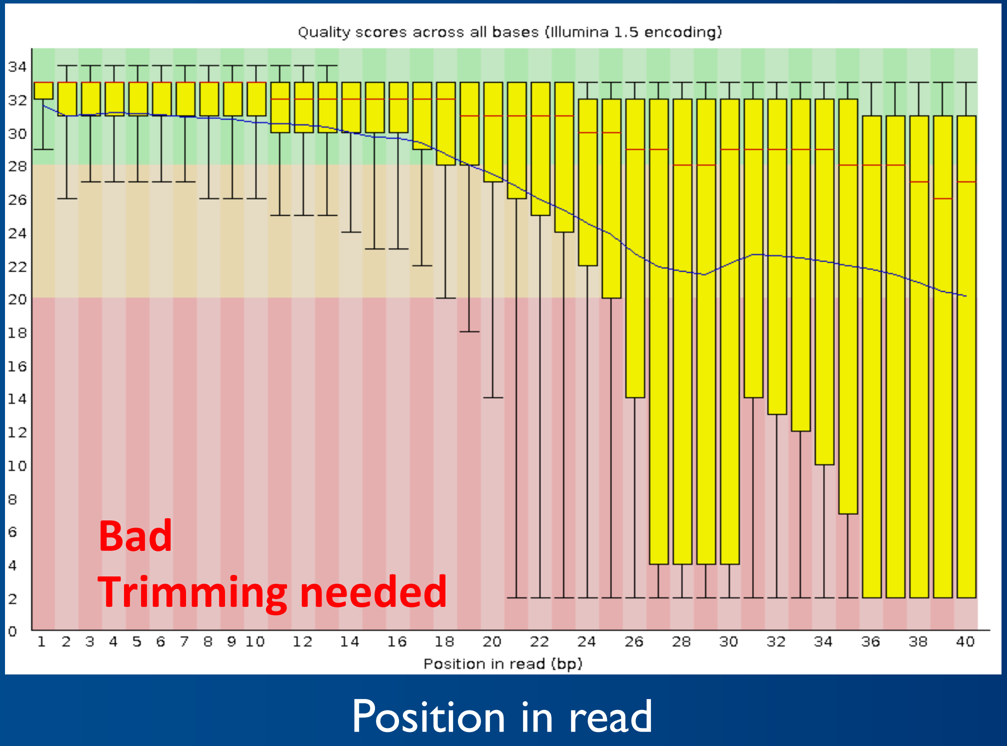


Q	ASCII	P
1	"	0.79433
2	#	0.63096
3	$	0.50119
4	%	0.39811
5	&	0.31623
6	'	0.25119
7	(	0.19953
8	)	0.15849
9	*	0.12589
10	+	0.10000
11	,	0.07943

Assessing quality: Reads

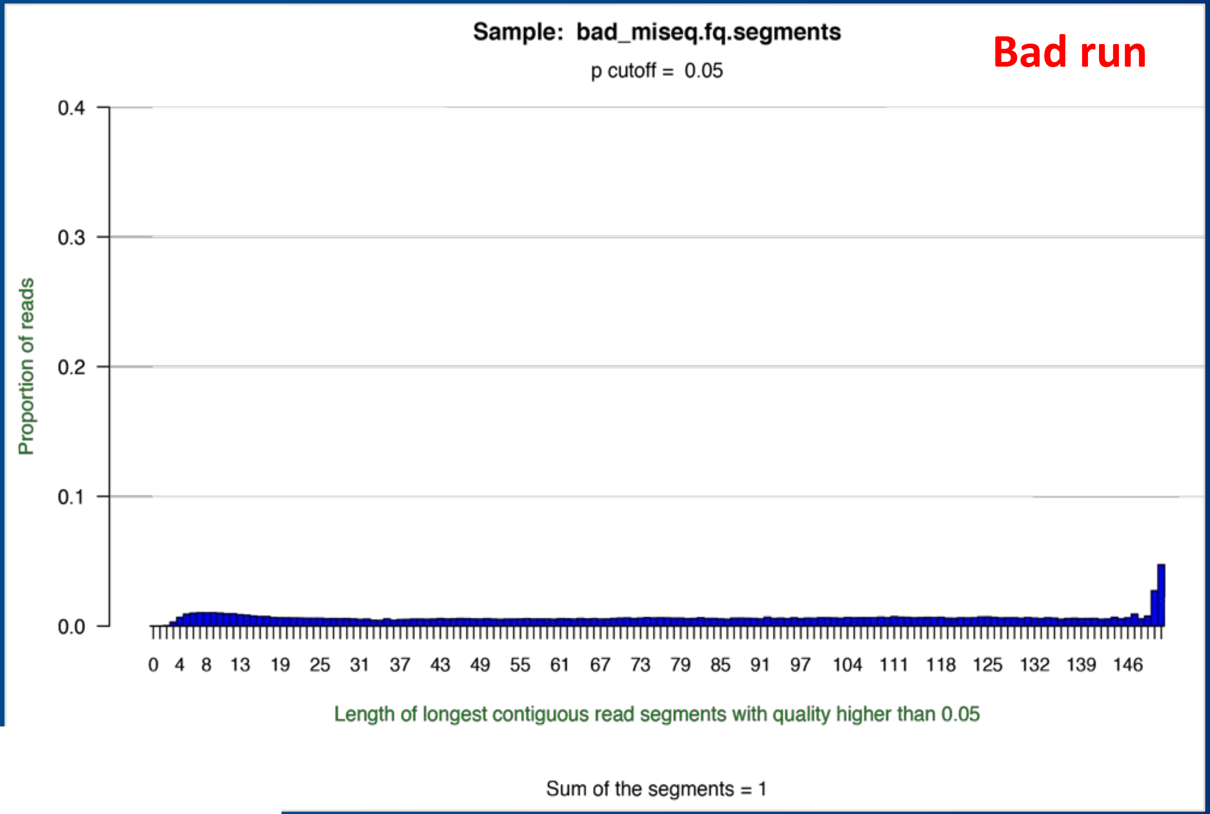
http://www.bioinformatics.babraham.ac.uk/projects/fastqc/

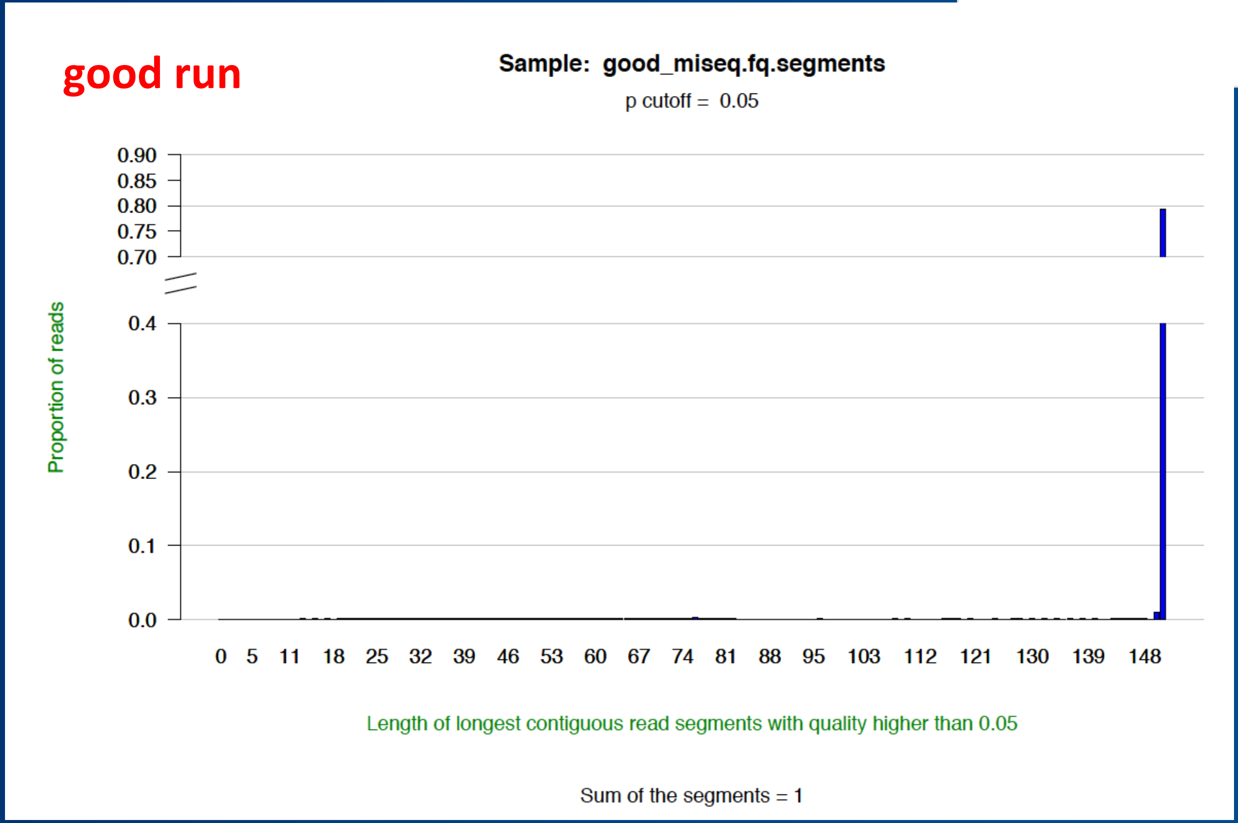




- If the quality of the reads is bad we can trim the nucleotides that are bad of the end of the reads
- Not trimming the end has a huge influence on downstream processes, e.g. assemblies

Assessing quality: Reads





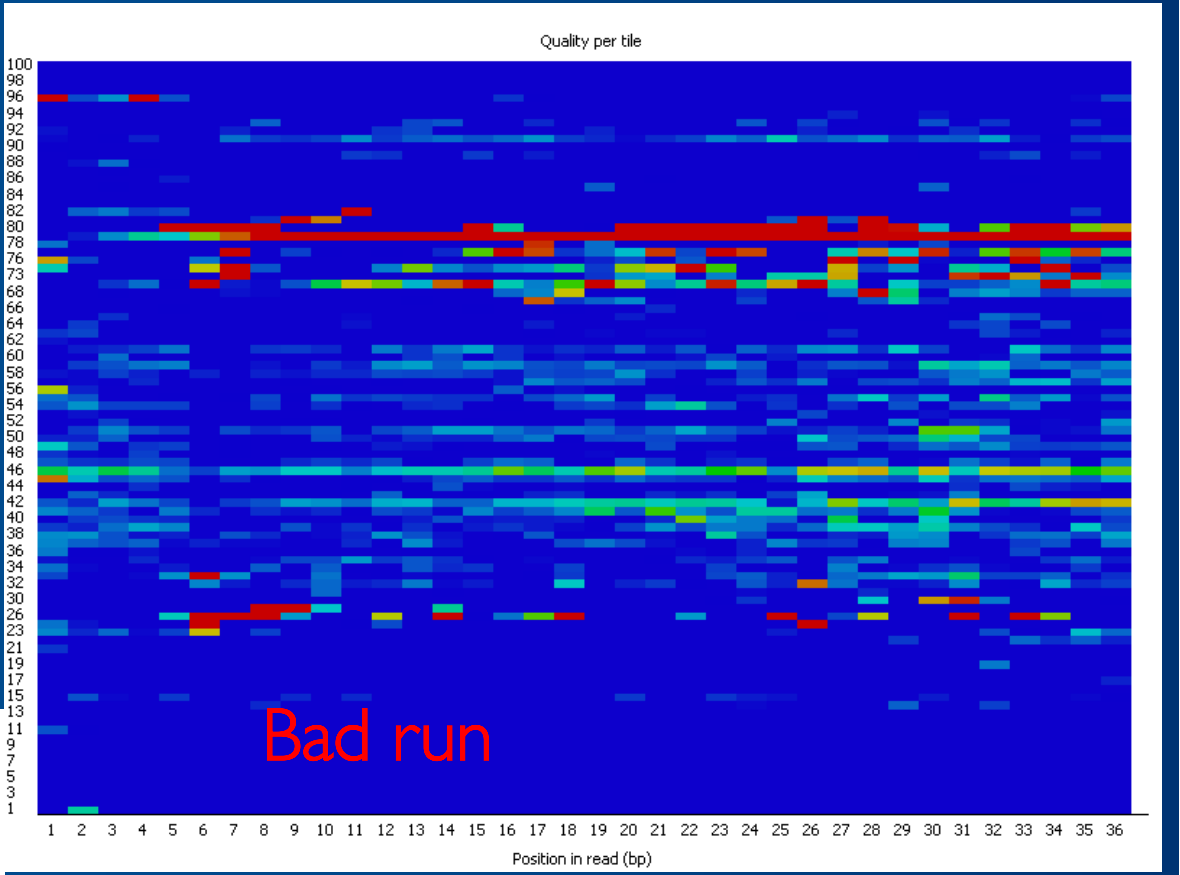
http://solexaqa.sourceforge.net/

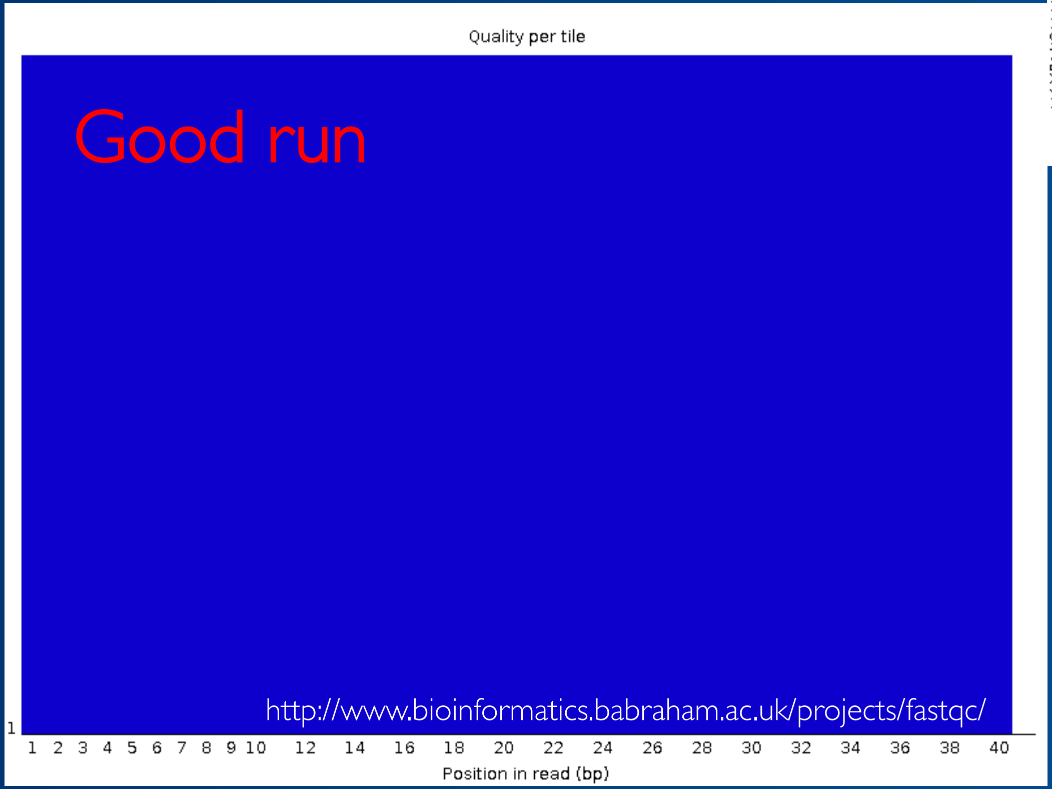
Assessing quality: Tiles

- One can assess also the quality of a run based on the tiles of a lane of a flowcell
  - → spot problems with a particular tile on a lane, e.g. Bubbles in the reagents
- The homogeneity of the Illumina process ensures that the relative frequencies are similar from tile to tile and distributed uniformly across each tile
  - → when the machine is functioning properly
- Major discrepancies in these conditions can be discerned by sight
- Many such discrepancies are small and their effects are limited to one, or a few, tiles.

Assessing quality: Tiles

- Encoded in these is the flowcell tile from which each read came.
- The graph allows you to look at the quality scores from each tile across all of your bases to see if there was a loss in quality associated with only one part of the flowcell.



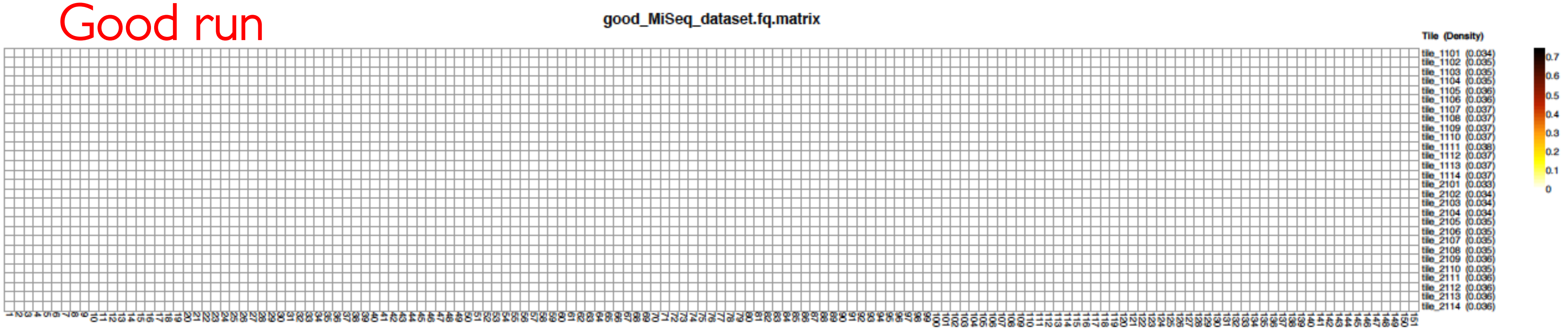


- The plot shows the deviation from the average quality for each tile.
- The colours are on a cold to hot scale
- Cold colours being positions where the quality was at or below the average for that base in the run
- Hotter colours indicate that a tile had worse qualities than other tiles for that base.

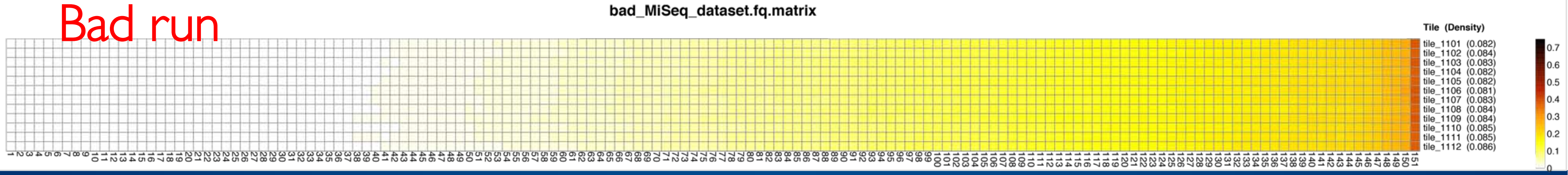
Assessing quality: Tiles

http://www.bioinformatics.babraham.ac.uk/projects/fastqc/

Good run



Bad run



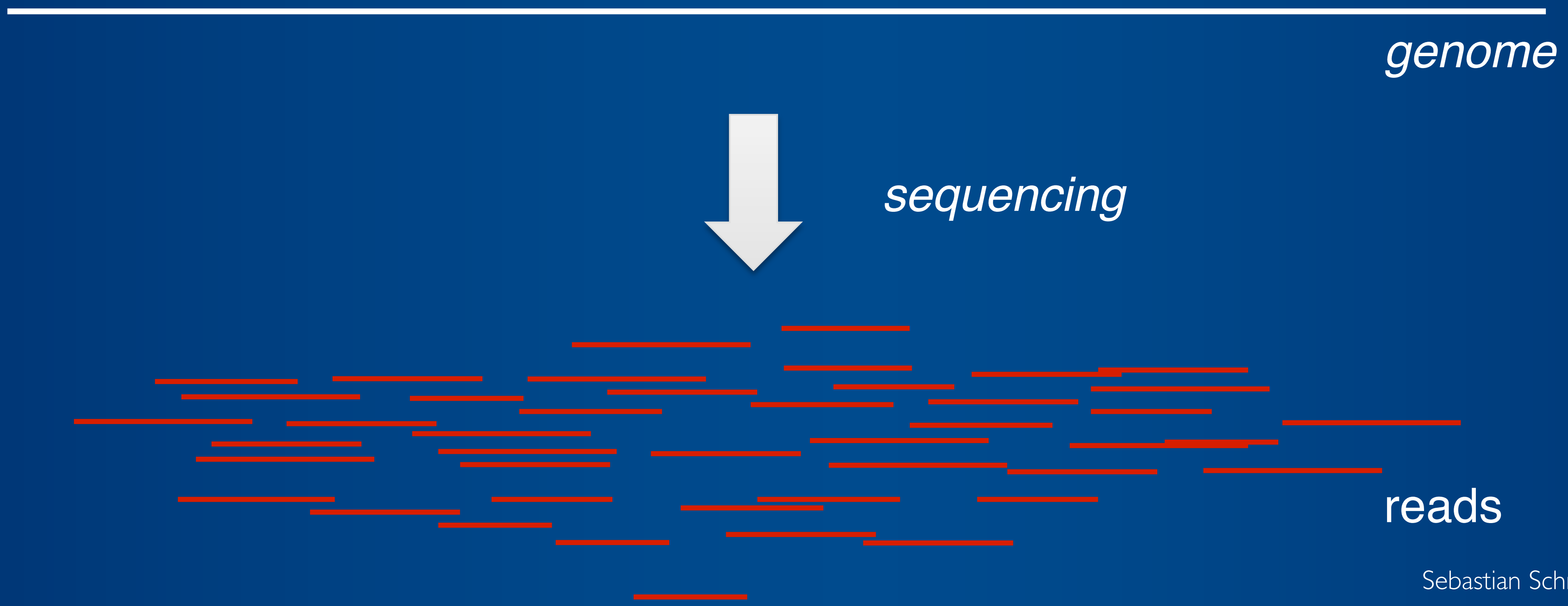
Position in read

Assessing quality: Final

- After assessing the quality we would try to remove all bp from the ends that do not fulfil a certain quality
- Thus, we work with a adjusted set of sequencing reads for which we are more certain that they represent correct nt sequences from the genome

De novo genome assembly

- The process of generating a new genome sequence from NGS genome sequence reads based on assembly algorithms
- Assembly involves joining short sequence fragments together into long pieces – contigs

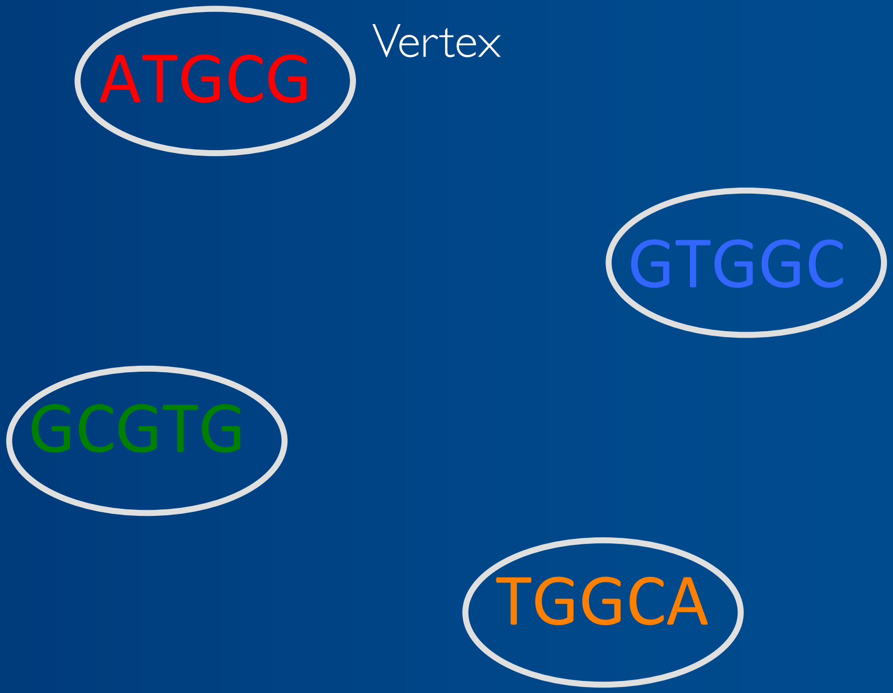


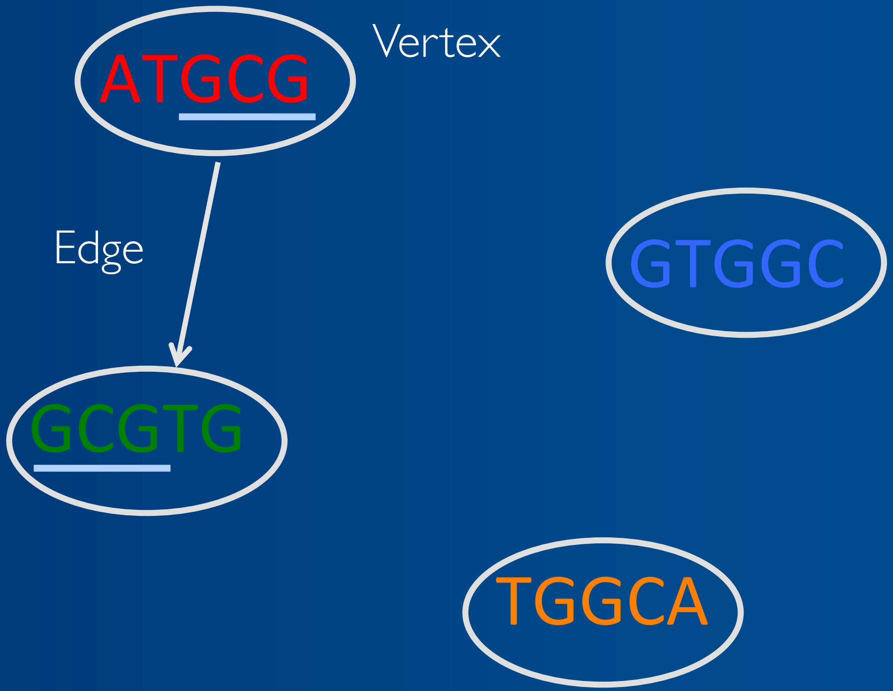
ATGCG

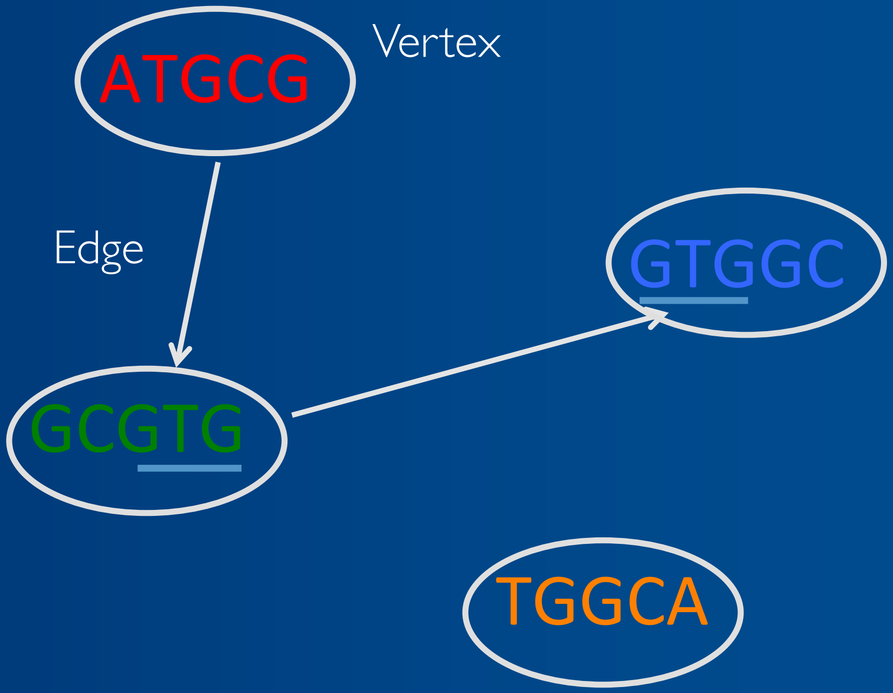
GTGGC

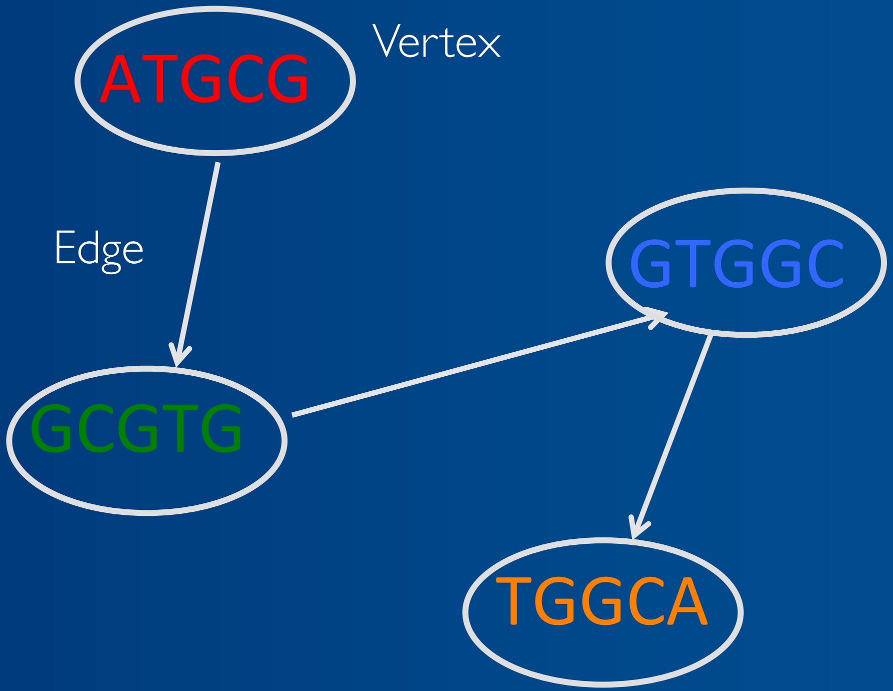
GCGTG

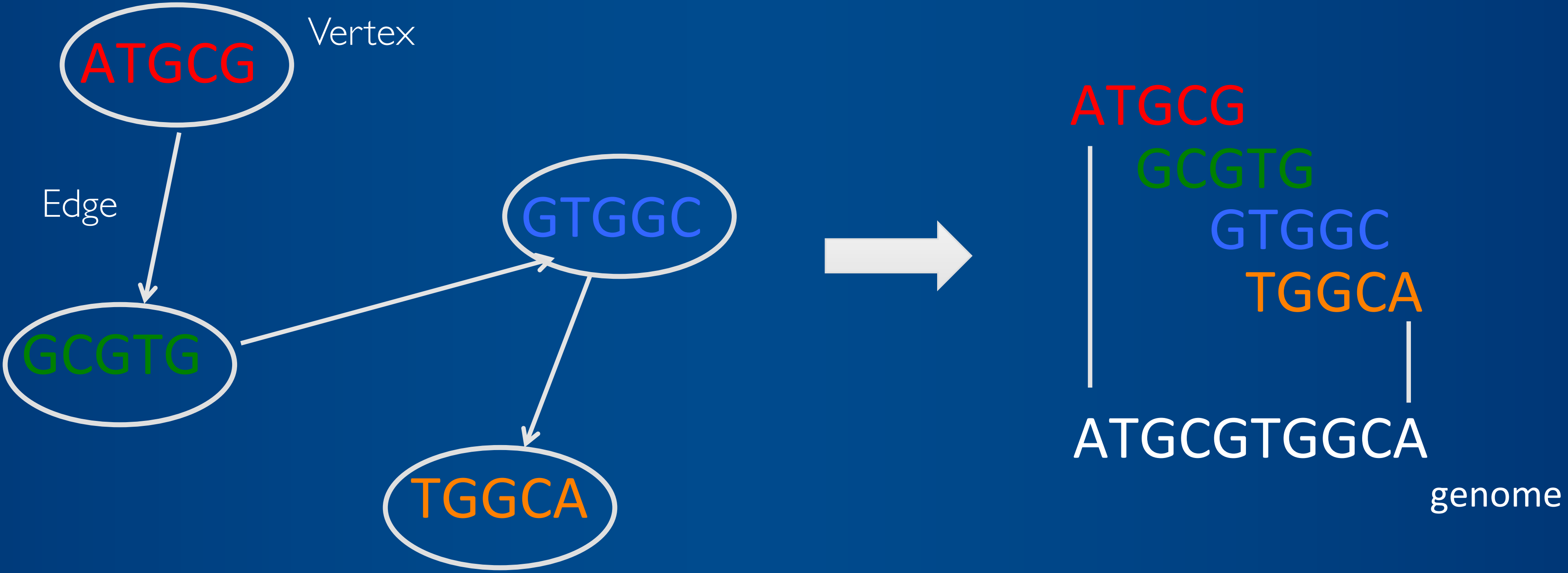
TGGCA


ATGCG
Vertex
GTGGC
GCGTG
TGGCA


ATGCG
Vertex
Edge
GCGTG
GTGGC
TGGCA


Vertex
ATGCG
Edge
GTGGC
GCGTG
TGGCA


Vertex
Edge
ATGCG
GCGTG
GTGGC
TGGCA


Vertex
ATGCG
Edge
GTGGC
GCGTG
TGGCA
ATGCG
GCGTG
GTGGC
TGGCA
ATGCGTGGCA
genome

The fragment assembly problem

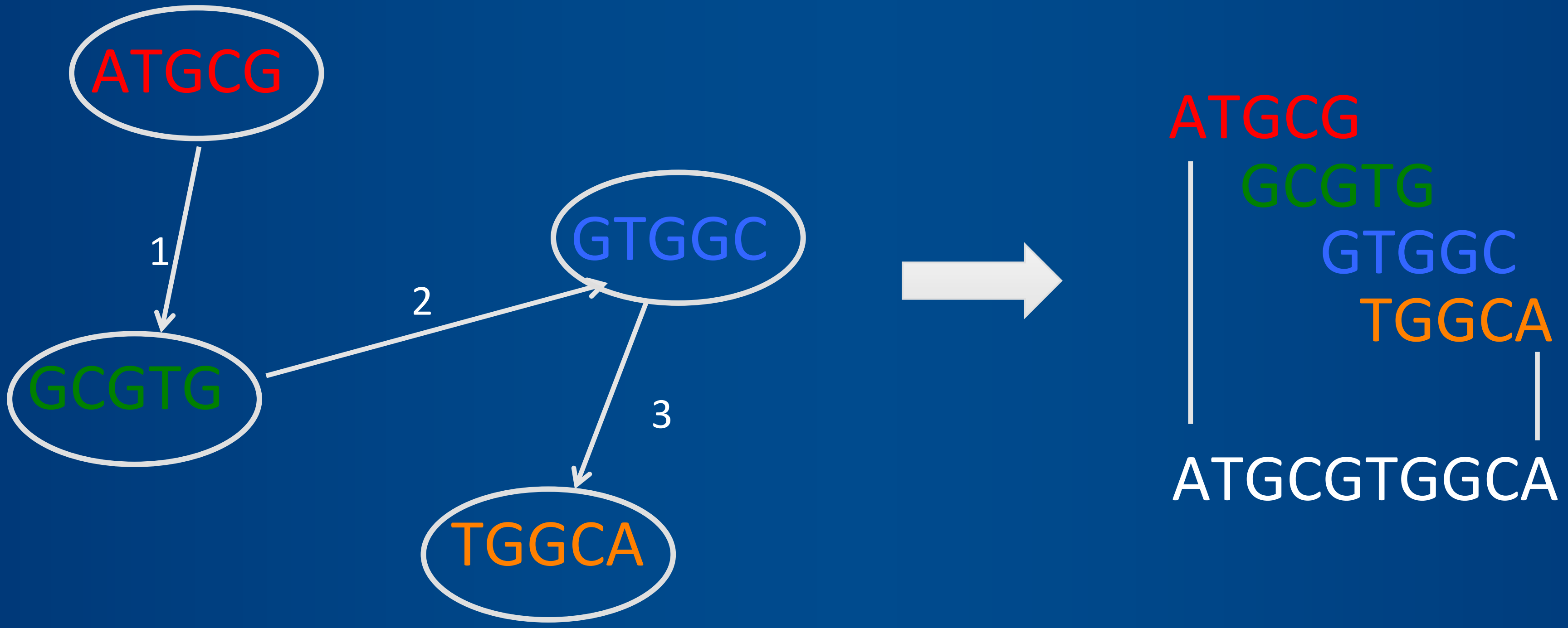
- Given: A set of reads (strings) $\{s_1, s_2, \dots, s_n\}$
- Do: Determine a large string s that "best explains" the reads

- What do we mean by "best explains"?
- What assumptions might we require?

Shortest superstring problem

- Objective: Find a string *s* such that
  - all reads $s_1, s_2, \ldots, s_n$ are substrings of *s*
  - *s* is as short as possible

- Assumptions:
  - Reads are 100% accurate
  - Identical reads must come from the same location on the genome
  - "best" = "simplest"



- The assumption is that all substrings are represented
- Even modern sequencers that generate 100nt reads <u>do not</u> cover all possible 100-mers

ATGCG GTGGC

GCGTG TGGCA

$\xrightarrow{\text{3-mers}}$

ATG, TGC, GCG,
GCG, CGT, GTG
GTG,TGG GGC
TGG, GGC, GCA

- Thus, people generally use *k-mers of certain length*

← Here we use 3-mers by cutting the original reads into reads of length 3

ATGCG
GTGGC
GCGTG
TGGCA

$\xrightarrow{\text{3-mers}}$

ATG, TGC, GCG,
~~GCG~~, CGT, GTG
~~GTG~~, TGG GGC
~~TGG~~, ~~GGC~~, GCA

make them unique

- Thus, people generally use *k*-mers of certain length

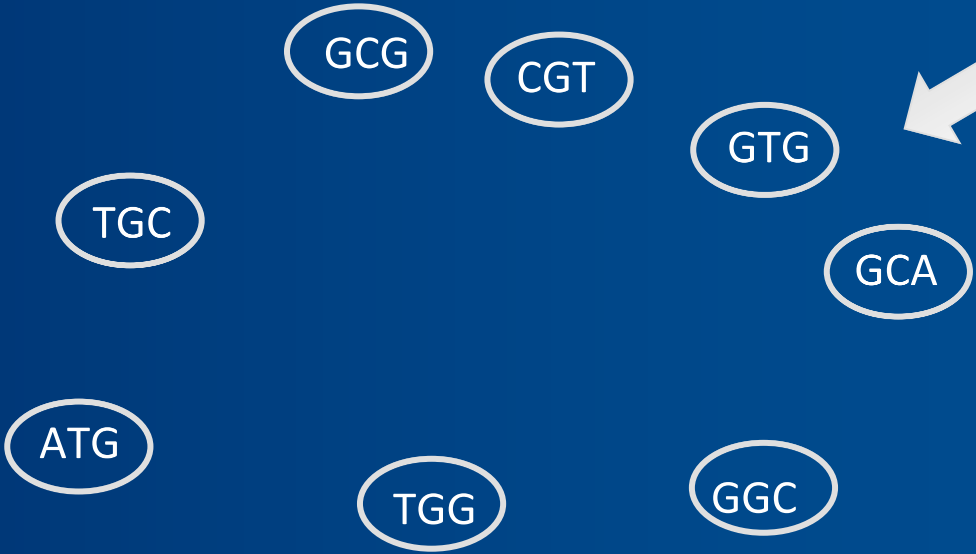
← Here we use 3-mers by cutting the original reads into reads of length 3

ATGCG GTGGC

GCGTG TGGCA

3-mers

ATG, TGC, GCG,
~~GCG~~, CGT, GTG
~~GTG~~, TGG GGC
~~TGG~~, ~~GGC~~, GCA



Draw edge from *x* to *y* where
<u>suffix</u> from *x* overlaps <u>prefix</u> from *y*

- Thus, people generally use <u>*k*-mers of certain length</u>

← Here we use 3-mers by cutting the original reads into reads of length 3

ATGCG

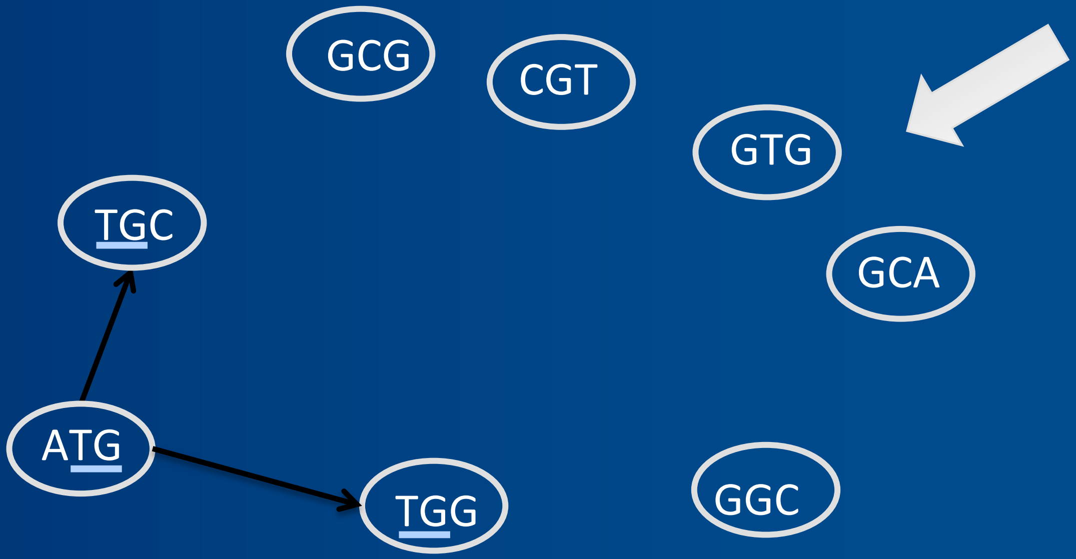
GTGGC

GCGTG

TGGCA

3-mers →

ATG, TGC, GCG,
~~GCG~~, CGT, GTG
~~GTG~~, TGG GGC
~~TGG~~, ~~GGC~~, GCA



Draw edge from *x* to *y*
where
<u>suffix</u> from *x* overlaps <u>prefix</u> from *y*

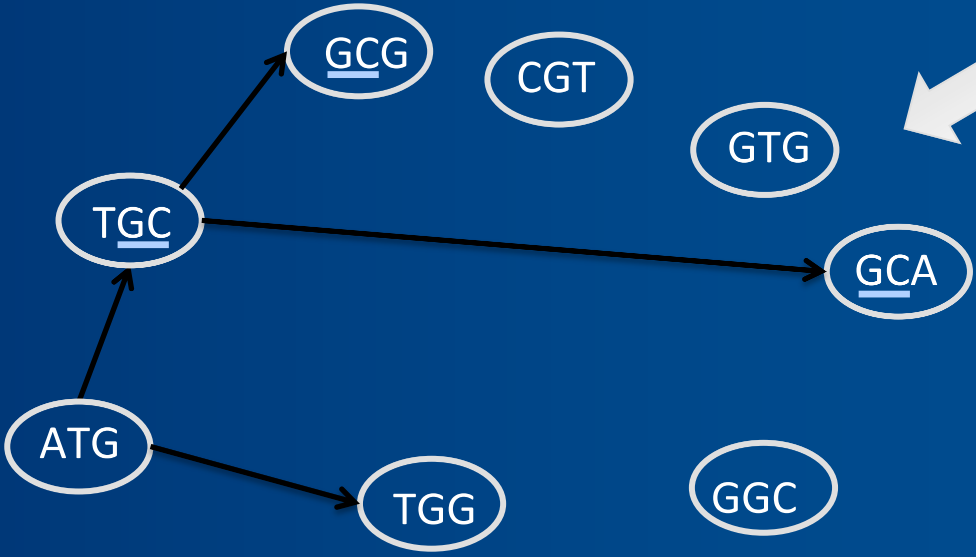
- Thus, people generally use <u>*k*-mers of certain length</u>

← Here we use 3-mers by cutting the original reads into reads of length 3

ATGCG GTGGC GCGTG TGGCA

3-mers

ATG, TGC, GCG,
~~GCG~~, CGT, GTG
~~GTG~~, TGG GGC
~~TGG~~, ~~GGC~~, GCA



Draw edge from *x* to *y* where
suffix from *x* overlaps prefix from *y*

- Thus, people generally use *k*-mers of certain length

← Here we use 3-mers by cutting the original reads into reads of length 3

ATGCG

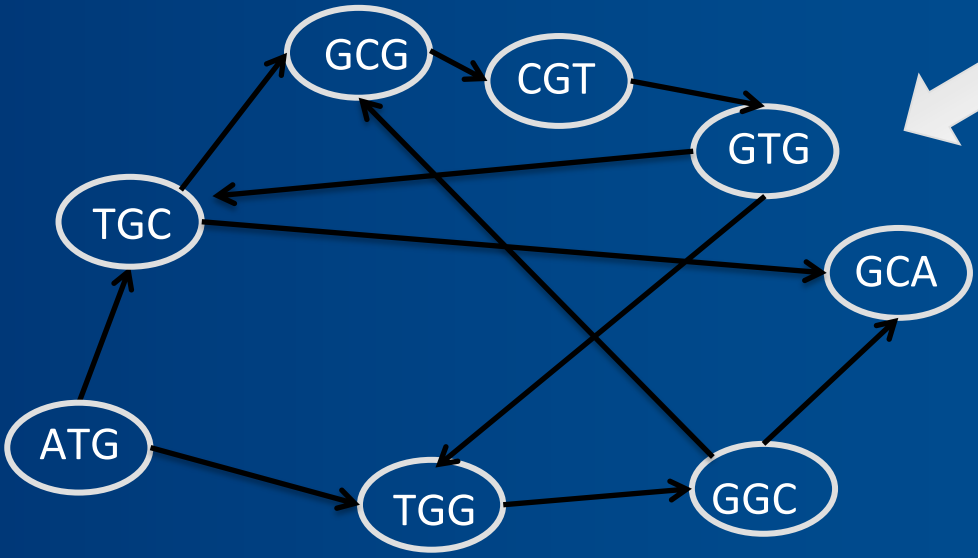
GTGGC

GCGTG

TGGCA

3-mers

ATG, TGC, GCG,
~~GCG~~, CGT, GTG
~~GTG~~, TGG GGC
~~TGG~~, ~~GGC~~, GCA



Draw edge from *x* to *y*
where
<u>suffix</u> from *x* overlaps <u>prefix</u> from *y*

- Thus, people generally use <u>*k*-mers of certain length</u>

← Here we use 3-mers by cutting the original reads into reads of length 3

ATGCG

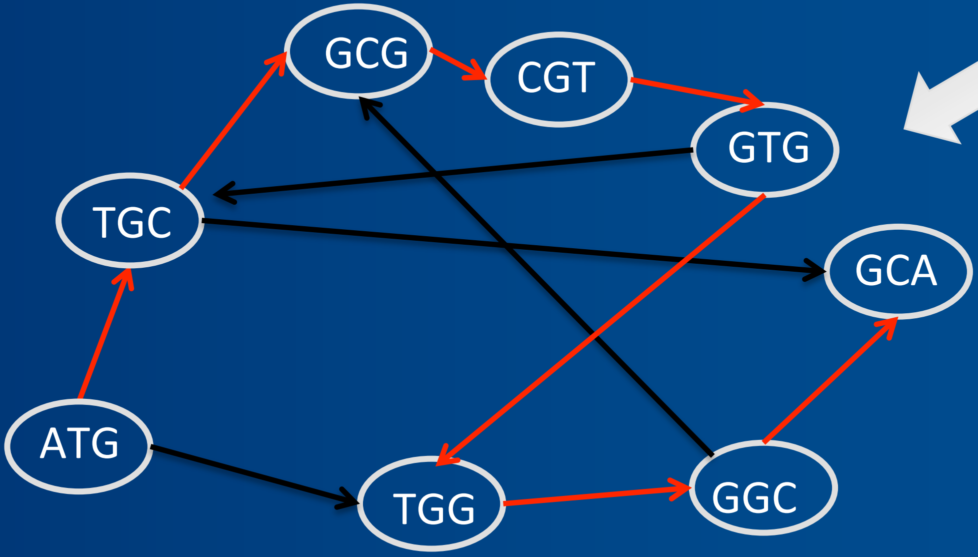
GTGGC

GCGTG

TGGCA

3-mers

ATG, TGC, GCG,
~~GCG~~, CGT, GTG
~~GTG~~, TGG GGC
~~TGG~~, ~~GGC~~, GCA



Find **Hamiltonian path**, that is, a path that visits every vertex exactly once

Record the First letter of each vertex + All letters of last vertex

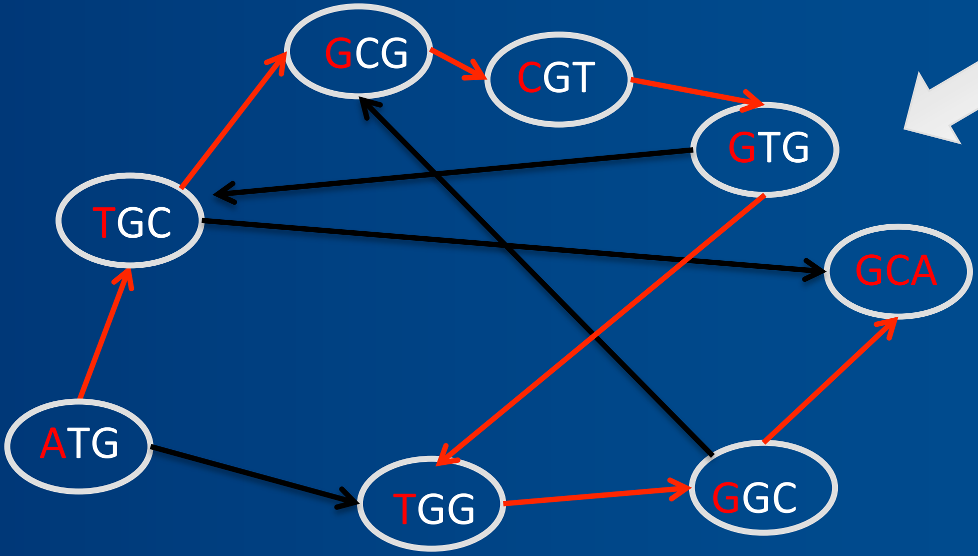
- Thus, people generally use *k-mers of certain length*

← Here we use 3-mers by cutting the original reads into reads of length 3

ATGCG GTGGC GCGTG TGGCA

3-mers

ATG, TGC, GCG,
~~GCG~~, CGT, GTG
~~GTG~~, TGG GGC
~~TGG~~, ~~GGC~~, GCA



Find **Hamiltonian path**, that is, a path that visits every vertex exactly once

Record the First letter of each vertex +
All letters of last vertex

ATGCGTGGCA

- Thus, people generally use *k*-mers of certain length

← Here we use 3-mers by cutting the original reads into reads of length 3

← UNFORTUNATELY: The Hamiltonian path problem is very difficult to solve (np-complete)

ATGCG

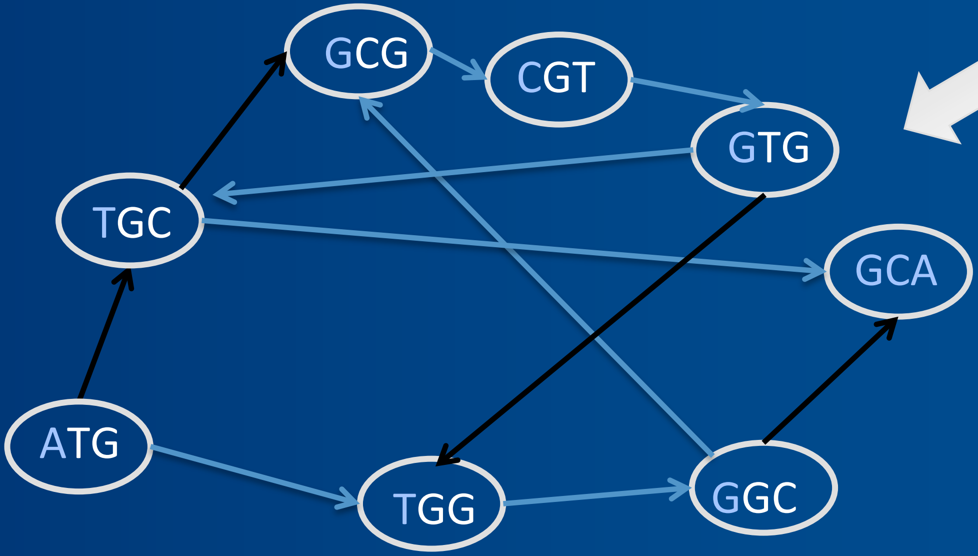
GTGGC

GCGTG

TGGCA

3-mers

ATG, TGC, GCG,
~~GCG~~, CGT, GTG
~~GTG~~, TGG GGC
~~TGG~~, ~~GGC~~, GCA



Find **Hamiltonian path**, that is, a path that visits every vertex exactly once

Record the First letter of each vertex + All letters of last vertex

ATGCGTGGCA

ATGGCGTGCA

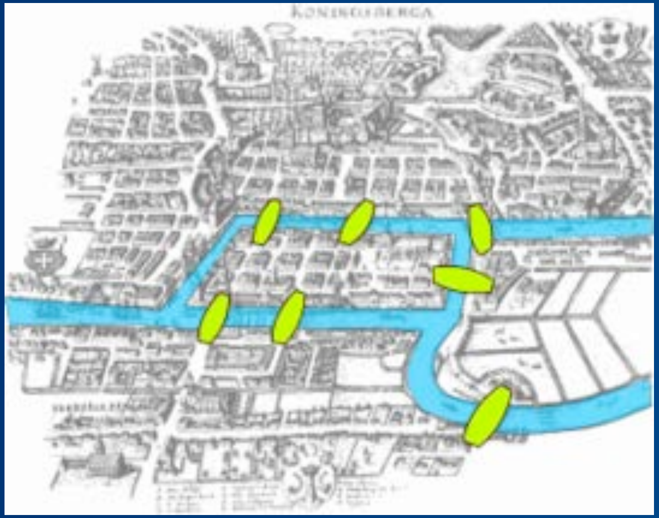
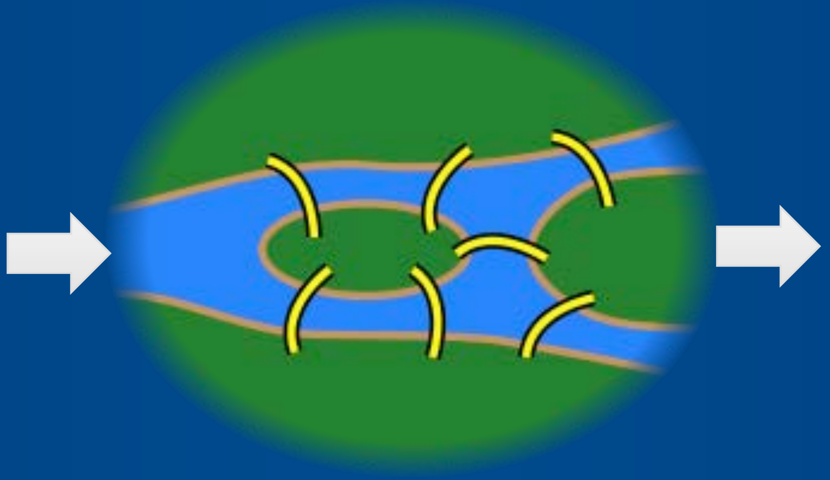
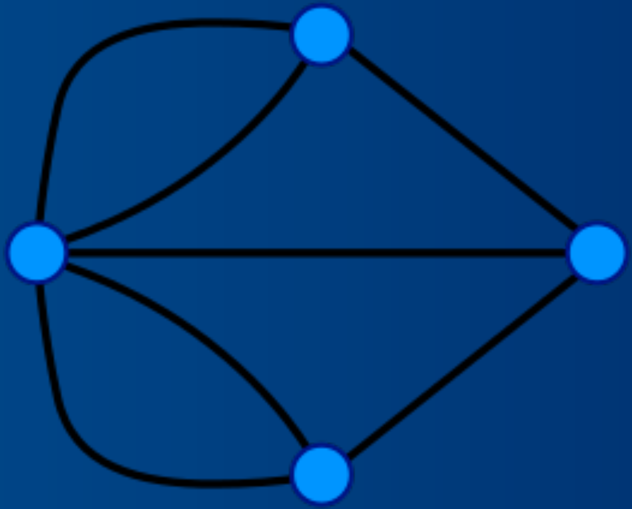
- Thus, people generally use *k*-mers of certain length

← Here we use 3-mers by cutting the original reads into reads of length 3

← UNFORTUNATELY: The Hamiltonian path problem is very difficult to solve (np-complete)

Seven bridges of Königsberg

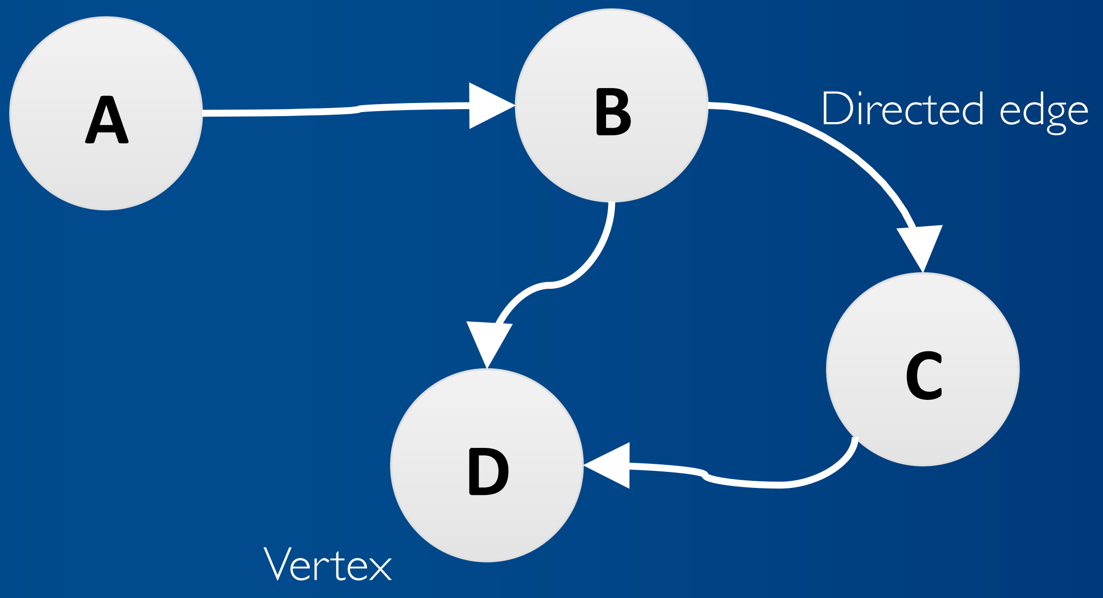
- In 1735 Leonhard Euler was presented with the following problem:
  - find a walk through the city that would cross each bridge once and only once
  - He proved that a connected graph with undirected edges contains an Eulerian cycle exactly when every node in the graph has an **even** number of edges touching it.
  - For the Königsberg Bridge graph, this is not the case because each of the four nodes has an odd number of edges touching it and so the desired stroll through the city does not exist.

https://en.wikipedia.org/wiki/Leonhard_Euler

Assembly as a graph theoretical problem

- The degree of a vertex: # of edges connected to it
- outdegree: # of outgoing edges
- indegree: # of ingoing edges
- degree(B)?
- outdegree(B)?
- indegree(D)?



Seven bridges of Königsberg II

- The case of directed graphs is similar:
  - A graph in which indegrees are equal to outdegrees for all nodes is called 'balanced'.
  - Euler's theorem states that a connected directed graph has an Eulerian cycle if and only if it is balanced.

- Mathematically/computationally finding Eulerian path is much easier than Hamiltonian

  → we need to reformulate our assembly problem

We construct a ***de Bruijn graph***:

- edges represent *k*-mers
- vertices correspond to (*k-1*)-mers

1. Form a node for every distinct prefix or suffix of a *k*-mer
2. Connect vertex *x* to vertex *y* with a directed edge if some *k*-mer (e.g., ATG) has prefix *x* (e.g., AT) and suffix *y* (e.g., TG), and label the edge with this *k*-mer.

k-mers: ATG, TGG, TGC, GTG, GGC, GCA, GCG, CGT

Distinct (k-1)-mers:

We construct a *de Bruijn graph*:

- edges represent *k*-mers
- vertices correspond to (*k*-1)-mers

1. Form a node for every distinct prefix or suffix of a *k*-mer
2. Connect vertex *x* to vertex *y* with a directed edge if some *k*-mer (e.g., ATG) has prefix *x* (e.g., AT) and suffix *y* (e.g., TG), and label the edge with this *k*-mer.

k-mers: ATG, TGG, TGC, GTG, GGC, GCA, GCG, CGT

Distinct (k-1)-mers:

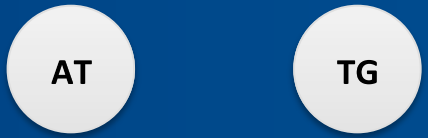
We construct a *de Bruijn graph*:

- edges represent *k*-mers
- vertices correspond to (*k*-*1*)-mers

1. Form a node for every distinct prefix or suffix of a *k*-mer
2. Connect vertex *x* to vertex *y* with a directed edge if some *k*-mer (e.g., ATG) has prefix *x* (e.g., AT) and suffix *y* (e.g., TG), and label the edge with this *k*-mer.

k-mers: A$\underline{\text{TG}}$, $\underline{\text{TG}}$G, $\underline{\text{TG}}$C, G$\underline{\text{TG}}$, GGC, GCA, GCG, CGT

Distinct (k-1)-mers:



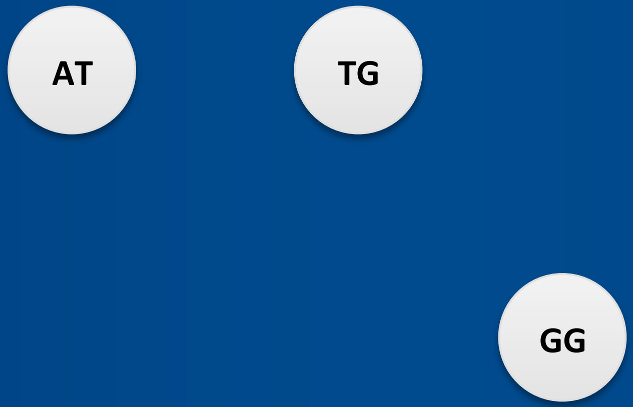
We construct a *de Bruijn graph*:

- edges represent *k*-mers
- vertices correspond to (*k-1*)-mers

1. Form a node for every distinct prefix or suffix of a *k*-mer
2. Connect vertex *x* to vertex *y* with a directed edge if some *k*-mer (e.g., ATG) has prefix *x* (e.g., AT) and suffix *y* (e.g., TG), and label the edge with this *k*-mer.

k-mers: ATG, TGG, TGC, GTG, GGC, GCA, GCG, CGT

Distinct (k-1)-mers:



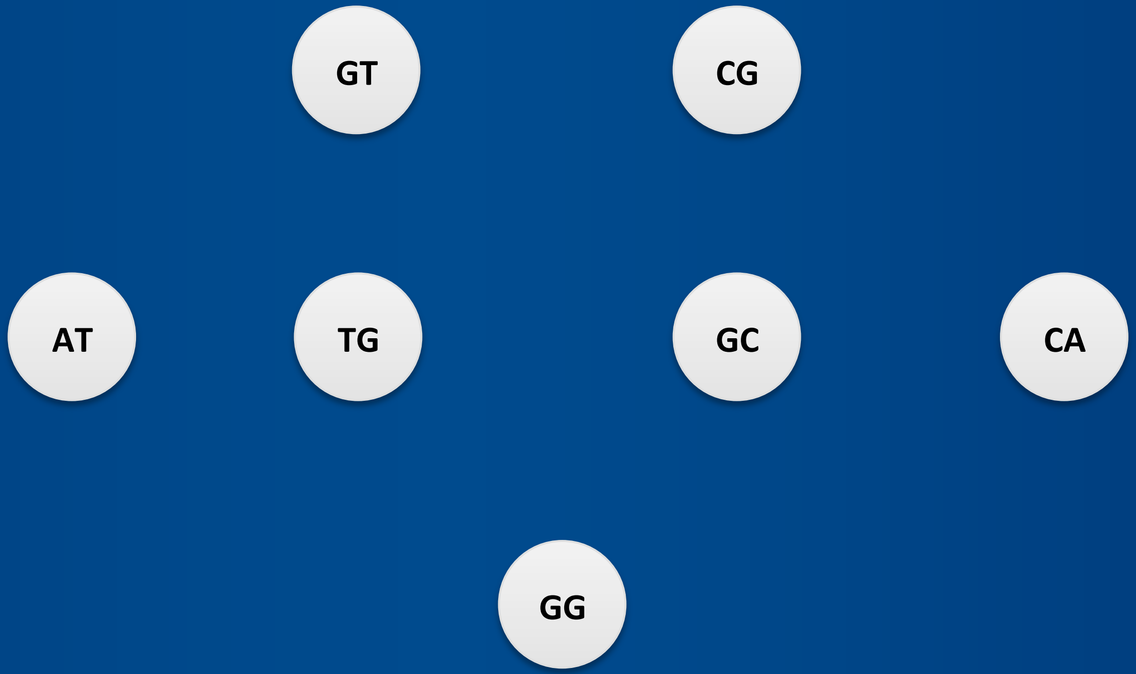
We construct a <u>*de Bruijn graph*</u>:

- edges represent *k*-mers
- vertices correspond to (*k*-*1*)-mers

1. Form a node for every <u>distinct</u> prefix or suffix of a *k*-mer
2. Connect vertex *x* to vertex *y* with a directed edge if some *k*-mer (e.g., ATG) has prefix *x* (e.g., AT) and suffix *y* (e.g., TG), and label the edge with this *k*-mer.

k-mers: ATG, TGG, TGC, GTG, GGC, GCA, GCG, CGT

Distinct (k-1)-mers:



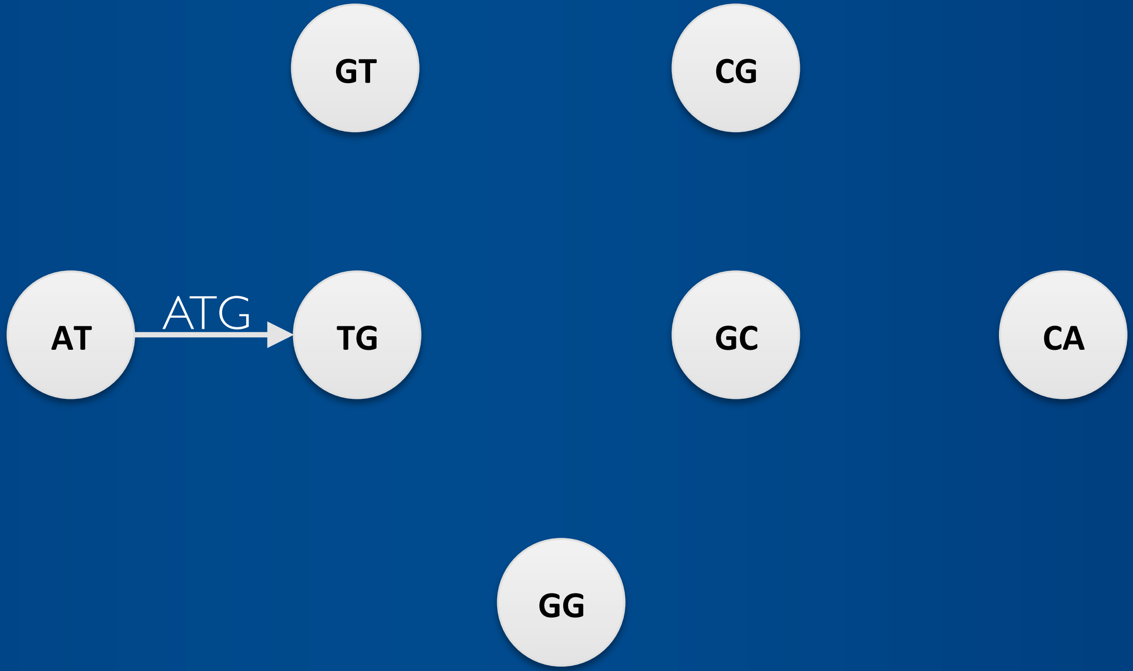
We construct a ***de Bruijn graph***:

- edges represent *k*-mers
- vertices correspond to (*k-1*)-mers

1. Form a node for every distinct prefix or suffix of a *k*-mer
2. Connect vertex *x* to vertex *y* with a directed edge if some *k*-mer (e.g., ATG) has prefix *x* (e.g., AT) and suffix *y* (e.g., TG), and label the edge with this *k*-mer.

k-mers: ATG, TGG, TGC, GTG, GGC, GCA, GCG, CGT

Distinct (k-1)-mers:



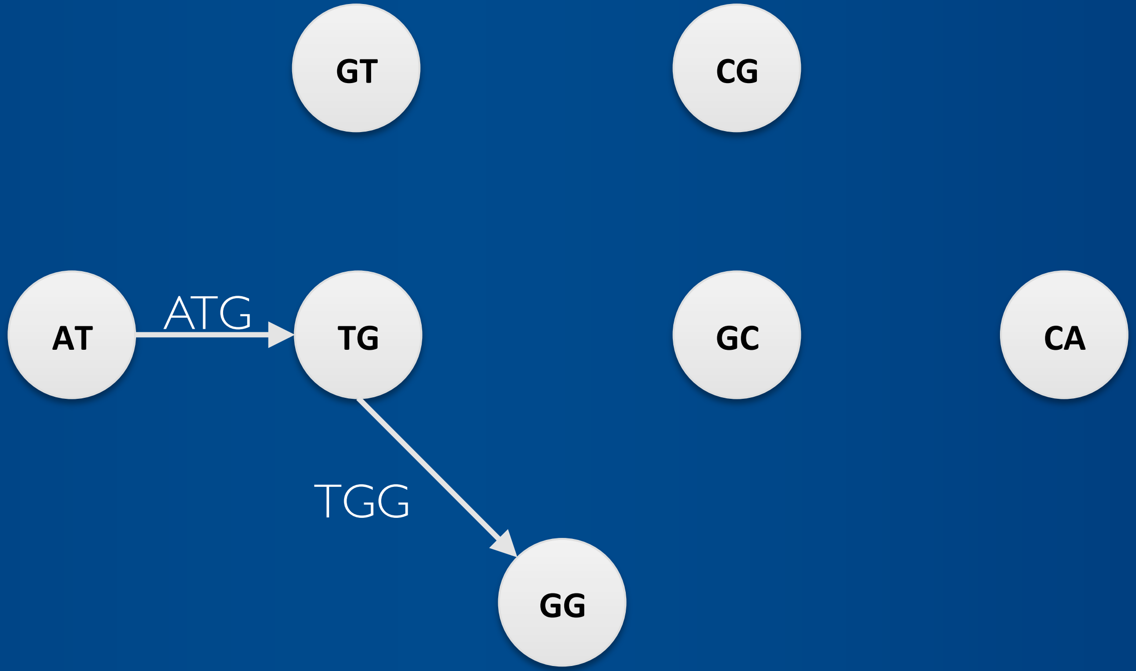
We construct a de Bruijn graph:

- edges represent *k*-mers
- vertices correspond to (*k*-1)-mers

1. Form a node for every distinct prefix or suffix of a *k*-mer
2. Connect vertex *x* to vertex *y* with a directed edge if some *k*-mer (e.g., ATG) has prefix *x* (e.g., AT) and suffix *y* (e.g., TG), and label the edge with this *k*-mer.

k-mers: ATG, TGG, TGC, GTG, GGC, GCA, GCG, CGT

Distinct (k-1)-mers:



$$\begin{tikzcd} & GT & & CG & \\ AT \arrow[r, "ATG"] & TG \arrow[dr, "TGG"'] & & GC & CA \\ & & GG & & \end{tikzcd}$$

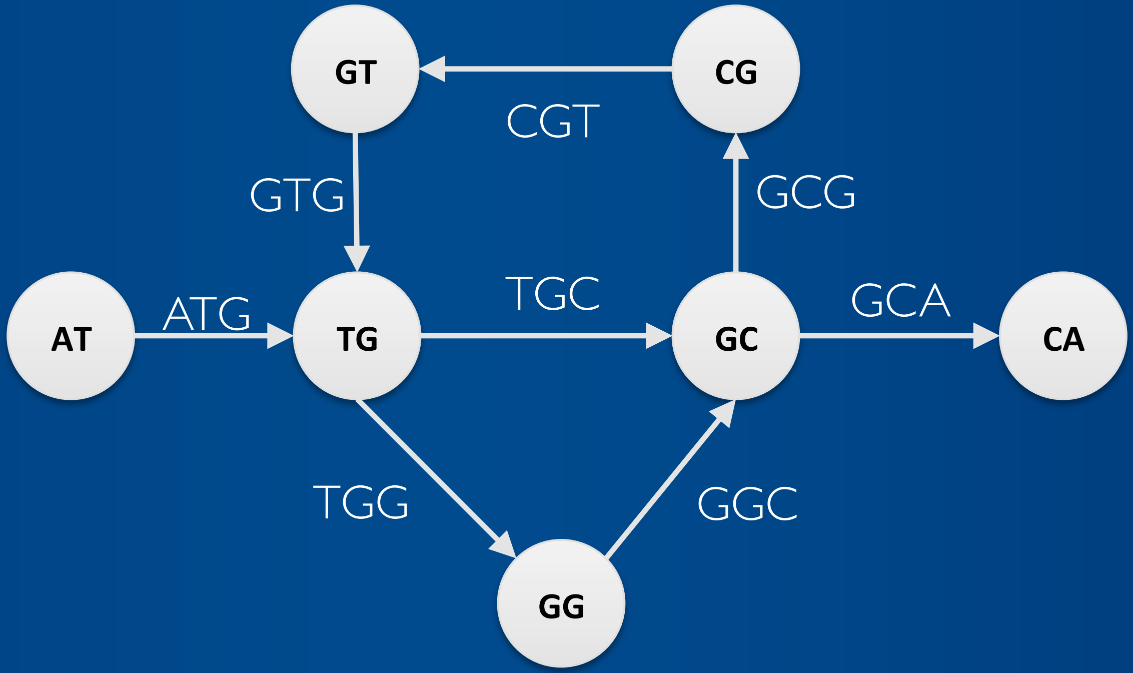
We construct a <u>*de Bruijn graph*</u>:

- edges represent *k*-mers
- vertices correspond to (*k*-*1*)-mers

1. Form a node for every <u>distinct</u> prefix or suffix of a *k*-mer
2. Connect vertex *x* to vertex *y* with a directed edge if some *k*-mer (e.g., ATG) has prefix *x* (e.g., AT) and suffix *y* (e.g., TG), and label the edge with this *k*-mer.

k-mers: ATG, TGG, TGC, GTG, GGC, GCA, GCG, CGT

Distinct (k-1)-mers:


$$\begin{tikzcd} & GT \arrow[d, "GTG"'] & & CG \arrow[ll, "CGT"] & \\ AT \arrow[r, "ATG"] & TG \arrow[rr, "TGC"] \arrow[dr, "TGG"'] & & GC \arrow[u, "GCG"'] \arrow[r, "GCA"] & CA \\ & & GG \arrow[ur, "GGC"'] & & \end{tikzcd}$$

Can we find a DNA sequence containing all *k*-mers?

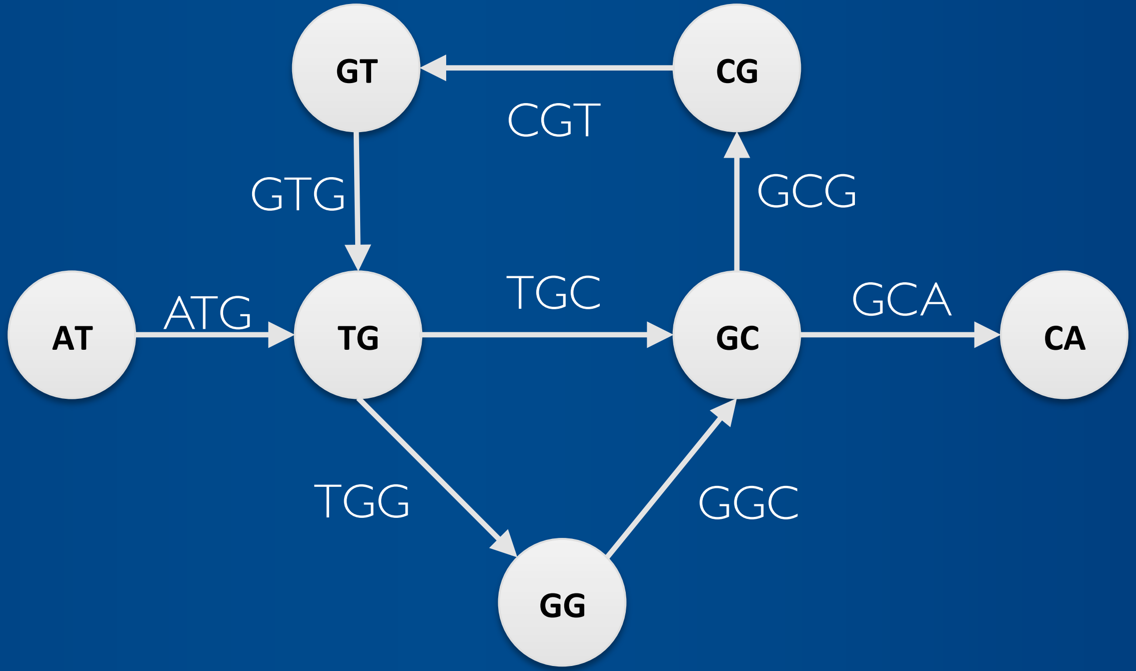
→ In a *de Bruijn* graph, can we find a path that visits every edge of the graph exactly once?

→ Eulerian path

- a vertex *v* is semibalanced if |indegree(v) – outdegree(v)| = 1
- a connected graph has an Eulerian path if and only if it contains at most two semibalanced vertices

k-mers: ATG, TGG, TGC, GTG, GGC, GCA, GCG, CGT

Distinct (k-1)-mers:



Can we find a DNA sequence containing all *k*-mers?

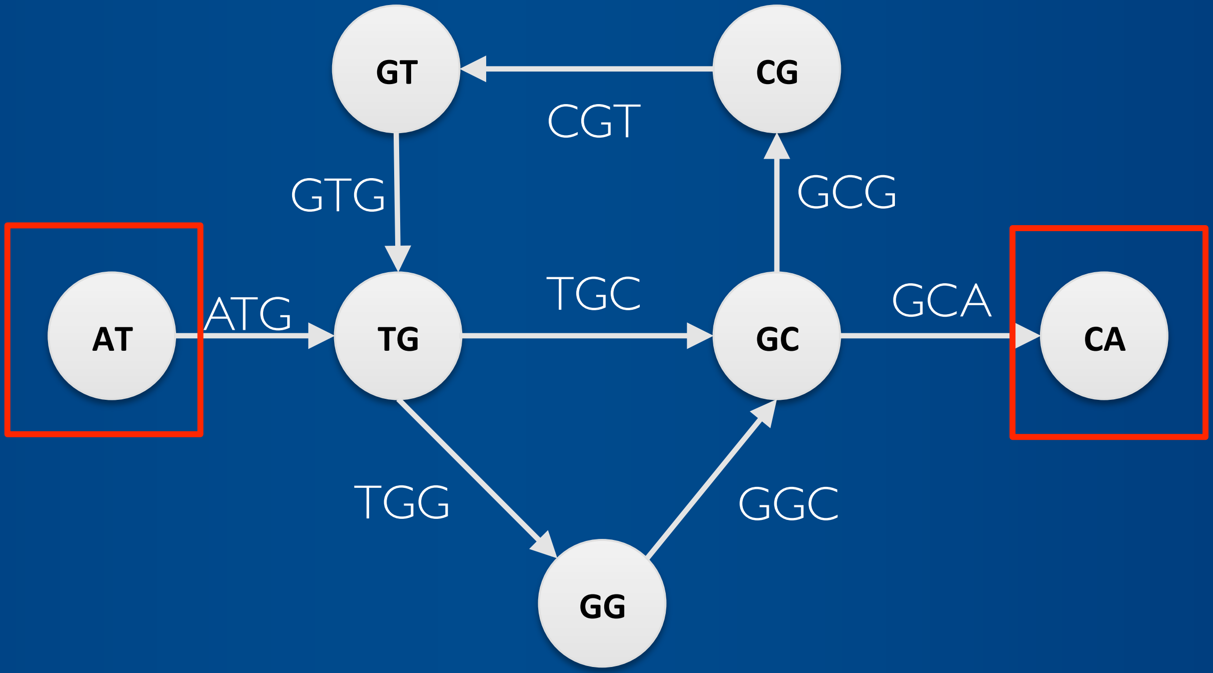
→ In a *de Bruijn* graph, can we find a path that visits every edge of the graph exactly once?

→ Eulerian path

- a vertex *v* is semibalanced if |indegree(v) – outdegree(v)| = 1
- a connected graph has an Eulerian path if and only if it contains at most two semibalanced vertices

k-mers: ATG, TGG, TGC, GTG, GGC, GCA, GCG, CGT

Distinct (k-1)-mers:



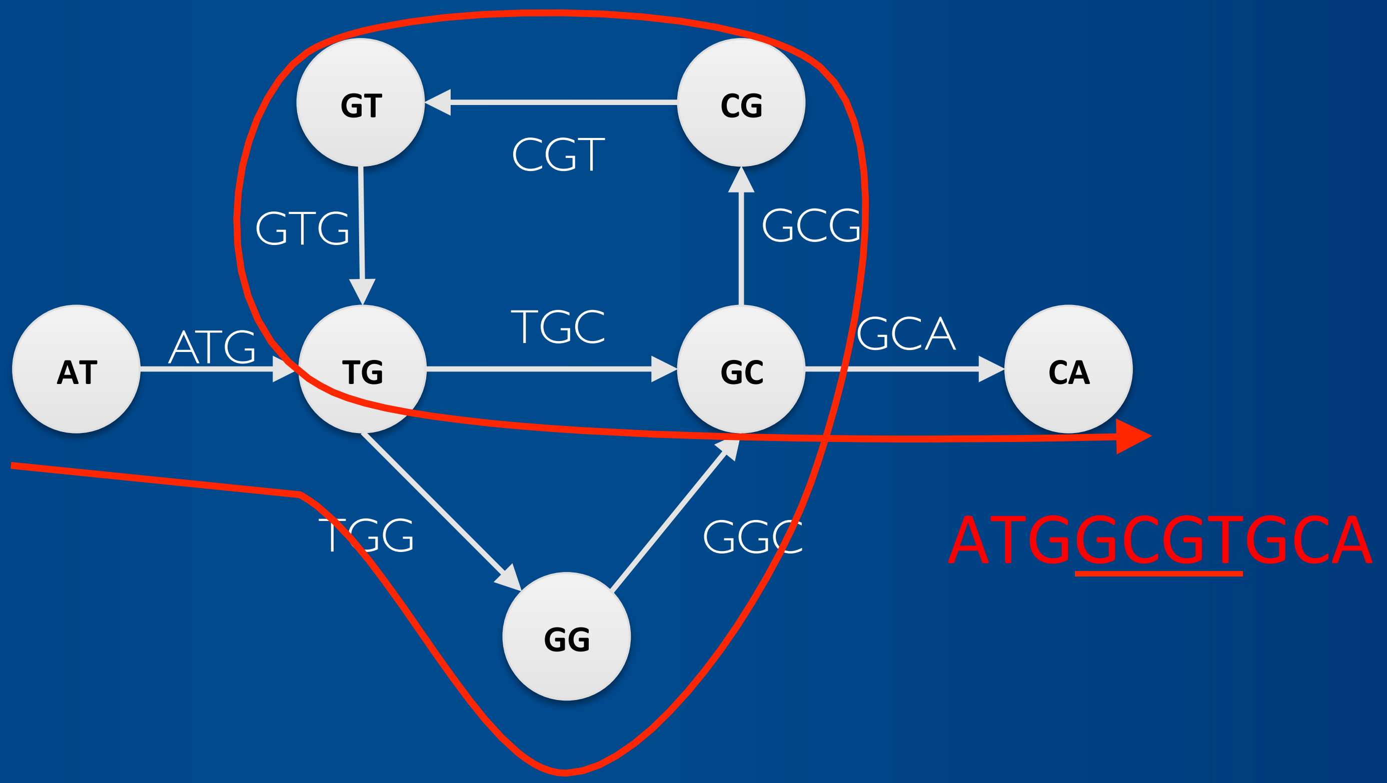
Can we find a DNA sequence containing all *k*-mers?

→ In a *de Bruijn* graph, can we find a path that visits every edge of the graph exactly once?

→ Eulerian path

k-mers: ATG, TGG, TGC, GTG, GGC, GCA, GCG, CGT

Distinct (k-1)-mers:



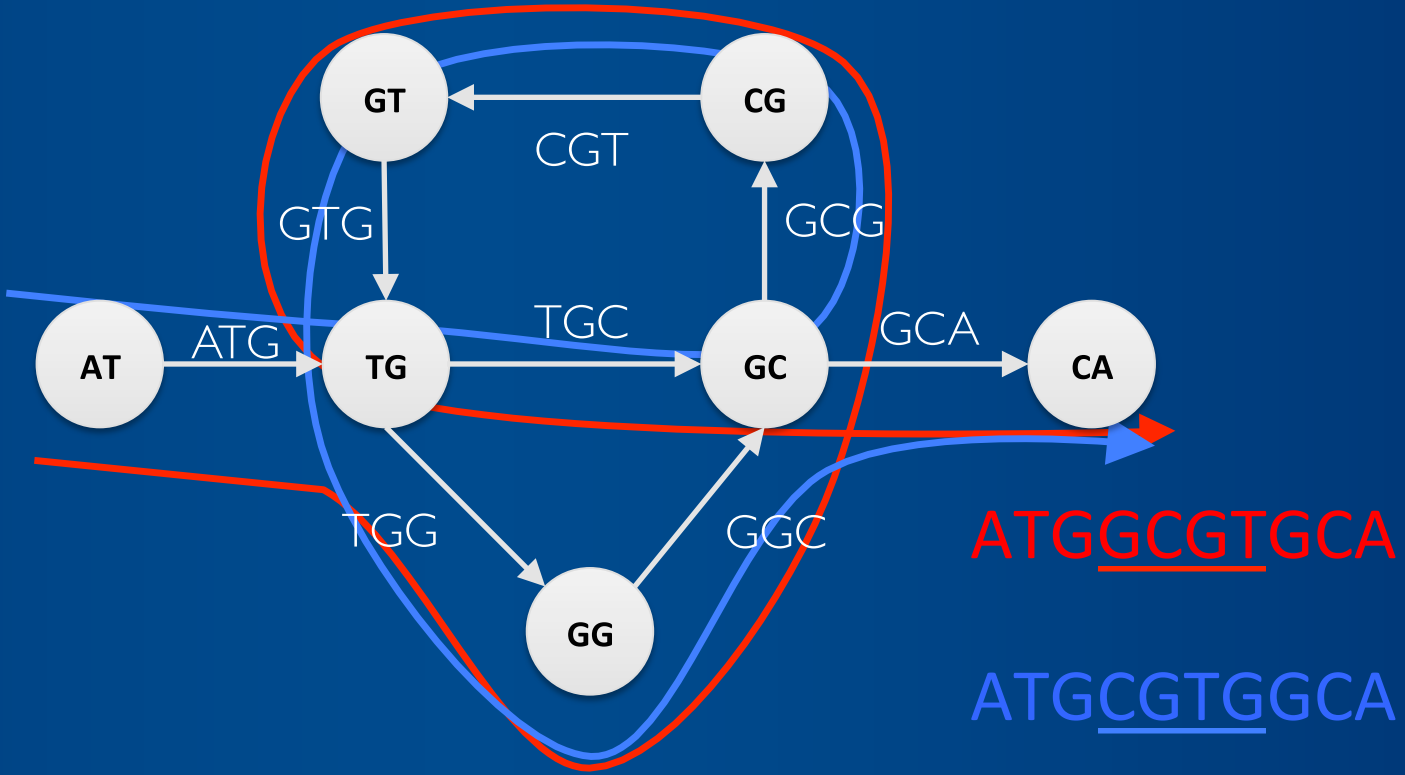
Can we find a DNA sequence containing all *k*-mers?

→ In a *de Bruijn* graph, can we find a path that visits every edge of the graph exactly once?

→ Eulerian path

k-mers: ATG, TGG, TGC, GTG, GGC, GCA, GCG, CGT

Distinct (k-1)-mers:



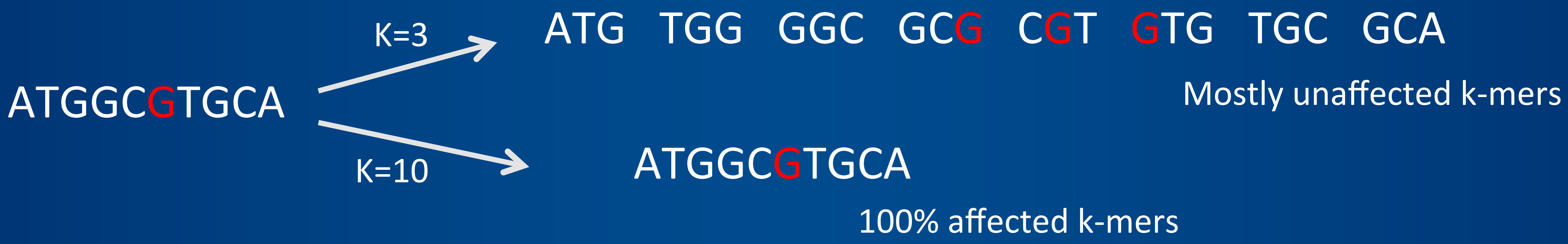
Underlying assumptions

- Four hidden assumptions that do **not** hold for next-generation sequencing
  We took for granted that:
  1. we can generate all *k*-mers present in the genome
  2. all *k*-mers are error free
  3. each *k*-mer appears at most once in the genome
  4. the genome consists of a single chromosome

Underlying assumptions

- Four hidden assumptions that do not hold for next-generation sequencing We took for granted that:
  1. we can generate all *k*-mers present in the genome
  2. all *k*-mers are error free

- That is the reason that we do not choose the longest possible *k*-mer
- The smaller the *k*-mer the higher the possibility that we see all *k*-mers
- Errors:

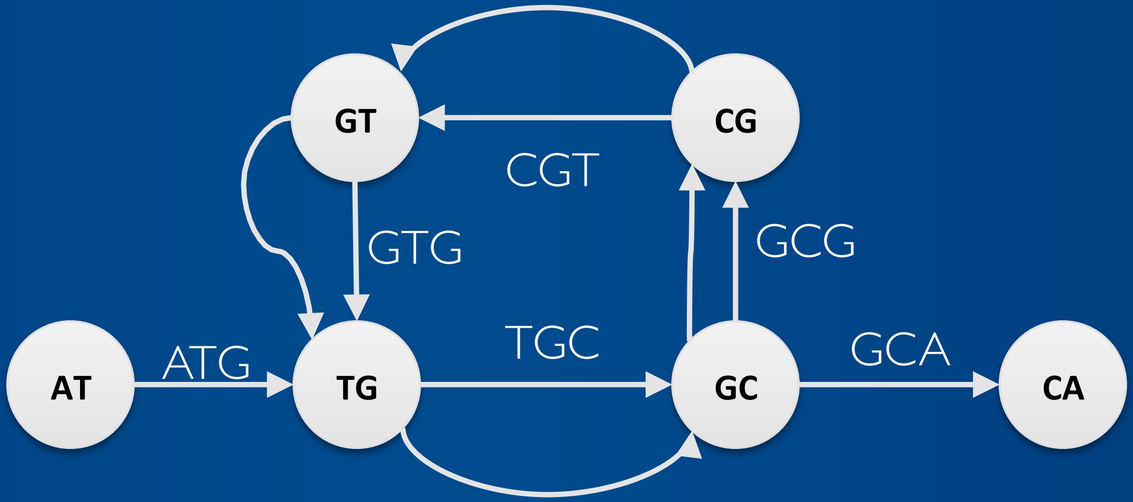


Each k-mer appears at most once in the genome → repeats

- This is most often not true
- This is known as k-mer multiplicity

k-mers:
ATG, GCA,
TGC, TGC, GTG, GTG, GCG, GCG, CGT, CGT

Distinct (k-1)-mers:



ATGCGTGCGTGCA

MASSEY UNIVERSITY
TE KUNENGA KI PŪREHUROA
UNIVERSITY OF NEW ZEALAND

Questions?

References

How to apply de Bruijn graphs to genome assembly. Phillip E C Compeau, Pavel A Pevzner & Glenn Tesler. *Nature Biotechnology* 29, 987–991 (2011) doi:10.1038/nbt.2023 Published online 08 November 2011

Sequence Assembly. Lecture by Mark Craven (craven@biostat.wisc.edu). BMI/CS 576 (www.biostat.wisc.edu/bmi576/), Fall 2011

Sebastian Schmeier
s.schmeier@gmail.com
http://sschmeier.github.io/bioinf-workshop/

THE ENGINE
OF THE NEW
NEW ZEALAND