MASSEY UNIVERSITY
TE KUNENGA KI PŪREHUROA
UNIVERSITY OF NEW ZEALAND

Genome Assembly
An Introduction

Sebastian Schmeier
s.schmeier@gmail.com
http://sschmeier.com/bioinf-workshop/
2016

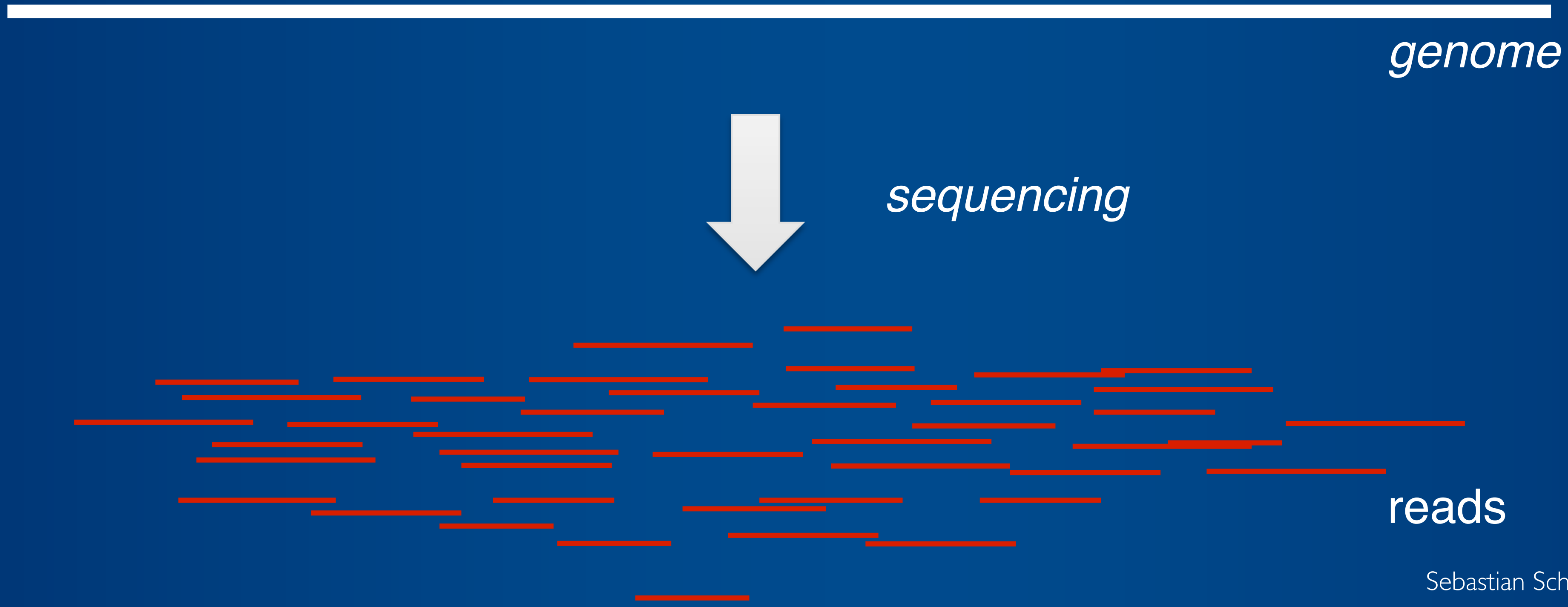
THE ENGINE
OF THE NEW
NEW ZEALAND

Overview

- *De novo* genome assembly
- The fragment assembly problem
- Shortest superstring problem
- Seven bridges of Königsberg
- Assembly as a graph theoretical problem
- We construct a ***de Bruijn* graph**
- Underlying assumptions of genome assemblies

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

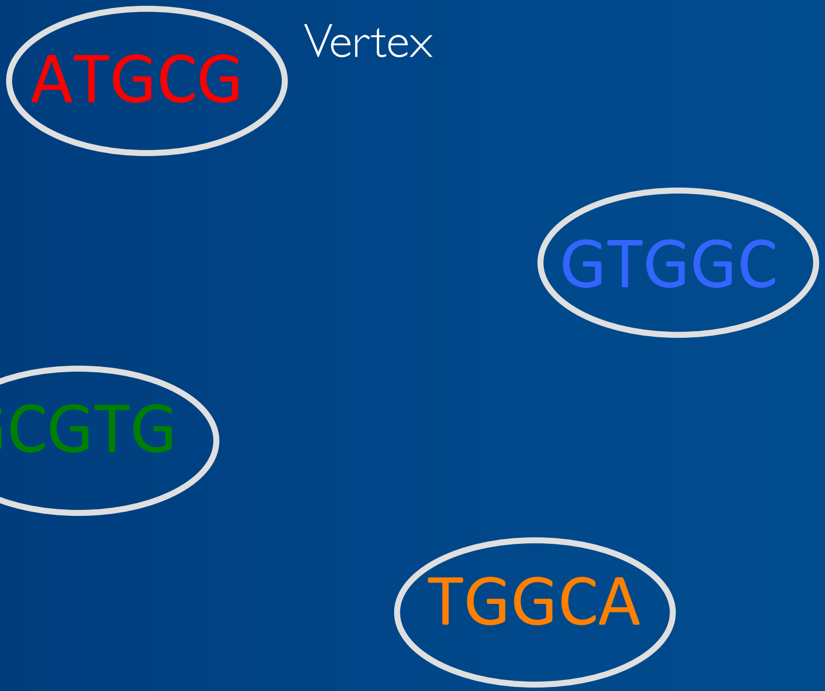


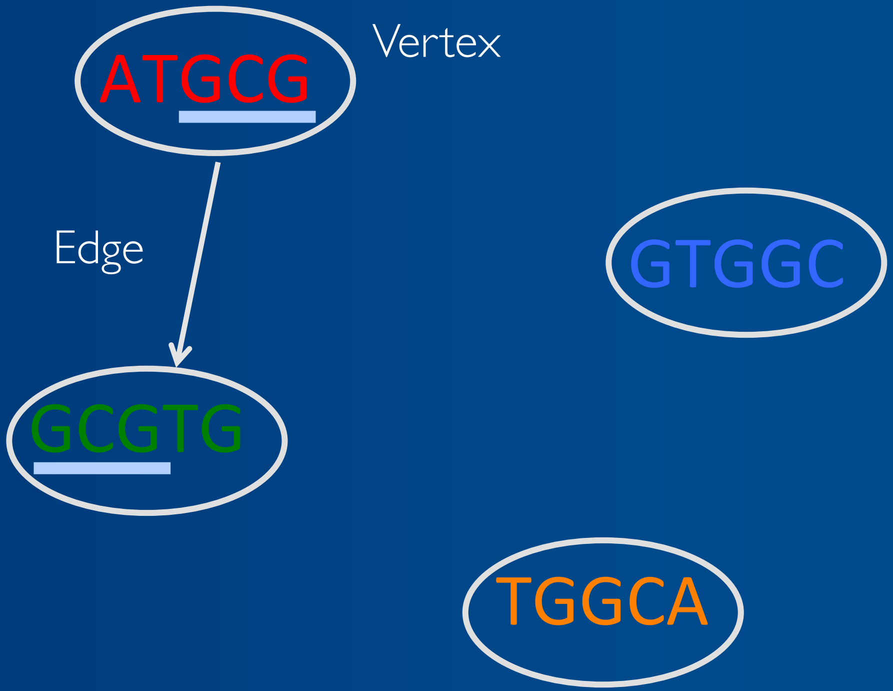
ATGCG

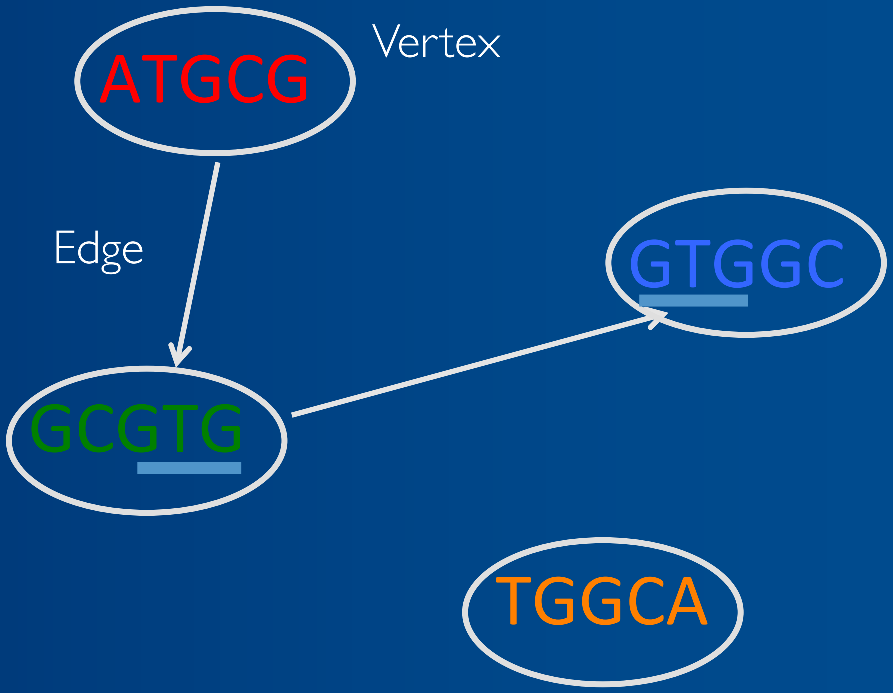
GTGGC

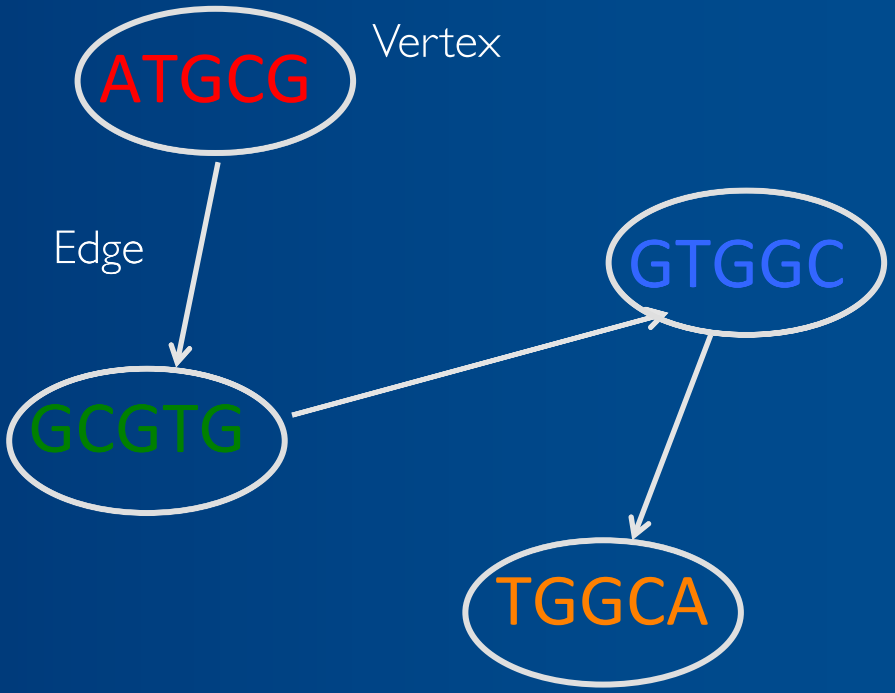
GCGTG

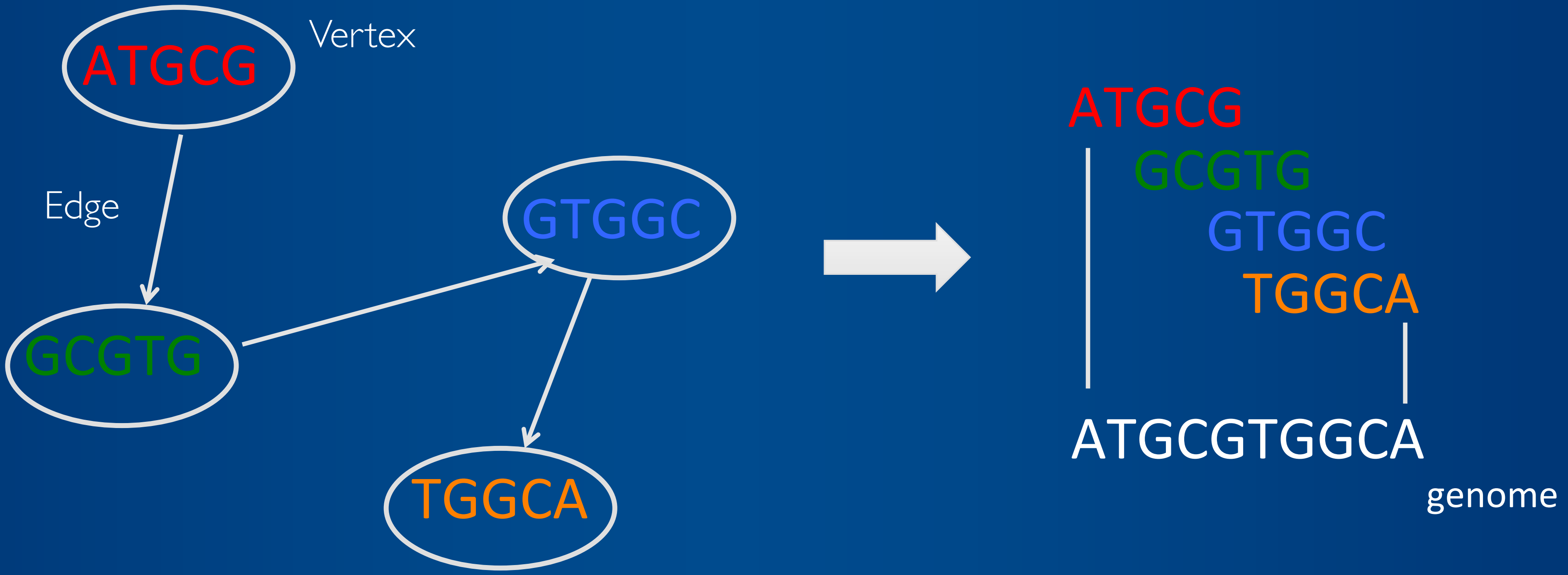
TGGCA


ATGCG
Vertex
GTGGC
GCGTG
TGGCA


ATGCG
Vertex
Edge
GCGTG
GTGGC
TGGCA


ATGCG
Vertex
Edge
GTGGC
GCGTG
TGGCA


ATGCG
Vertex
Edge
GTGGC
GCGTG
TGGCA


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

The fragment assembly problem

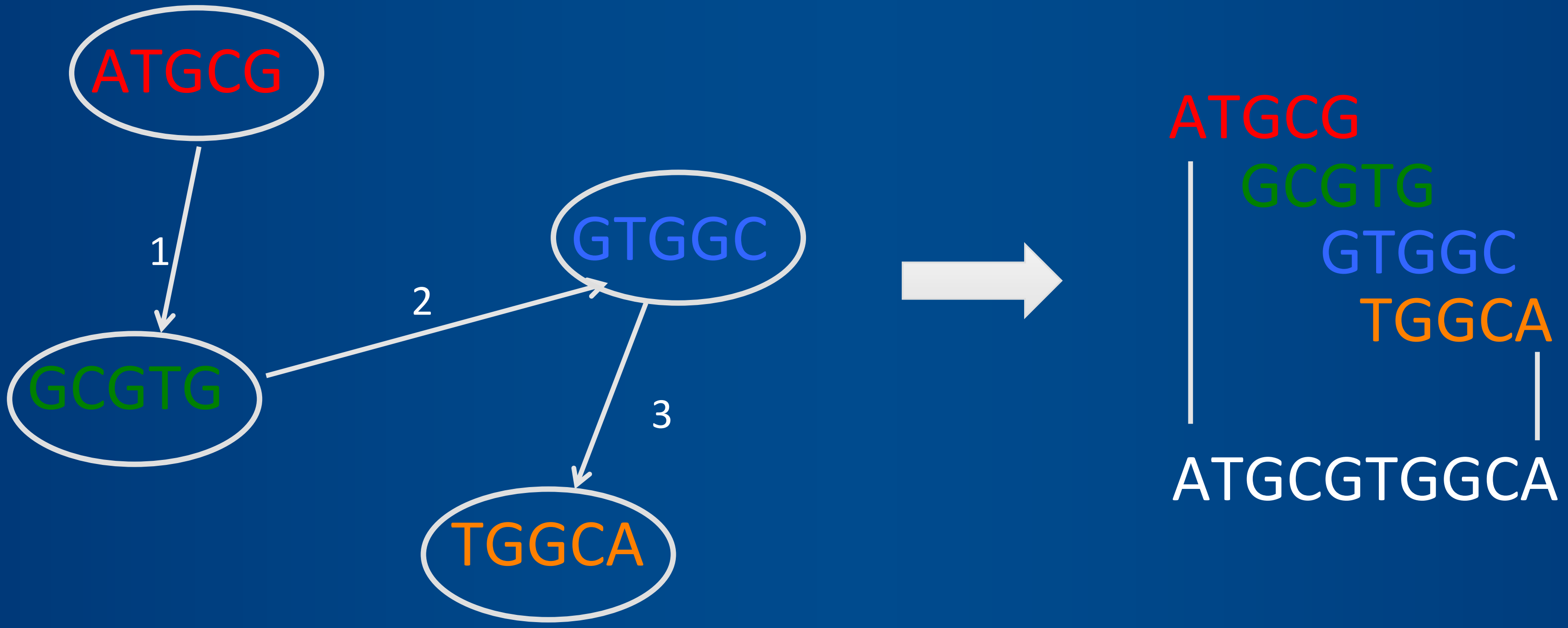
- Given: A set of reads (strings) $\{s_1, s_2, \ldots , s_n\}$
- Do: Determine a large string *s* that "best explains" the reads

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

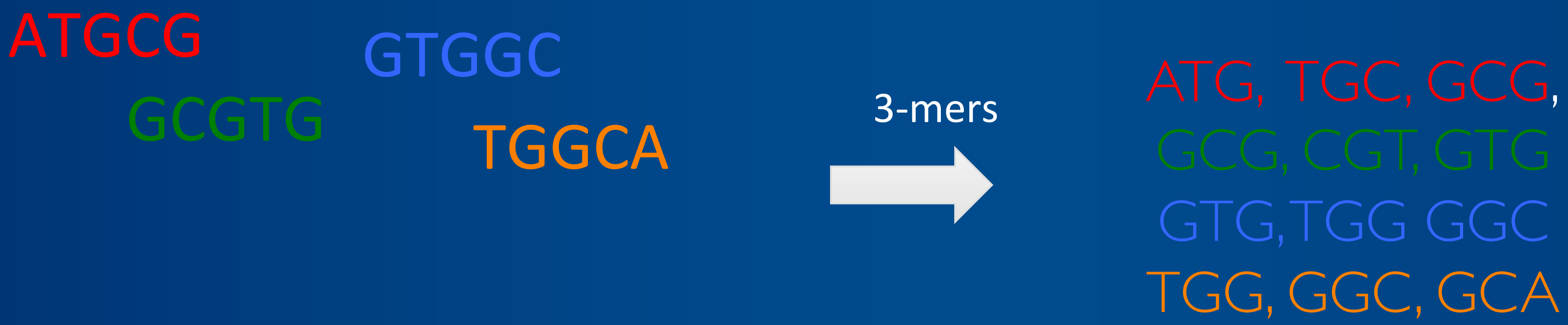
Shortest superstring problem

- 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



- 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 <u>*k*-mers of certain length</u>

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

ATGCG

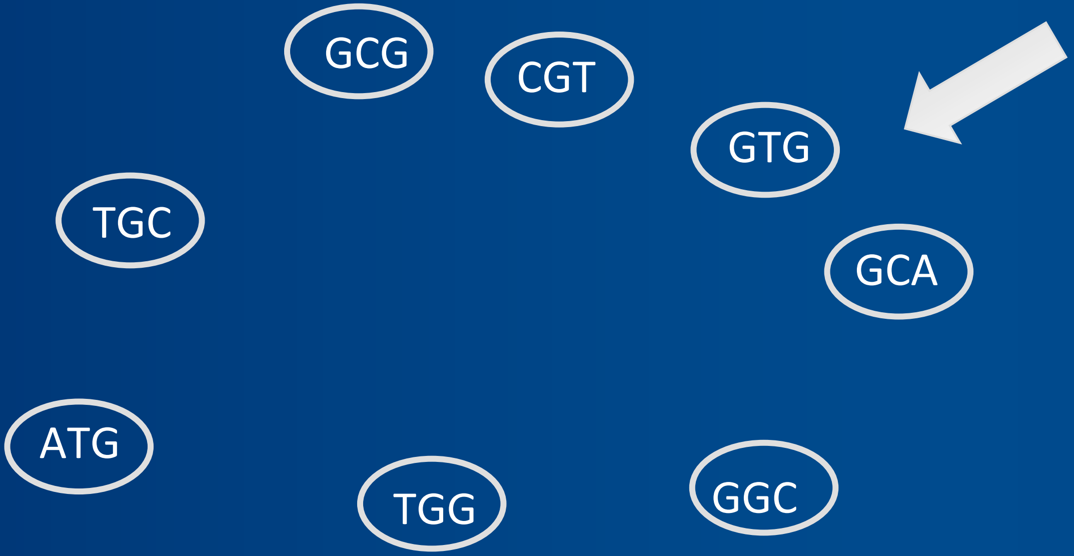
GTGGC

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

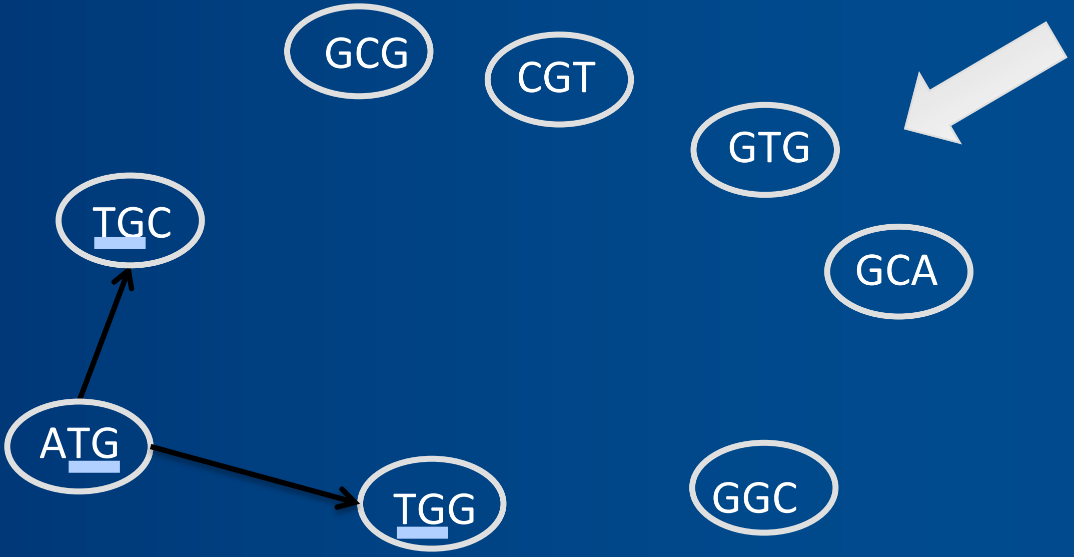
GTGGC

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 *k*-mers of certain length

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

ATGCG

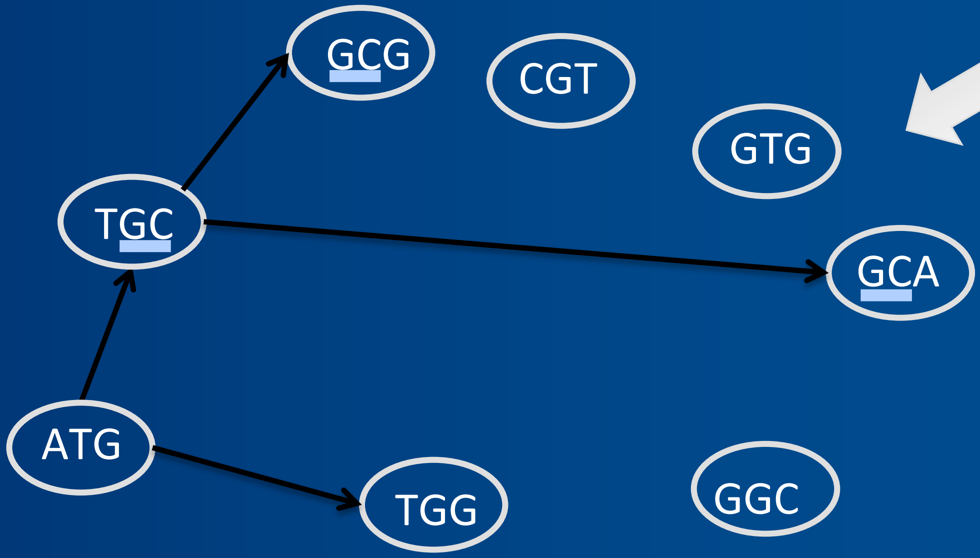
GCGTG

GTGGC

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

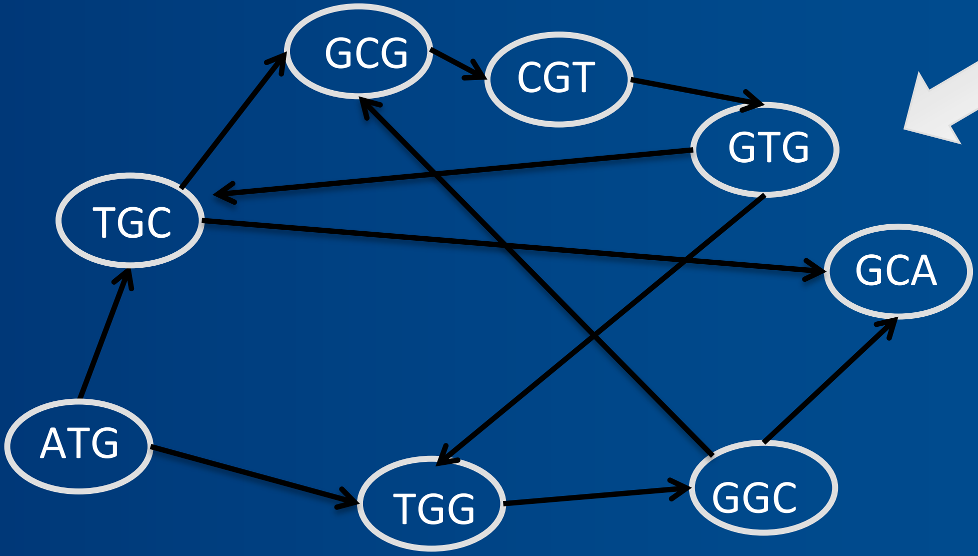
← Here we use 3-mers by cutting the original reads into reads of length 3

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

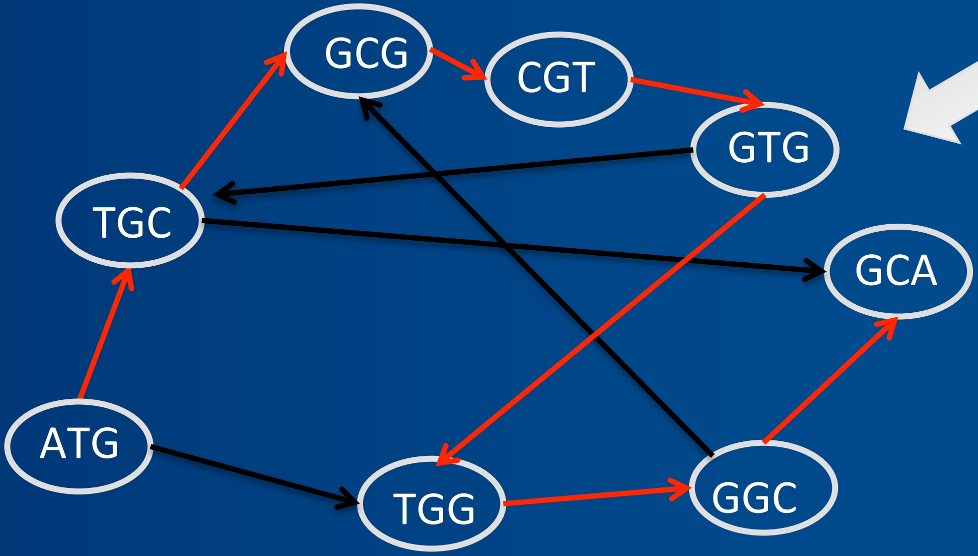
GTGGC

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

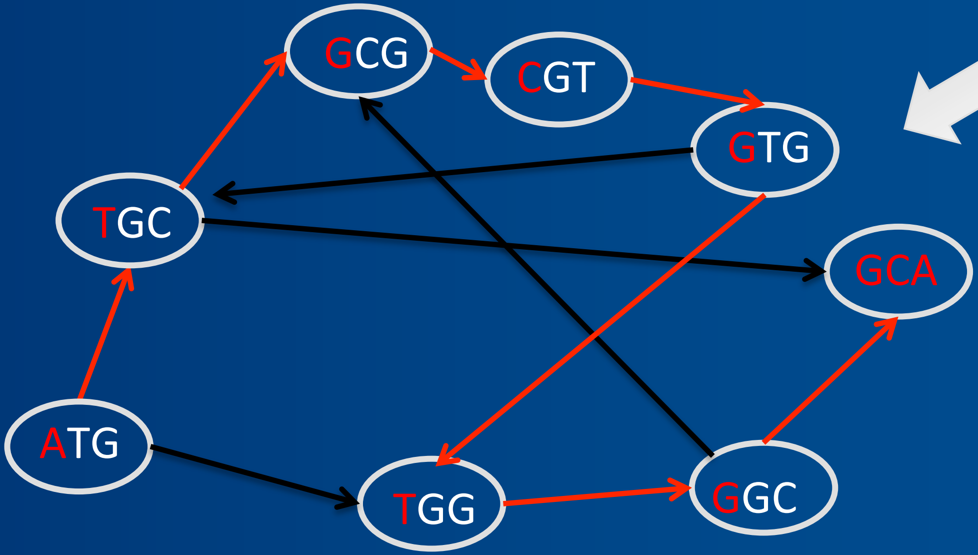
- Thus, people generally use *k*-mers of certain length

← 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

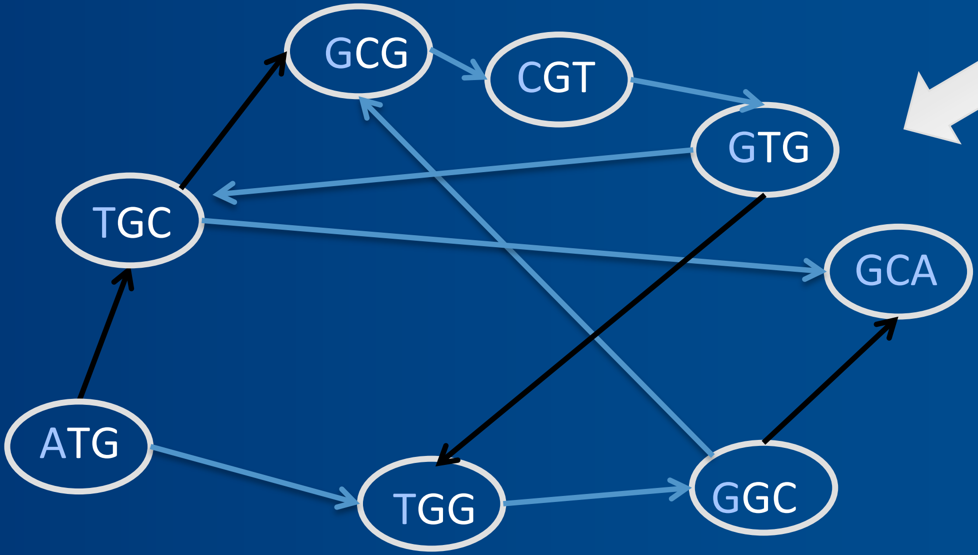
← UNFORTUNATELY: The Hamiltonian path problem is very difficult to solve (np-complete)

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 <u>vertex</u> exactly once

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

ATGCGTGGCA

ATGGCGTGCA

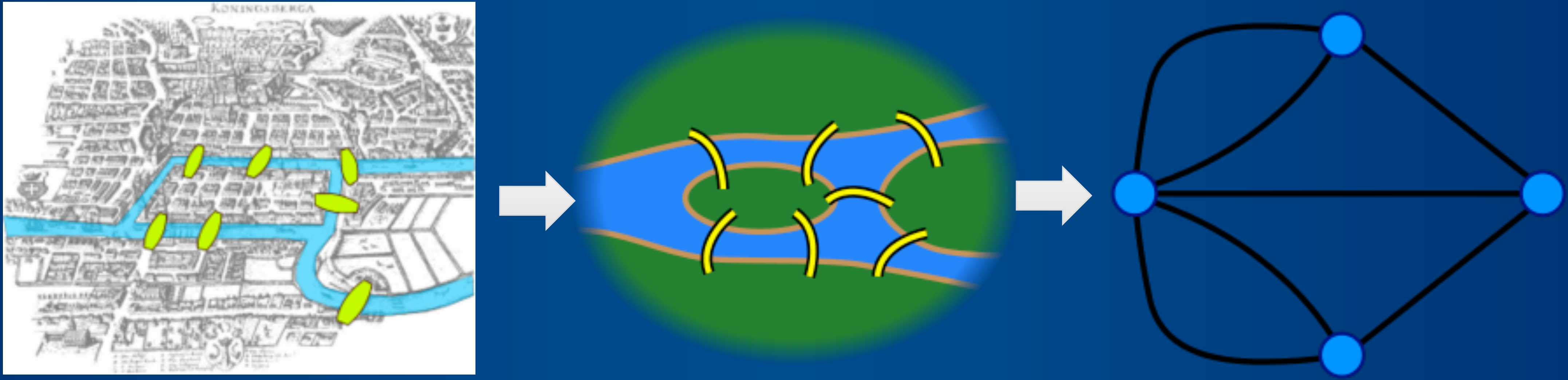
- Thus, people generally use <u>*k*-mers of certain length</u>

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

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

Seven bridges of Königsberg

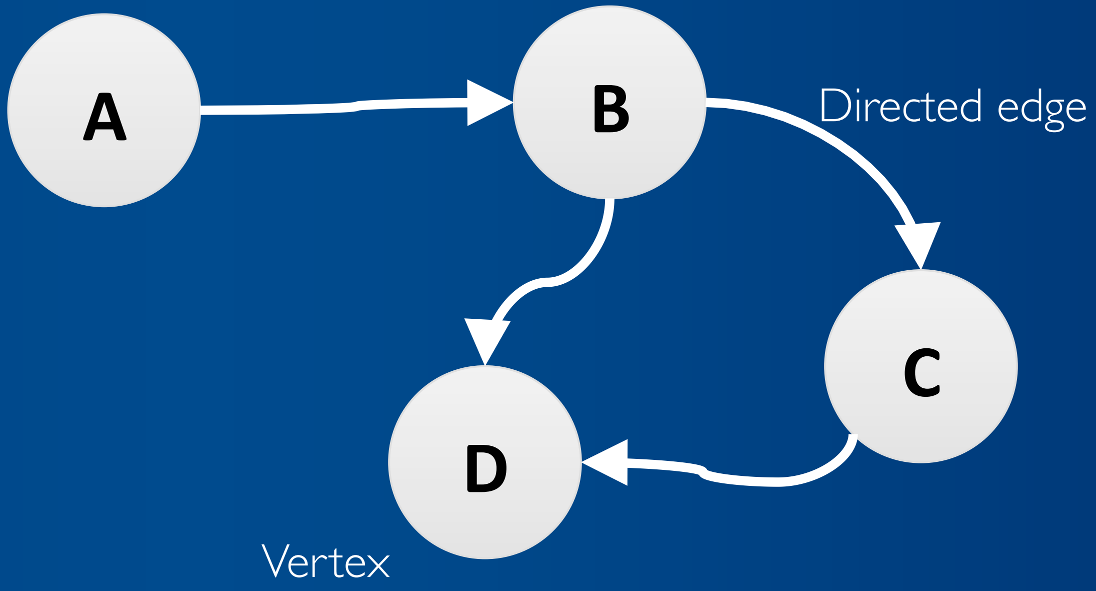
- In 1735 Leonhard Euler was presented with the following problem:
  - find a walk through the city that would cross each bridge once and only once
  - He proved that a connected graph with undirected edges contains an Eulerian cycle exactly when every node in the graph has an **even** number of edges 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:

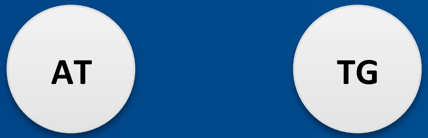
We construct a ***de Bruijn graph***:

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

Distinct (k-1)-mers:



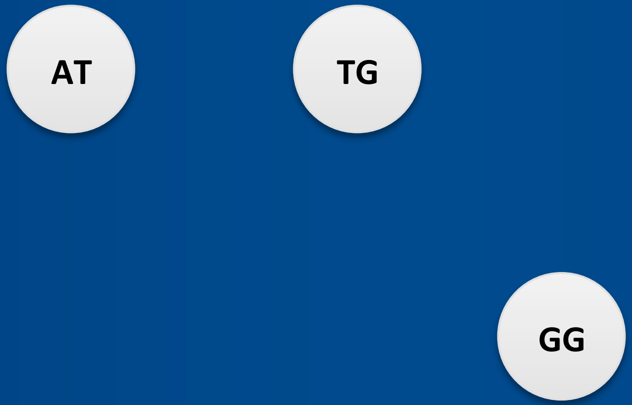
We construct a ***de Bruijn graph***:

- 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:



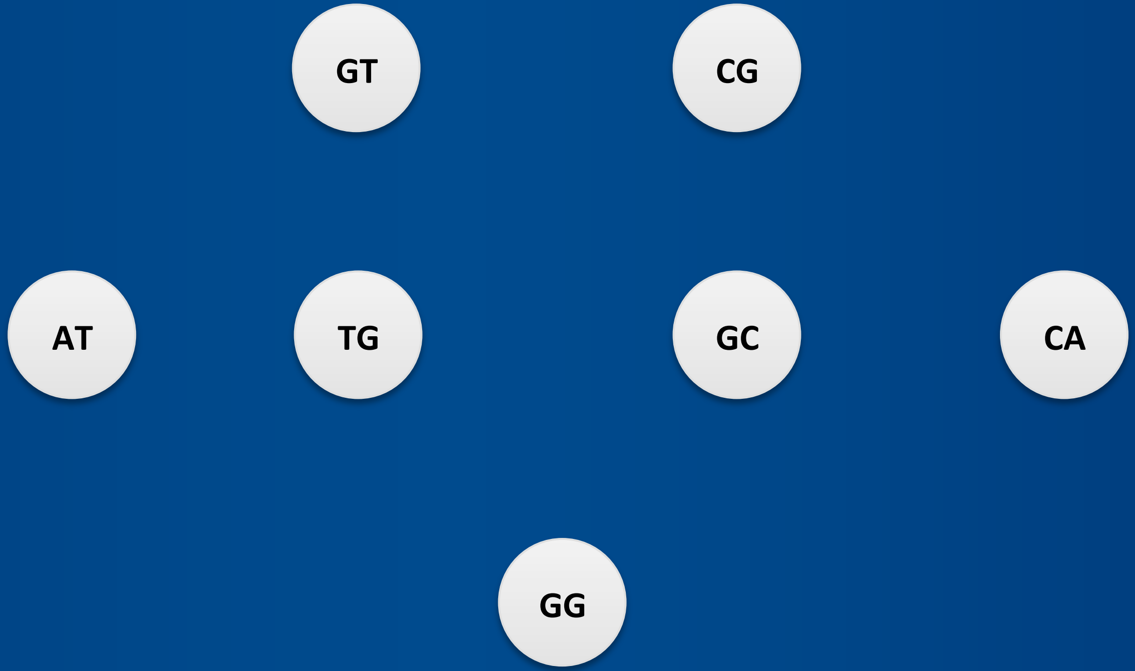
We construct a _de Bruijn graph_:

- 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:



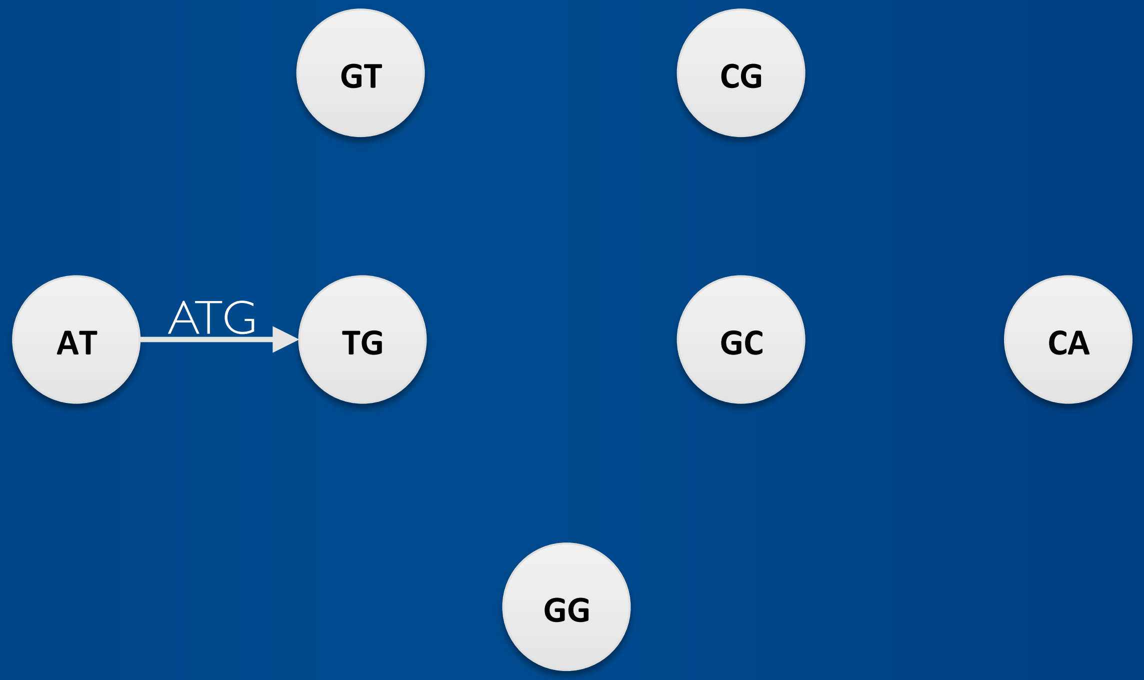
We construct a *de Bruijn graph*:

- 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:



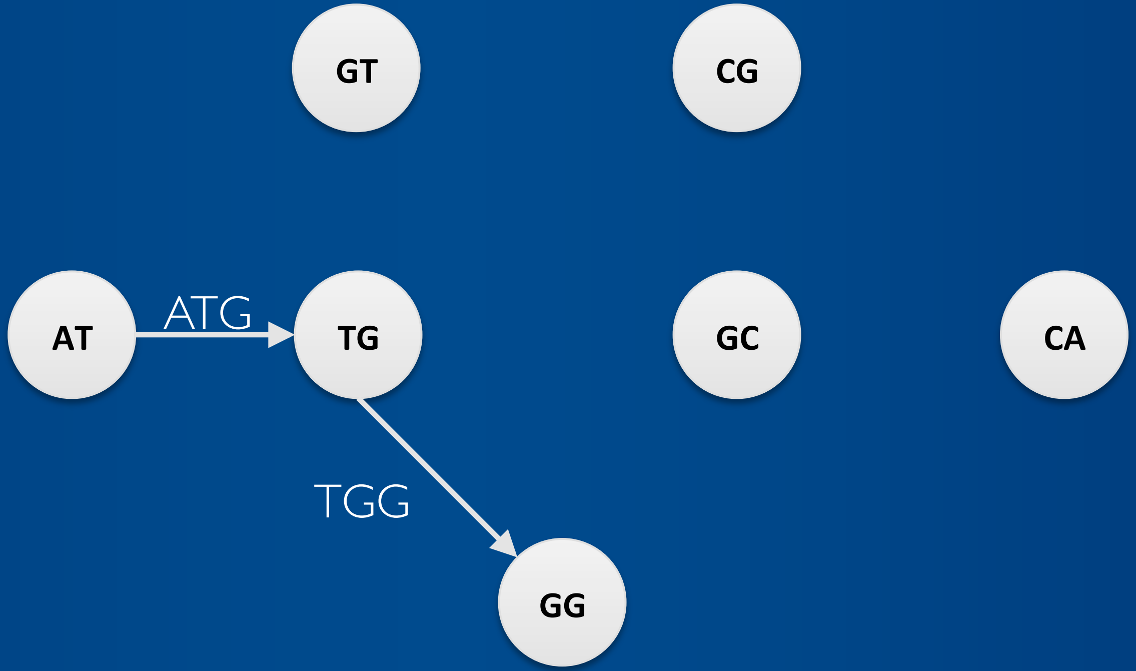
We construct a *de Bruijn graph*:

- 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:



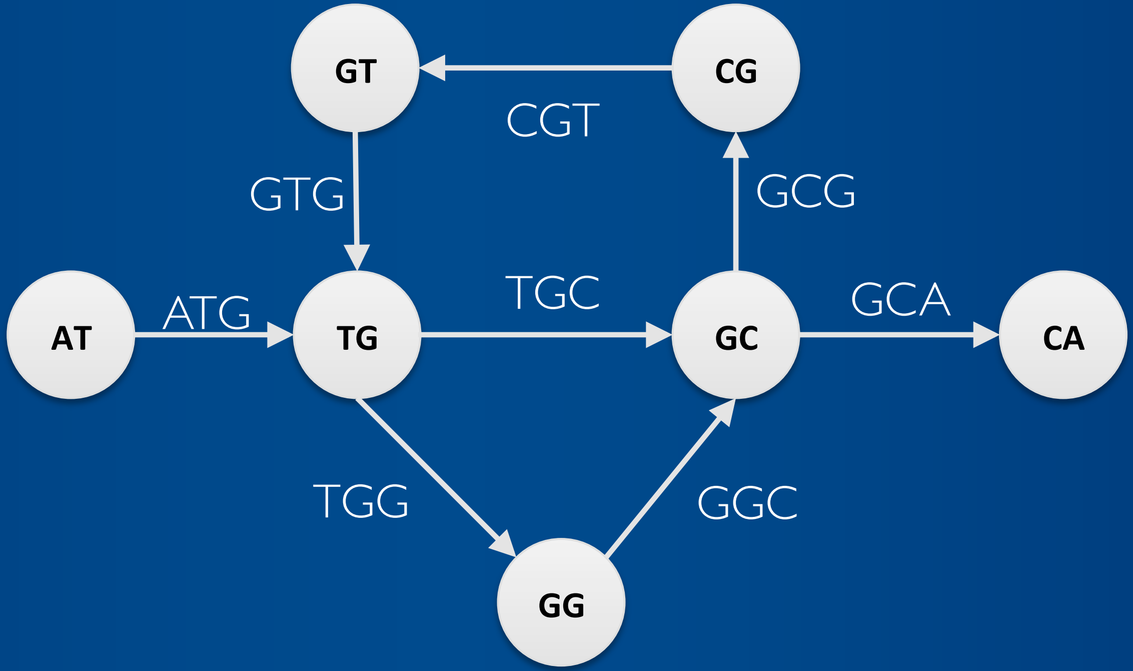
We construct a ***de Bruijn graph***:

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

Distinct (k-1)-mers:



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

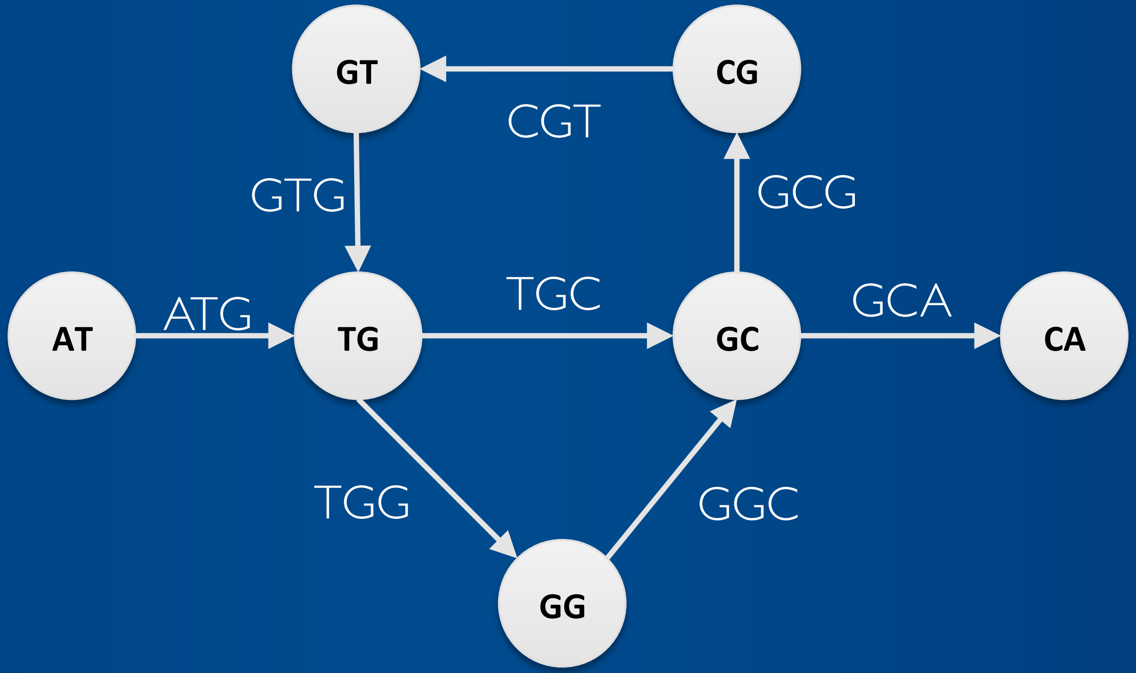
→ In a *de Bruijn* graph, can we find a path that visits every edge of the graph exactly once?

→ 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:


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

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

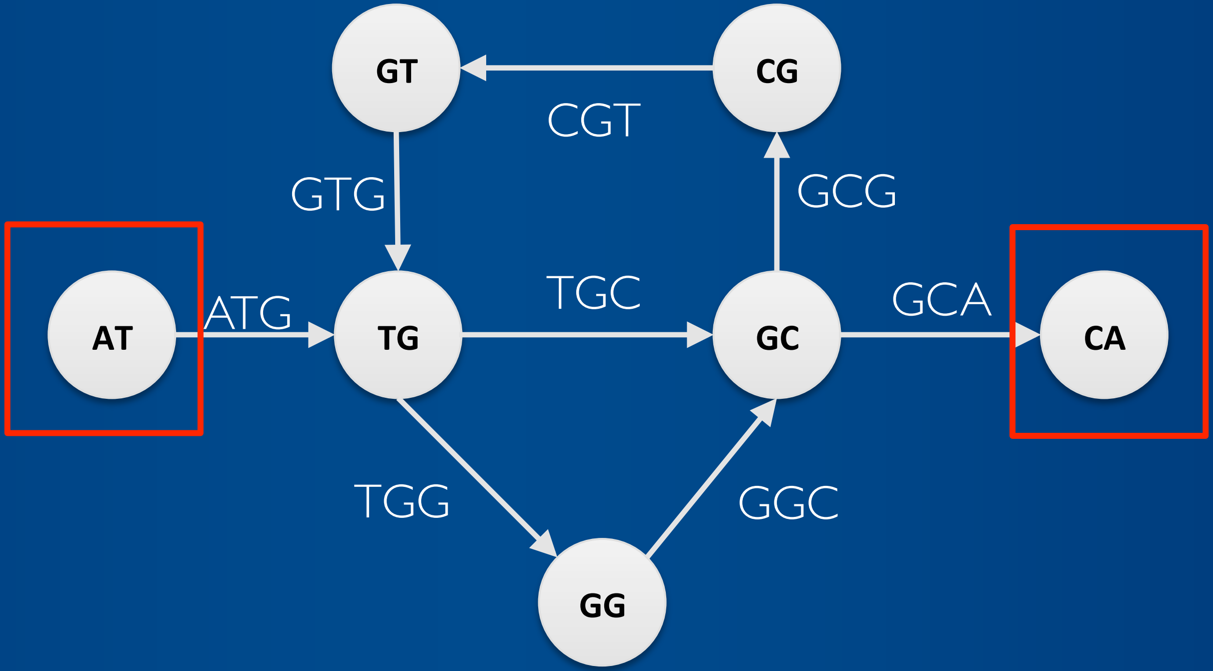
→ In a *de Bruijn* graph, can we find a path that visits every edge of the graph exactly once?

→ 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:



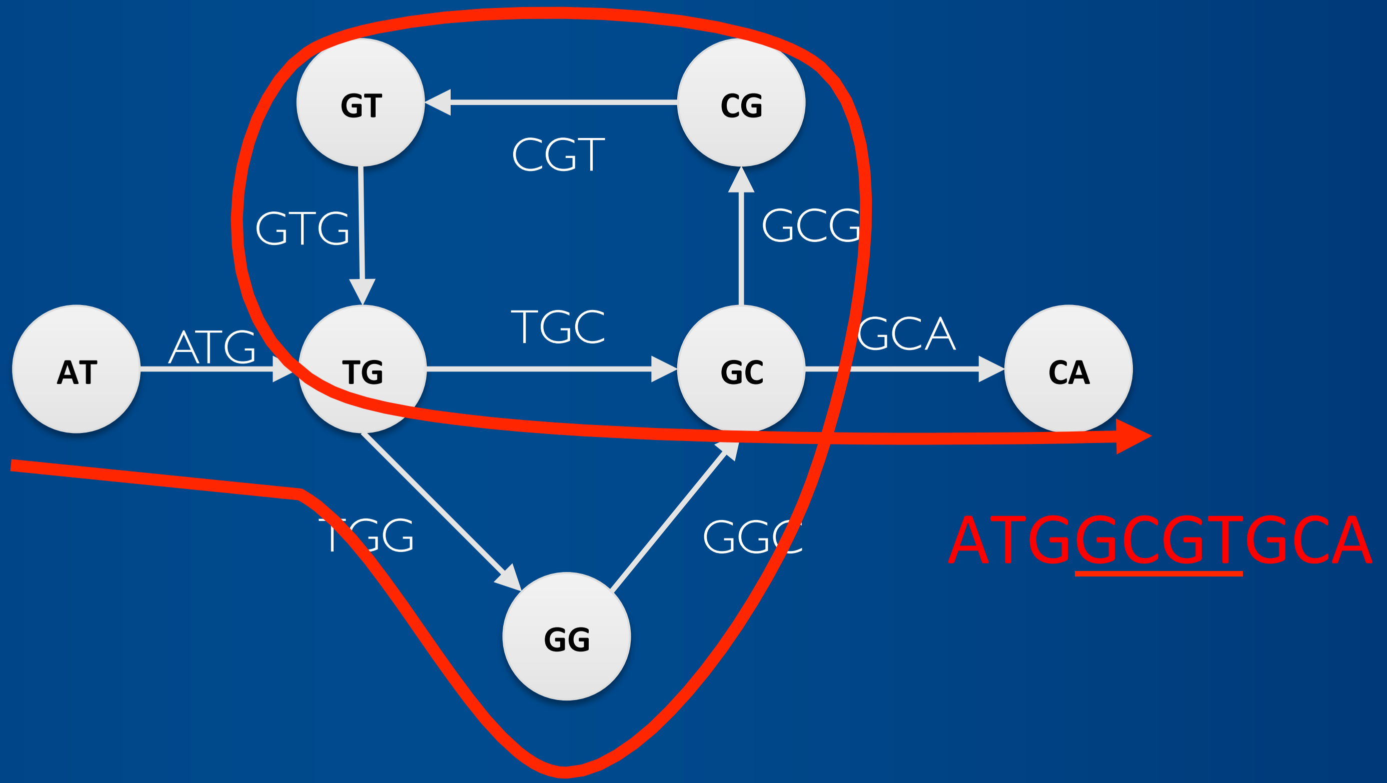
Can we find a DNA sequence containing all *k*-mers?

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

→ Eulerian path

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

Distinct (k-1)-mers:



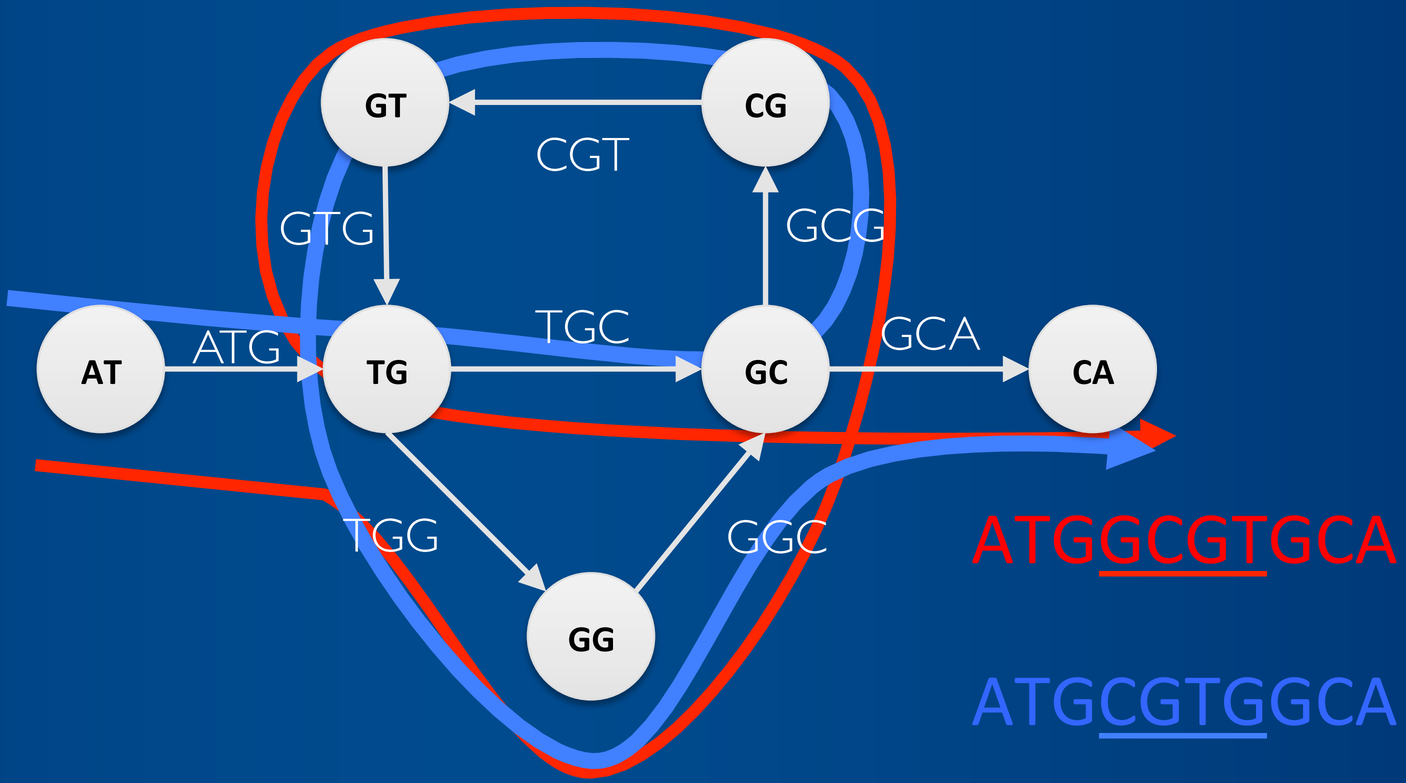
Can we find a DNA sequence containing all *k*-mers?

→ 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:



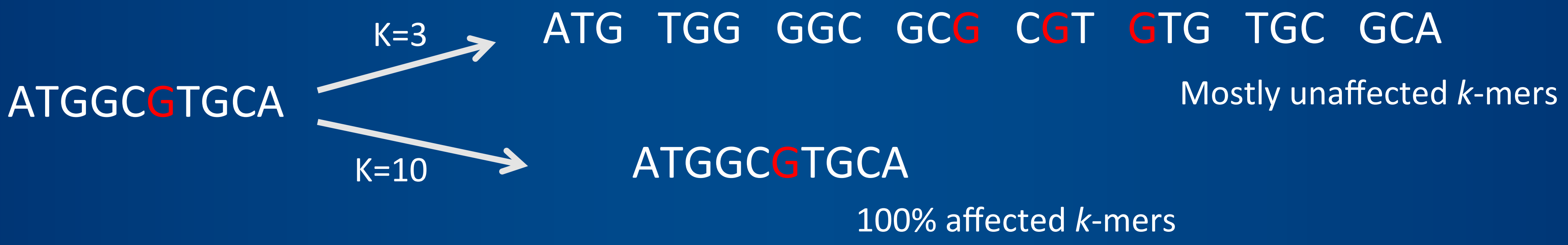
Underlying assumptions of genome assemblies

- 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 of genome assemblies

- 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

- Due to these reasons 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:

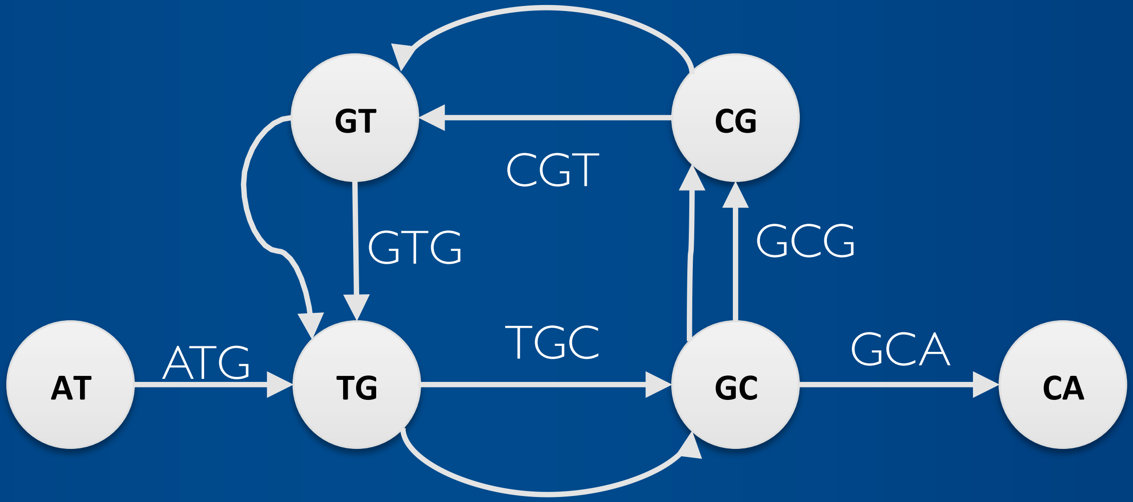


Each *k*-mer appears at most once in the genome → repeats

- 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

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.com/bioinf-workshop/

THE ENGINE
OF THE NEW
NEW ZEALAND