Bioinformatics Algorithms COS-BIOL-530/630 Lecture12

Days & Times	**Room**	**Meeting Dates**
Tu 2:00PM - 3:50PM	Thomas Gosnell Hall (GOS)-2178	01/13/2025 - 04/28/2025
Th 2:00PM - 3:50PM	Thomas Gosnell Hall (GOS)-2178	01/13/2025 - 04/28/2025

Instructor:
Fernando Rodriguez
email: frvsbi@rit.edu
Office: Orange Hall 1311

De Bruijn Graphs
- Lecture12-

Announcements

Lecture12

Lab 12: Genome Assembly in Oedipus

- Discussion12
- Activity12 (*Edited due date assignments)

Quiz10: Lecture/Lab12 + Lecture13: Opens Friday April 23rd.

Exam 2: Thursday , April 17th 2pm (GOS)-2178
Lecture/Lab 07 to Lecture/Lab 11

- Gene ontologies
- Algorithms and pattern matching
- RNA secondary structure
- Gene prediction
- Sequencing technologies (HTS Intro)

De Bruijn Graphs
- Lecture12-

Topics:

- Sanger *vs.* High Throughput Sequencing Assembly
- Graph Theory / Eulerian cycles
- de Bruijn graphs

The Human Genome ~~Project~~ Battle

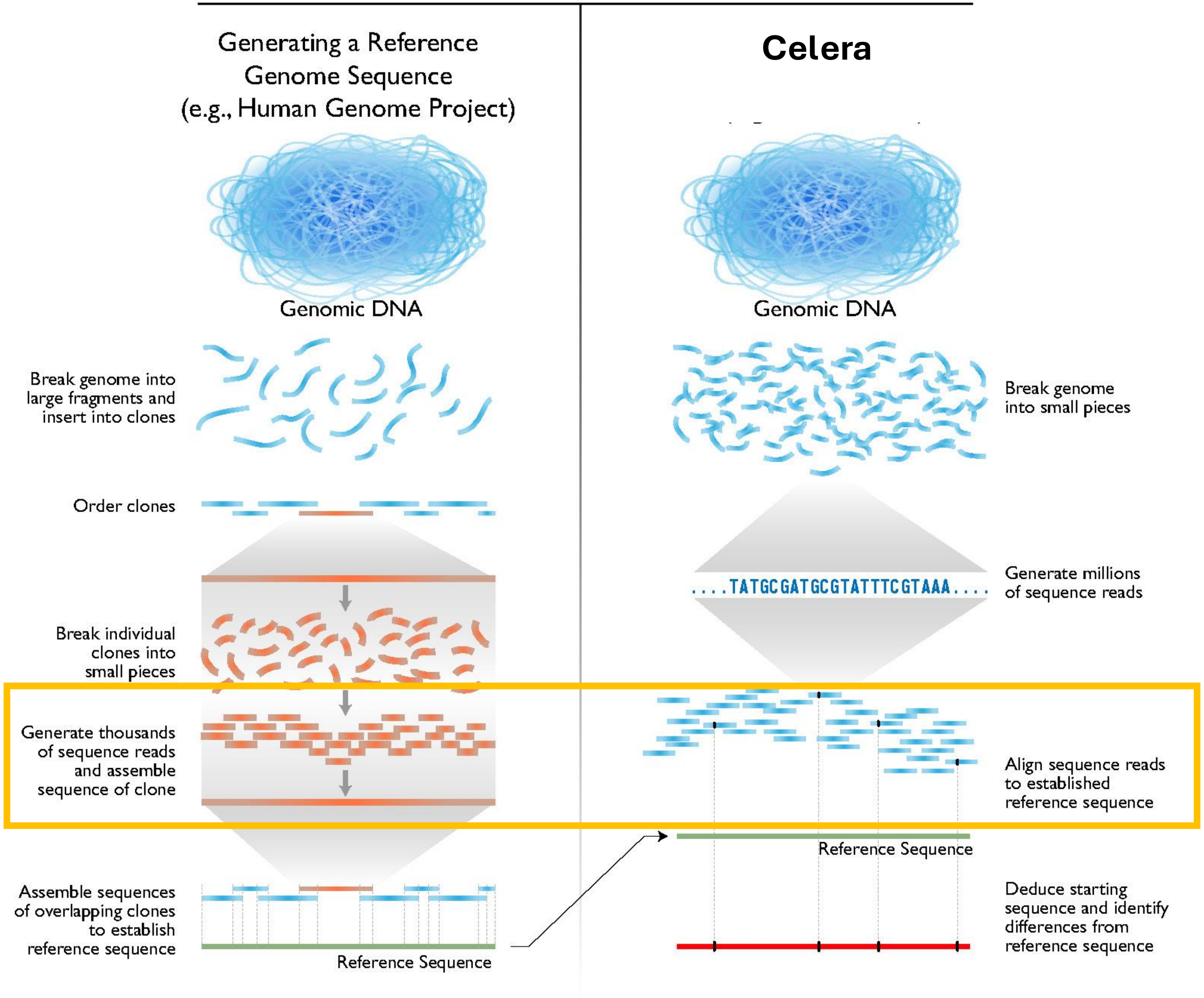
Public consortium:

- $3 billion
- Hierarchical approach (Lander et al. 2001)
- Initial step of clone-based physical mapping to generate a list of overlapping BAC clones to be sequenced ($$$$)
- Followed by shotgun sequencing of the individually mapped BAC clones (100 kb – 200 kb)

Celera:

- $300 million
- It was a proof-of-concept
- Whole genome shotgun approach
- No clones: generates data more quickly
- Direct sequencing of randomly sheared genomic DNA ($$).
- Libraries insert size: 2, 10 and 50 Kb
- Computationally expensive

Human Genome Sequencing



Generating a Reference Genome Sequence (e.g., Human Genome Project)

Genomic DNA

Break genome into large fragments and insert into clones

Order clones

Break individual clones into small pieces

Generate thousands of sequence reads and assemble sequence of clone

Assemble sequences of overlapping clones to establish reference sequence

Reference Sequence

Celera

Genomic DNA

Break genome into small pieces

Generate millions of sequence reads

Align sequence reads to established reference sequence

Reference Sequence

Deduce starting sequence and identify differences from reference sequence

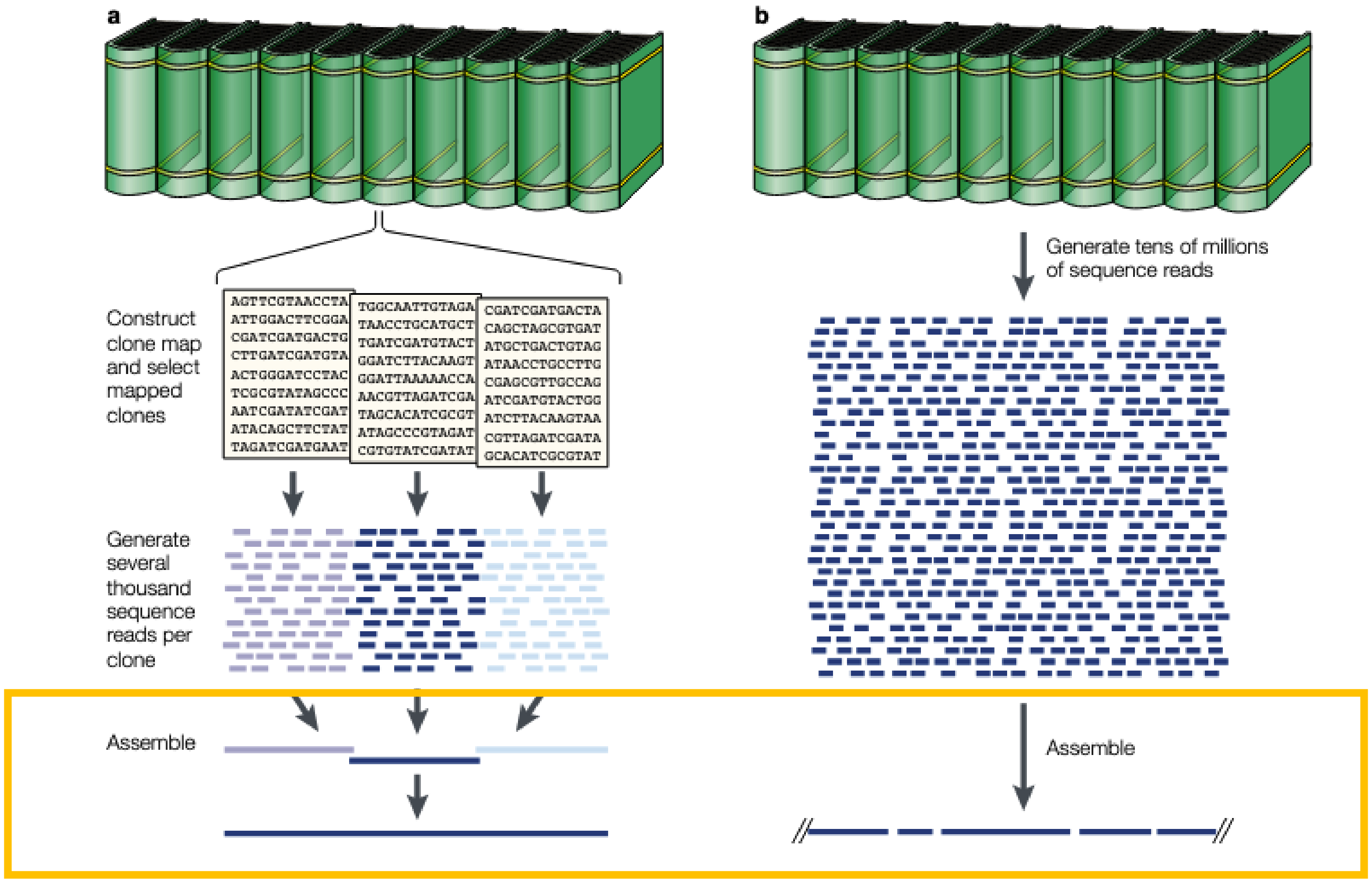
Celera used more computational power and more sophisticated (proprietary) algorithms, but it was based on the same algorithm the Human Genome Project was using.

Computational issues arise from alignment-based assembly.

Two main shotgun-sequencing strategies.

clone-by-clone shotgun sequencing

whole-genome shotgun sequencing

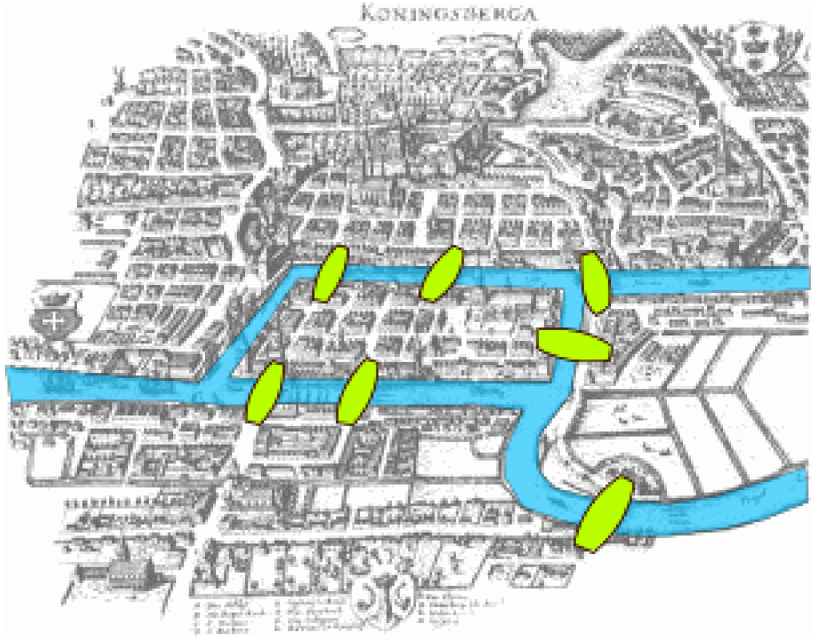


Source: http://dx.doi.org/10.1038/35084503

Assembling billions of reads into a contiguous genome: Challenge Accepted

- The development of algorithmic ideas for High Throughput Sequencing can be traced back 300 years to the Prussian city of Königsberg (present-day Kaliningrad, Russia).

- **Bridges of Konigsberg problem**:
  - Seven bridges joined the four parts of the city.
  - Residents, who enjoyed strolling through the city, wondered "**is it possible to visit every part of the city by walking across each of the seven bridges exactly once and returning to one's starting location?**"

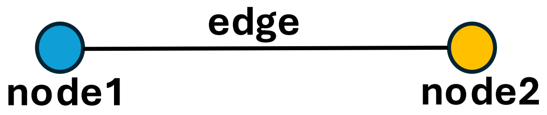


Source: Wikipedia

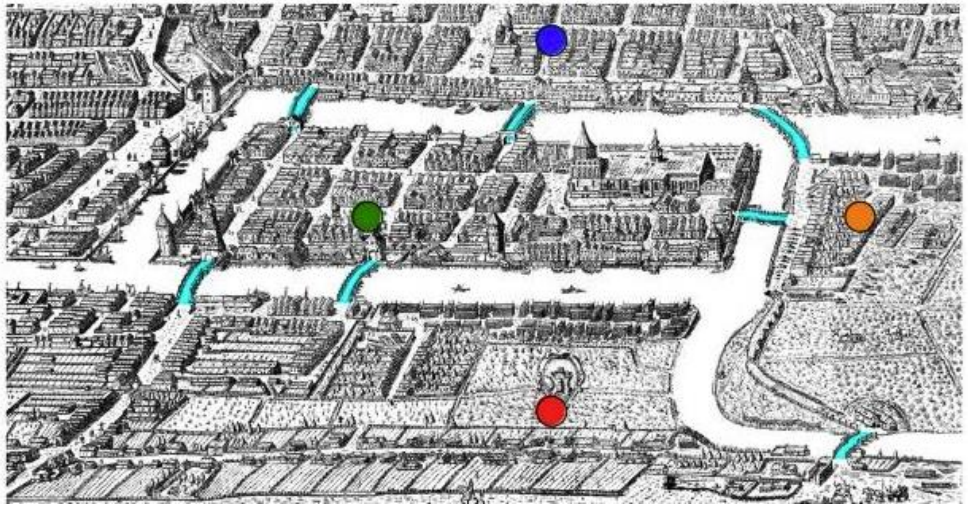
Remarkably, the conceptual breakthrough used by the mathematician Leonhard Euler in 1735 to solve this problem will enable the assembly of billions of short sequencing reads.

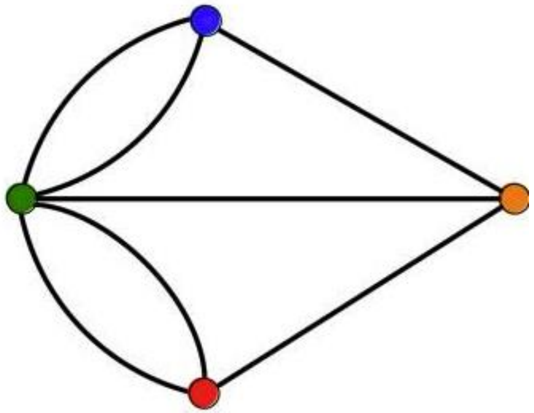
Assembling billions of reads into a contiguous genome: Challenge Accepted

- Euler represented each landmass as a point (**node**), and each bridge as a line segment (**edge**) connecting the appropriate two points.



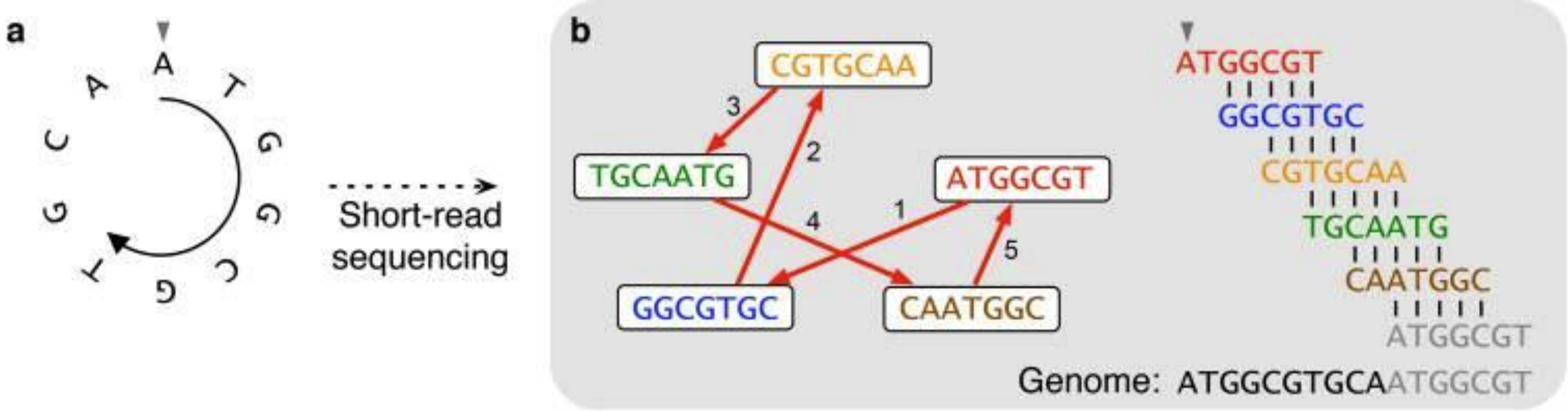
- This creates a **graph** – a network of nodes connected by edges.
- Euler described the procedure for determining if an arbitrary graph contains an **Eulerian cycle:** a path through the graph that visits every edge exactly once and returns back where it started.
- Euler not only *resolved* the Bridges of Königsberg Problem but also effectively launched the entire branch of mathematics known today as **graph theory**.





Why are graphs useful for genome assembly?

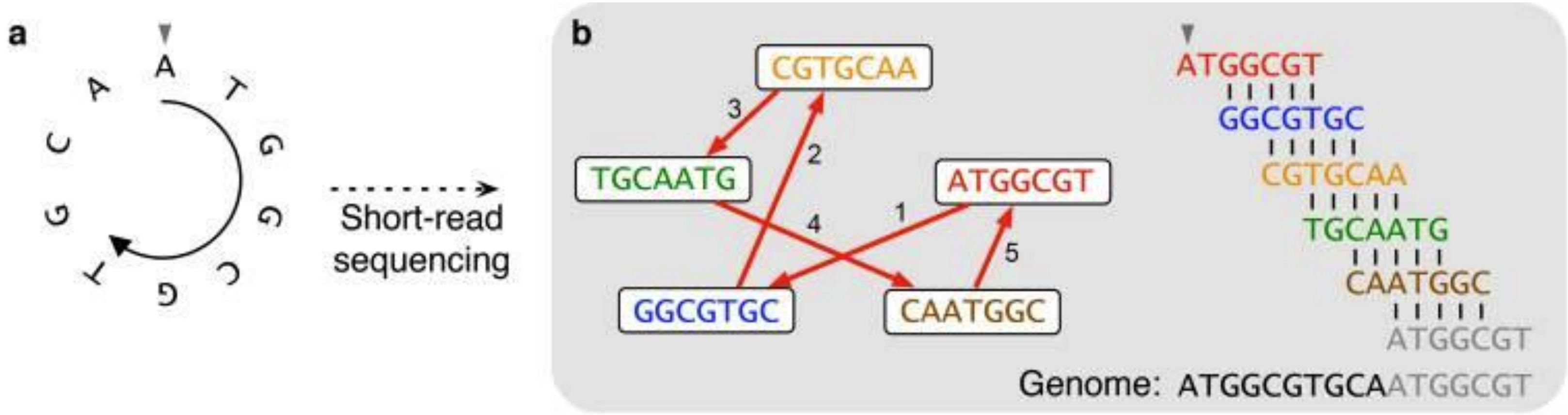
- Suppose a really small circular genome.
- With Sanger sequencing, we produce a few short reads.
- A method for assembling reads uses graphs
  - Each read is represented by a **node**
  - Overlap between reads is represented by an arrow (**directed edge**) joining two reads.
  - *Eg*. two nodes may be connected with a directed edge if the reads overlap 5 bases.



Source: https://doi.org/10.1038/nbt.2023

Why are graphs useful for genome assembly?

- Try to visualize the path (*let's take a stroll*) along the edges of the graph.
- In genome assembly, the path traces a series of overlapping reads -> candidate assembly.
- This path induces a **Hamiltonian cycle** in the graph: a cycle that travels to every **node exactly once** in the assembly.



- The circular *genome* resulting from a Hamiltonian cycle contains all five reads

Source: https://doi.org/10.1038/nbt.2023

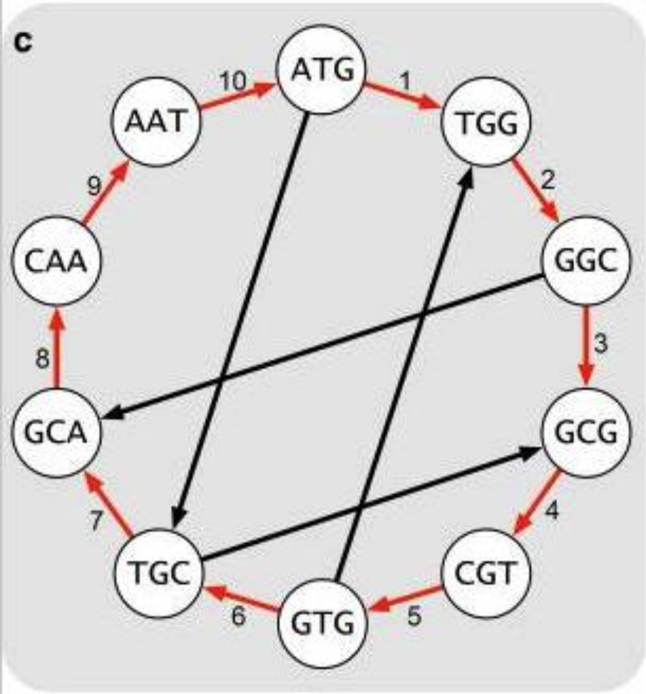
“Old” assemblers

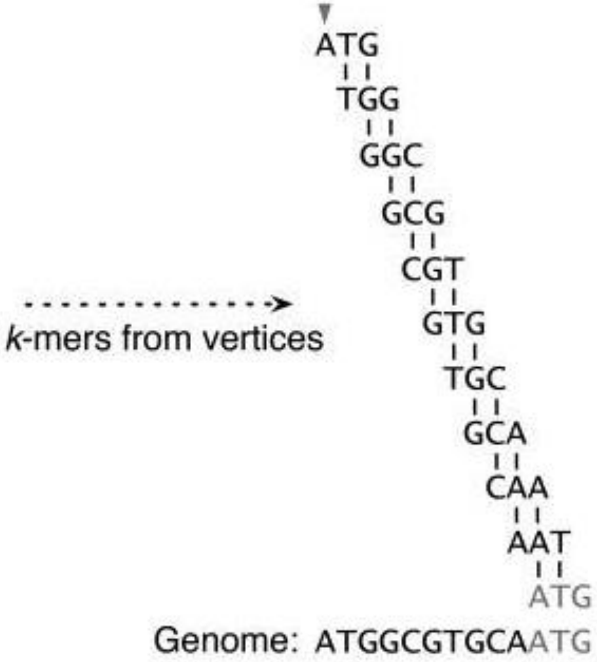
- Assemblers work with strings of length *k* (***k*-mers**):
  - Shorter than reads (eg. an Illumina 100bp read can be divided into 46 overlapping 55-mers)
- From a set of reads, form a node for every *k*-mer in the reads.
- Connect one *k-mer* to another if the two *k-mers* completely overlap except for one nucleotide at each end.
- A **Hamiltonian cycle (red edges)** will represent the candidate genome since it travels each *k-mer* exactly once.

Hamiltonian cycle
Visit each node once

ATGGCGT

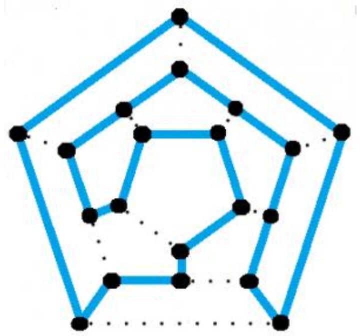
k = 3, comprises
ATG,
TGG,
GGC,
GCG,
CGT





Source: https://doi.org/10.1038/nbt.2023

Hamiltonian cycle



- This method is easy to implement for small genomes/small number of reads.

- The Hamiltonian cycle approach was feasible for sequencing the first microbial genome in 1995 and the human genome in 2001.

- However, the algorithm for finding a Hamiltonian cycle in a large graph with millions of nodes (e.g., Illumina sequencing) would not be efficient.

- The computational burden was so large that most HTS projects have abandoned the Hamiltonian cycle approach.

- Maybe finding a cycle visiting all edges (*bridges*) of a graph once is much easier.

de Bruijn graphs

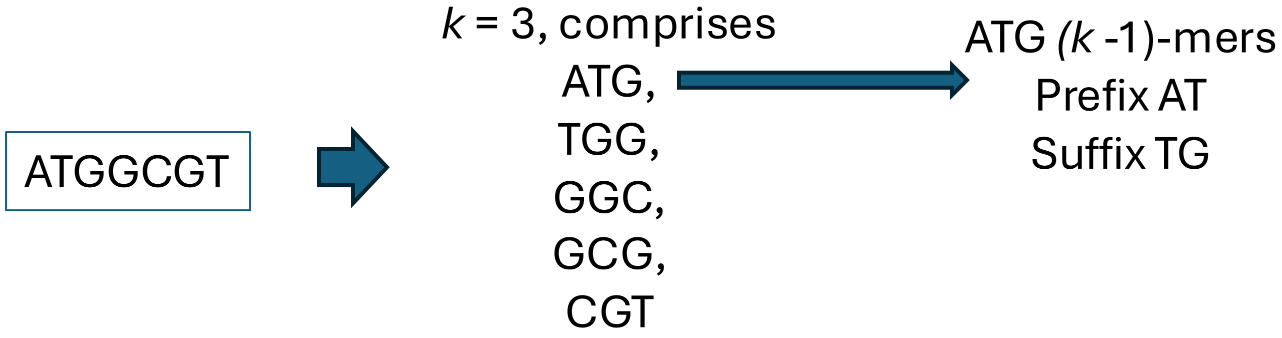
- The origin of de Bruijn graphs dated back to 1946, when the mathematician Nicolaas de Bruijn became interested in the *Superstring Problem*: find a shortest circular *superstring* that contains all possible *substrings* of length *k* (*k*-mers) in a given alphabet.

- In an alphabet containing ***n*** symbols, there are $\mathbf{n}^{k}$ *k-mers*

- In DNA: the alphabet {A,T,G,C}, for *k* = 3 there are 4^3 = 64 trinucleotides.

- The de Bruijn graph represents all *k-mer* prefixes and suffixes as nodes and draws edges representing *k-mers* with a particular prefix and suffix.

- For example, *k*-mer edge ATG has prefix AT and suffix TG



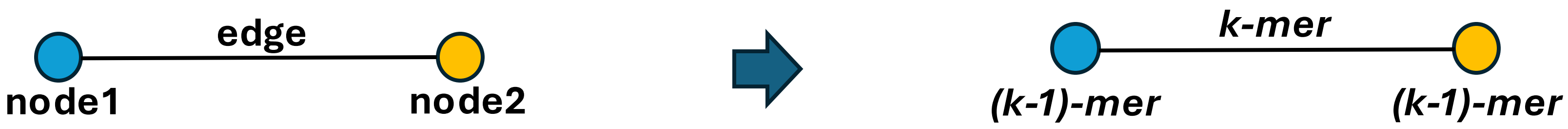
ATGGCGT

k = 3, comprises
ATG,
TGG,
GGC,
GCG,
CGT

ATG *(k* -1)-mers
Prefix AT
Suffix TG

de Bruijn graphs

- The biologist's decision to fragment the genome into smaller pieces motivated computer scientists to cast fragment assembly as such a problem.
- Instead of assigning each *k-mer* to a node, they assign each *k-mer* (within a read) to an edge.
- In the construction of a de Bruijn graph, every possible (*k-1*)-mer is assigned to a node.
- Connect one (*k-1*)-mer by a directed edge to a second (*k-1*)-mer if there is some *k*-mer whose prefix is the former and whose suffix is the latter.
- A crucial aspect of the de Bruijn graph is that it is an **Eulerian graph**, which implies it has an **Eulerian cycle.**
- An Eulerian cycle implies that there exists an Eulerian path that visits each edge exactly once and returns to the starting point without visiting any vertex twice.



de Bruijn graphs

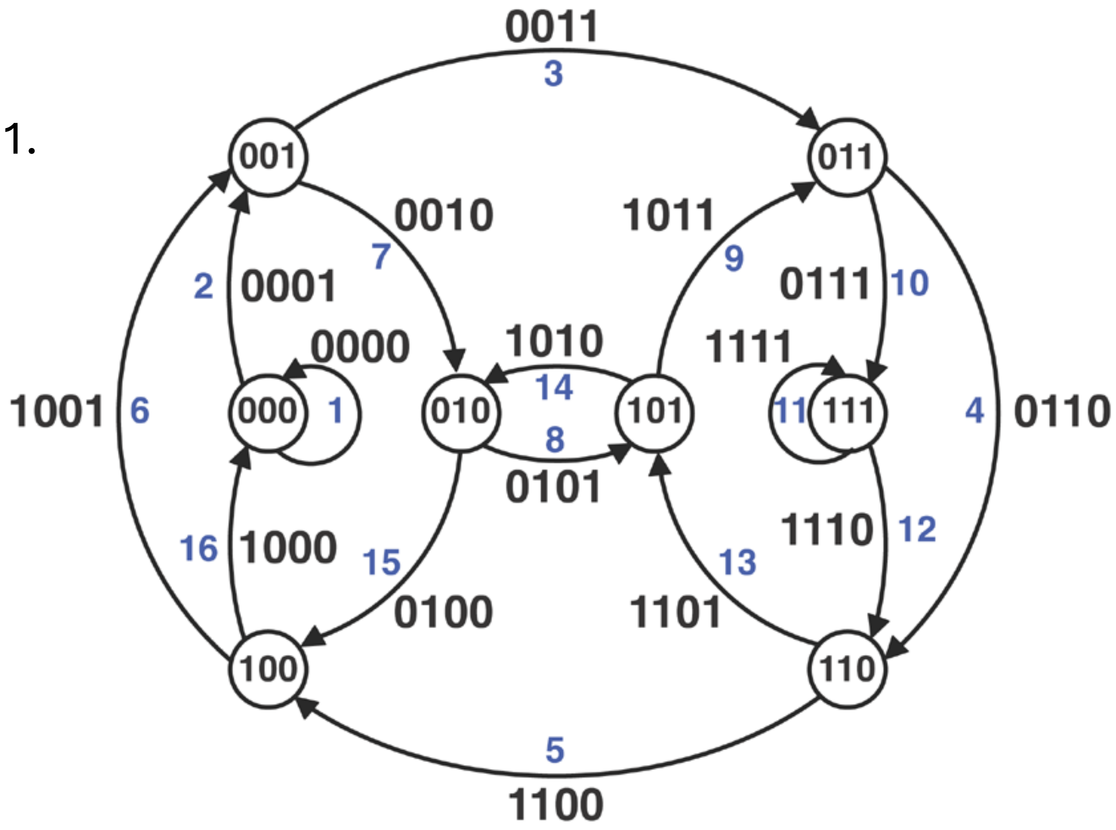
- two-character alphabet: **0** and **1**

- All possible *3-mers*: 000, 001, 010, 011, 100, 101, 110, 111.

- The circular superstring 0001110100 :
  - contains all 3-mers
  - each 3-mer exactly once (graph short as possible)

- how can one construct such a superstring for all *k*-mers
  - if an arbitrary value of *k?*
  - an arbitrary alphabet?

- De Bruijn answered this question by borrowing Euler's solution of the Bridges of Königsberg problem.

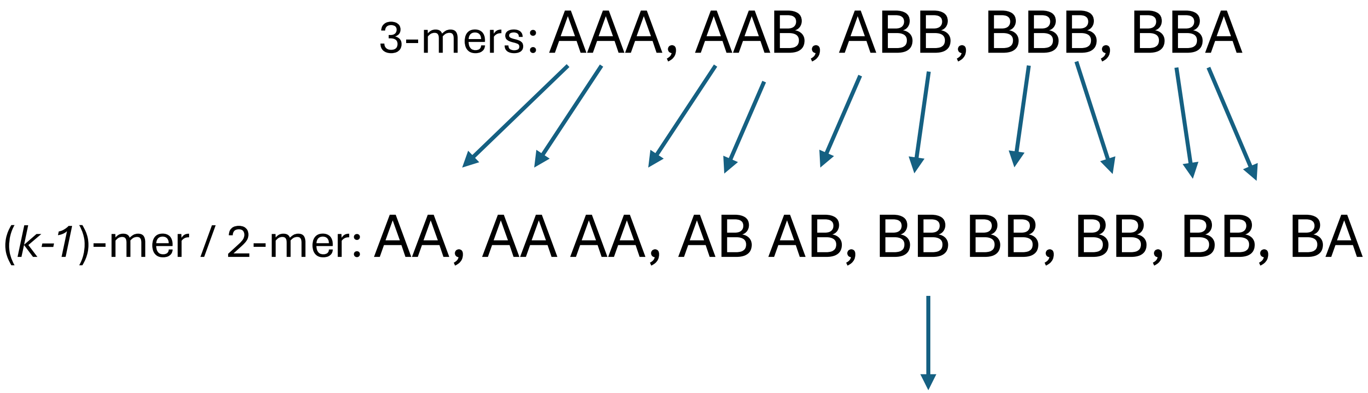


Source: https://doi.org/10.1038/nbt.2023

de Bruijn graphs

- For example, S={A,B} and *k* = 3
- Take each length-3 input string and split it into two overlapping substrings of length 2 (*k-1*). Call these the left and right 2-mers (prefix and suffix).

For a linear genome: AAABBBA and *k* = 3



3-mers: AAA, AAB, ABB, BBB, BBA

(*k-1*)-mer / 2-mer: AA, AA AA, AB AB, BB BB, BB, BB, BA

Represent *k-mer* in a graph:

- One edge per k-mer
- One node per distinct (*k-1*)-mer

Source: Ben Langmead, JHU

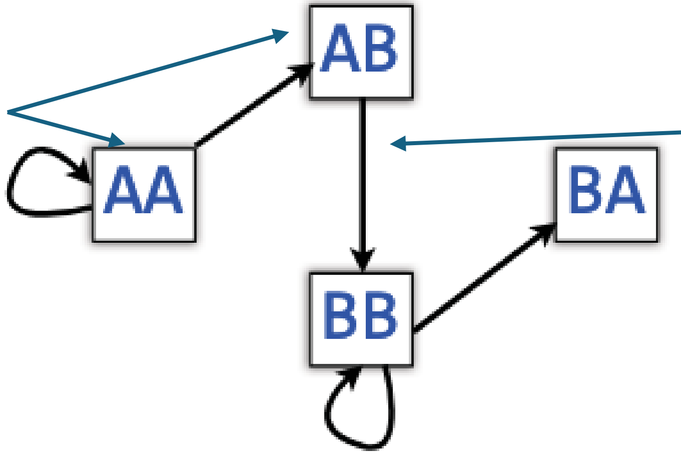
de Bruijn graphs

- For example, S={A,B} and k = 3
- Take each length-3 input string and split it into two overlapping substrings of length 2. Call these the left and right 2-mers (prefix and suffix).

For a genome: AAABBBA

3-mers: AAA, AAB, ABB, BBB, BBA

(k-1)-mer / 2-mer: AA, AA AA, AB AB, BB BB, BB, BB, BA



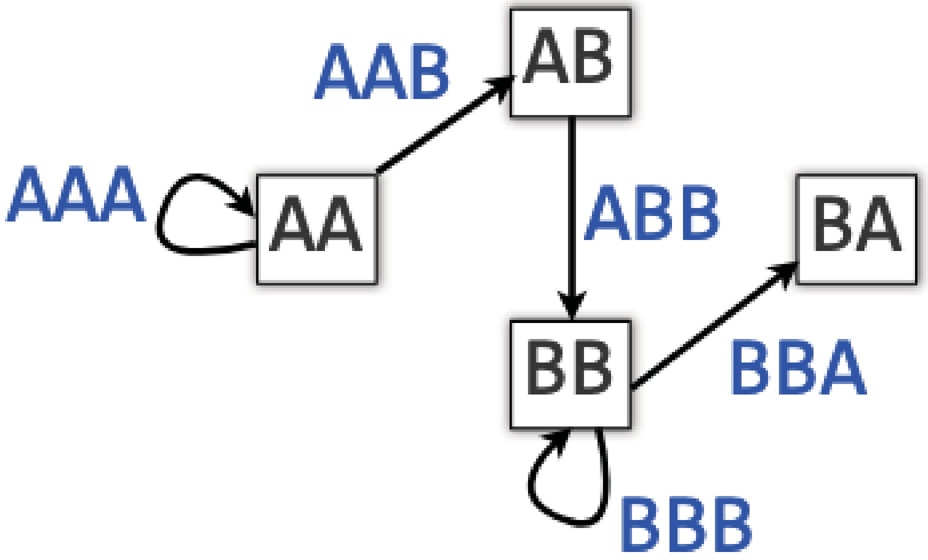
Let 2-mers be nodes in a new graph. Draw a directed edge from each left 2-mer to corresponding right 2-mer.

Each edge in this graph corresponds to a length-3 input string.

Source: Ben Langmead, JHU

de Bruijn graphs

An edge corresponds to an overlap (of length k-2) between two *k-1* mers.
More precisely, it corresponds to a *k-mer* from the input.



This is an Eulerian cycle!
Walk crossing each edge exactly once gives a reconstruction of the genome
AAABBBA

de Bruijn graphs

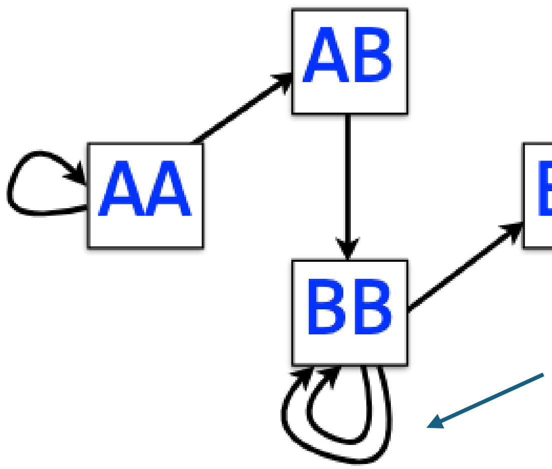
- If we add one more B to our input string:

For a genome: AAABBBBA

3-mers: AAA, AAB, ABB, BBB, BBB, BBA

(*k-1*)-mer / 2-mer: AA, AA AA, AB AB, BB BB, BB BB, BB BB, BA

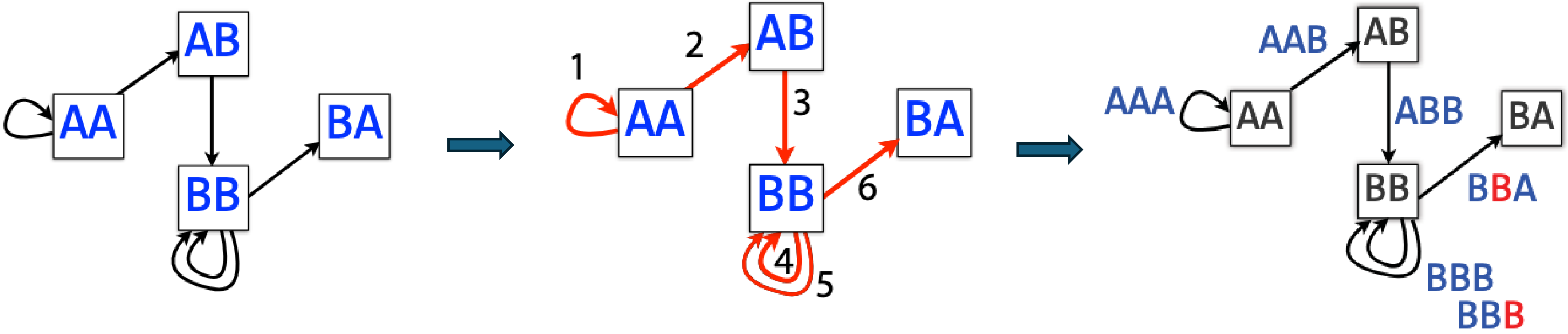
Nodes (2-mers) in a new graph: no change.



Rebuild the De Bruijn graph accordingly: we get a multi-edge.

Source: Ben Langmead, JHU

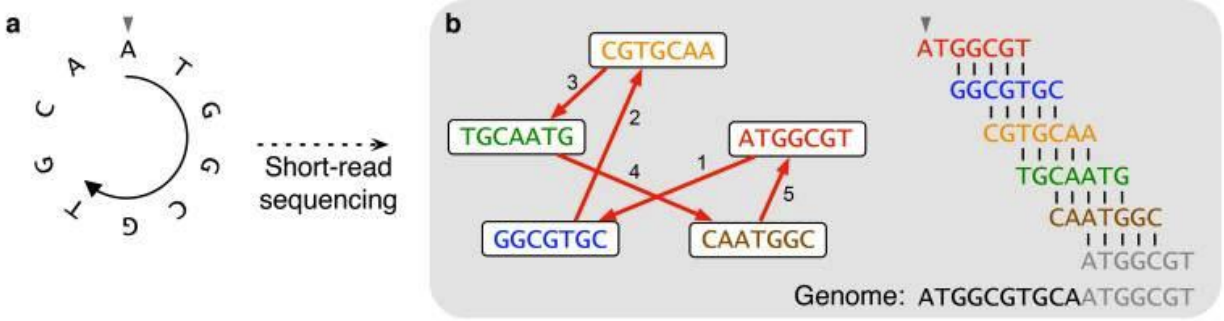
de Bruijn graphs



This is an Eulerian cycle!
Walk crossing each edge exactly once gives a reconstruction of the genome
AAABBBBA

Source: Ben Langmead, JHU

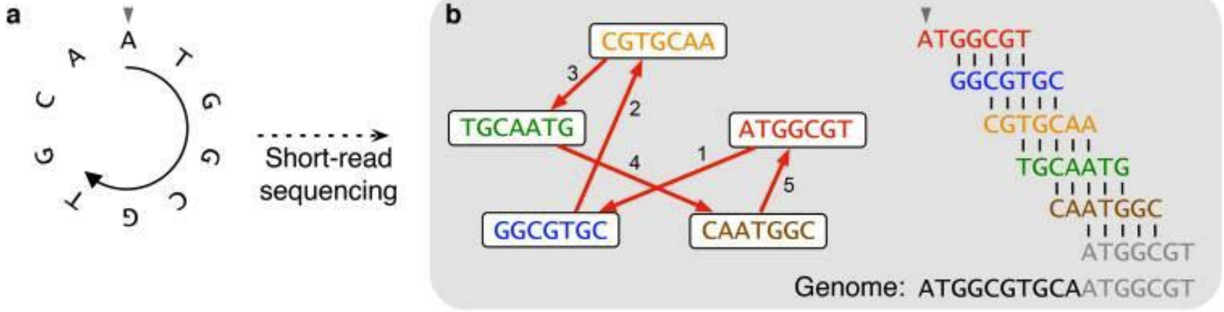
de Bruijn graphs



This time, nodes are (k-1)-mers
Edges are k-mers

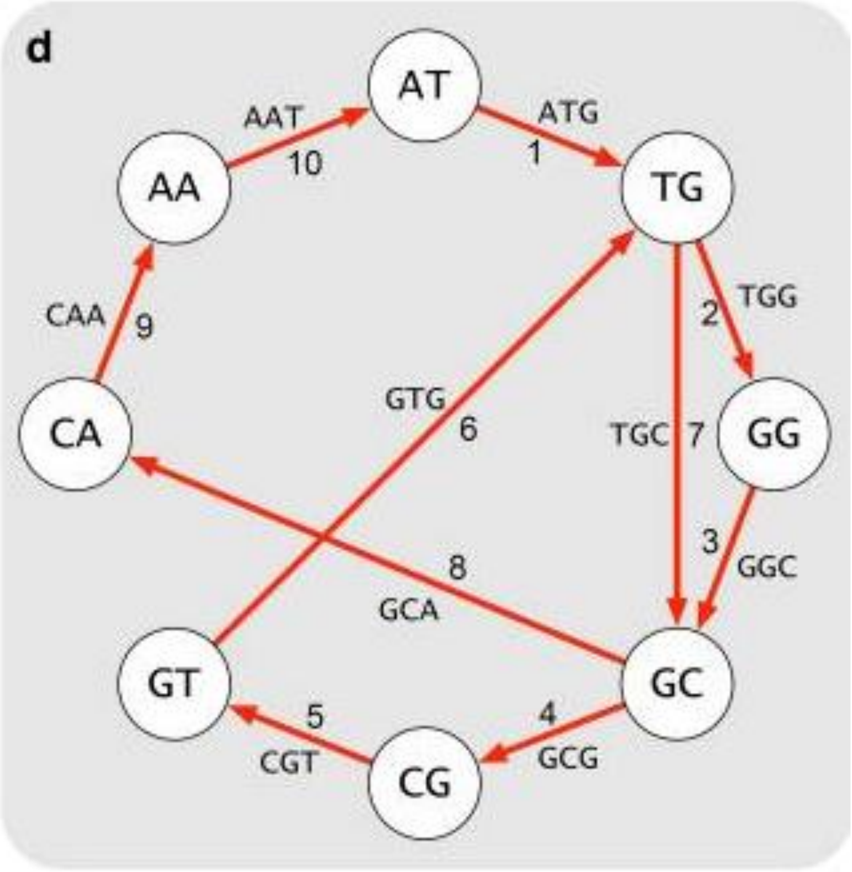
Source: https://doi.org/10.1038/nbt.2023

de Bruijn graphs



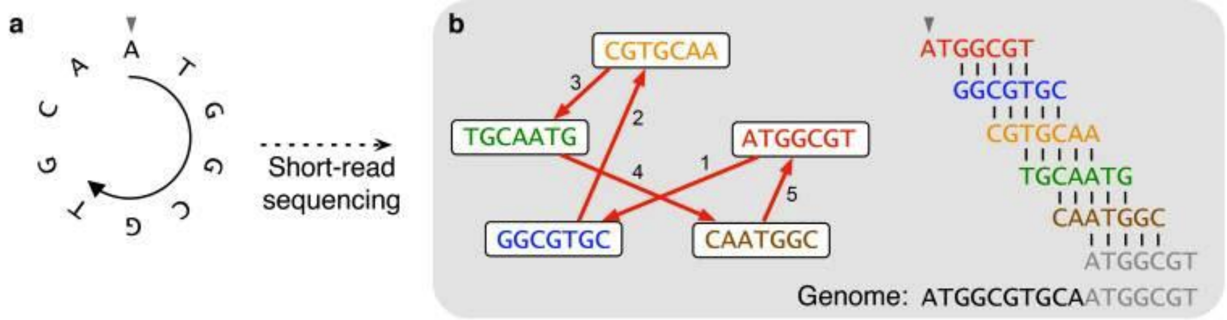
This time, nodes are (k-1)-mers
Edges are k-mers
k = 3

Eulerian cycle
Visit each edge once



Source: https://doi.org/10.1038/nbt.2023

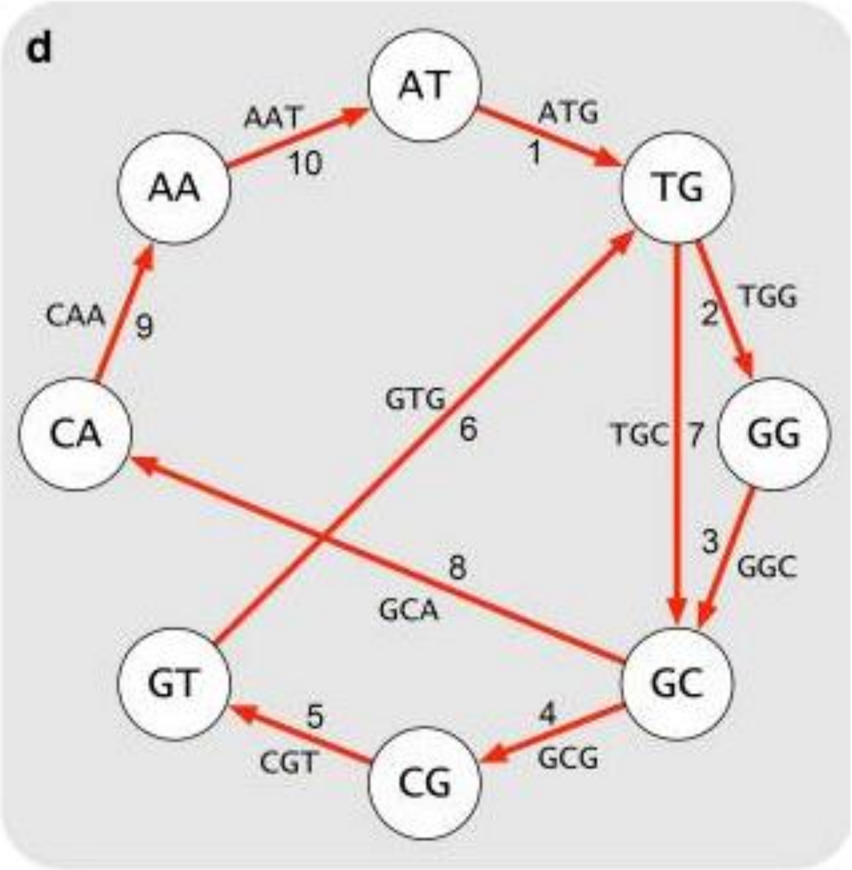
de Bruijn graphs



Easier to solve!

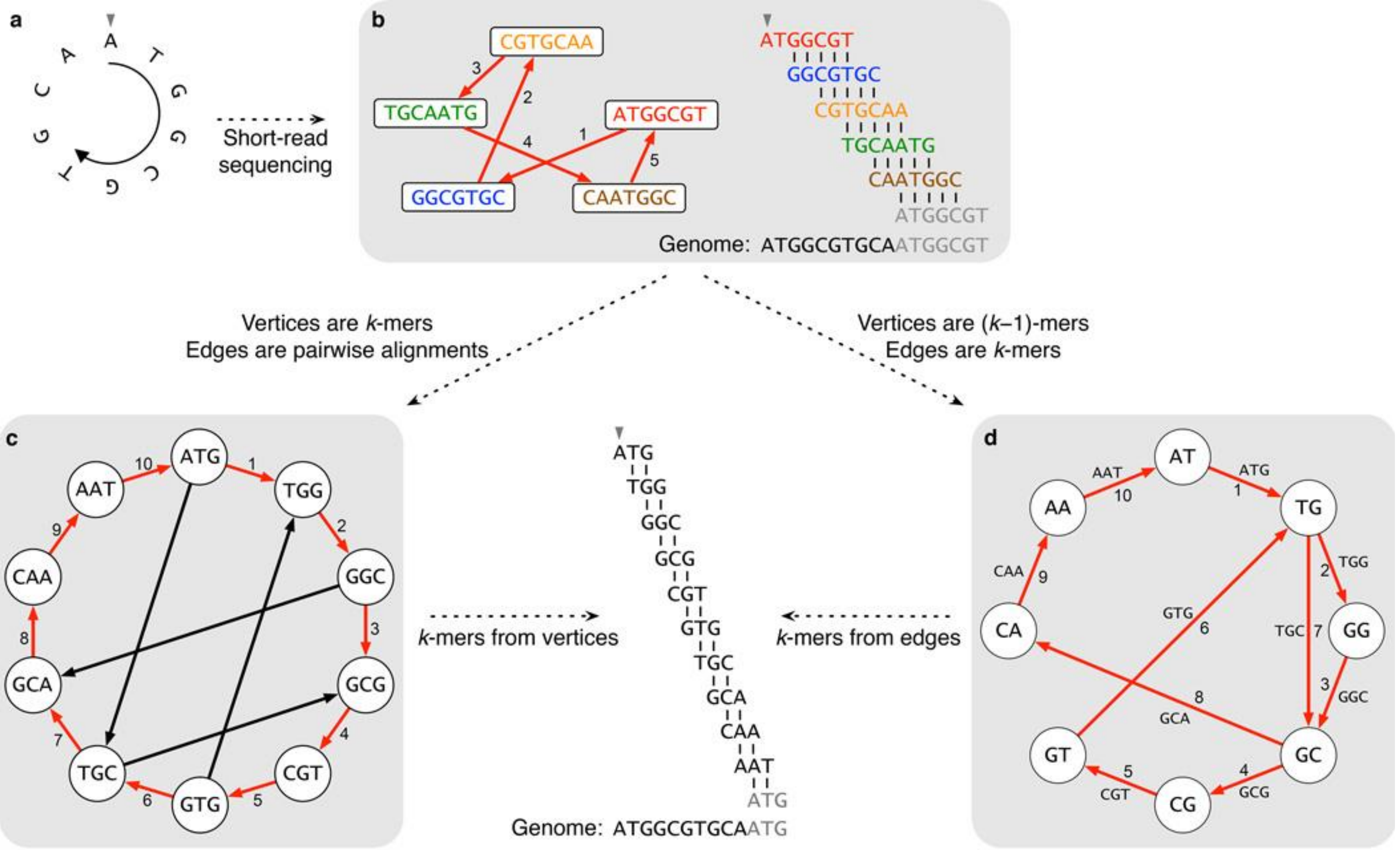
Eulerian cycle
Visit each edge once

This time, nodes are (k-1)-mers
Edges are k-mers



Source: https://doi.org/10.1038/nbt.2023

Two strategies for genome assembly: from Hamiltonian cycles to Eulerian cycles.



Hamiltonian cycle
Visit each vertex once
(harder to solve)

Eulerian cycle
Visit each edge once
(easier to solve)

Source: https://doi.org/10.1038/nbt.2023

de Bruijn graphs

- Modern short-read assembly algorithms construct a de Bruijn graph by representing all *k*-mer prefixes and suffixes as nodes.

- Then drawing edges that represent *k*-mers having a particular prefix and suffix.

- Finding an Eulerian cycle allows one to reconstruct the genome by forming an alignment in which each successive *k*-mer (from successive edges) is shifted by one position.

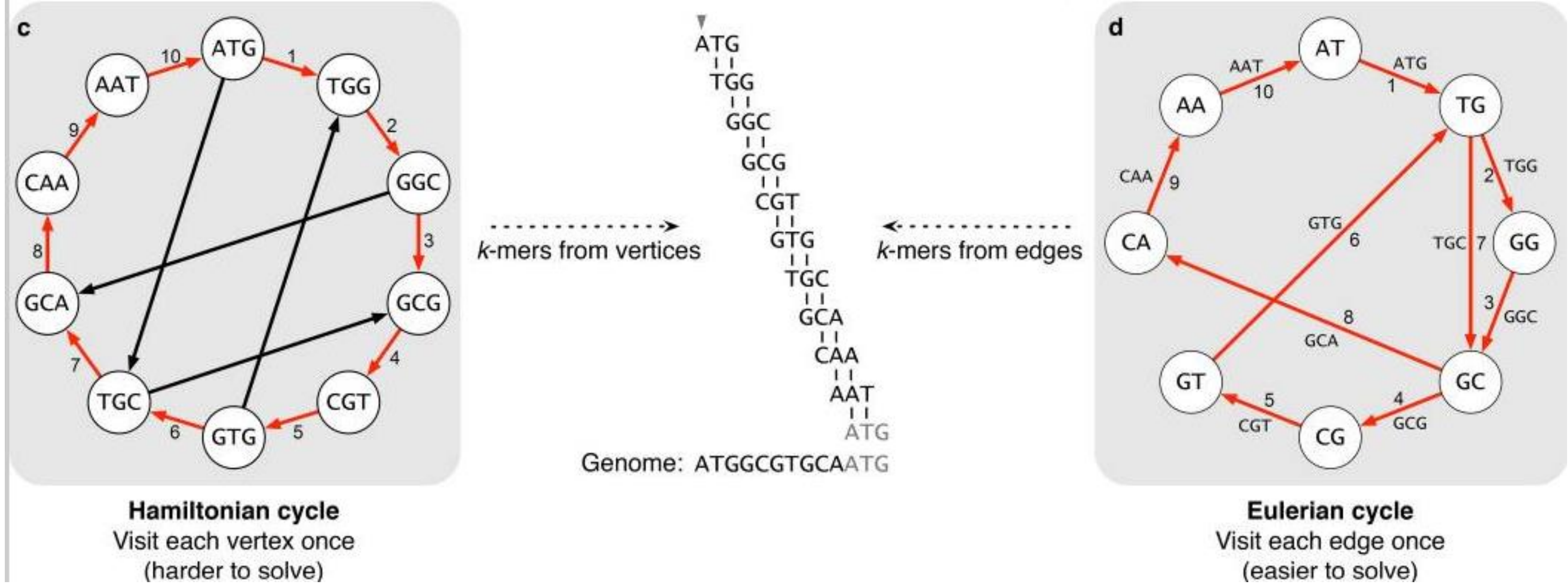
- This generates the same cyclic genome sequence without the computational strain of finding a Hamiltonian cycle.

Two strategies for genome assembly: from Hamiltonian cycles to Eulerian cycles.

- The run time required by a computer implementation of Euler’s algorithm is roughly proportional to the number of edges in the graph.

- In the Hamiltonian approach, the time is potentially a lot larger, due to the large number of pairwise alignments needed to construct the graph, and to the *NP*-Completeness of finding a Hamiltonian cycle.

- The Hamiltonian Path Problem is NP-complete , which means that the efficient algorithms for solving this problem are unknown.



Hamiltonian cycle
Visit each vertex once
(harder to solve)

Eulerian cycle
Visit each edge once
(easier to solve)

Assembly methods

- Current genome sequencing technology can only sequence a tiny portion of a genome in a contiguous read.

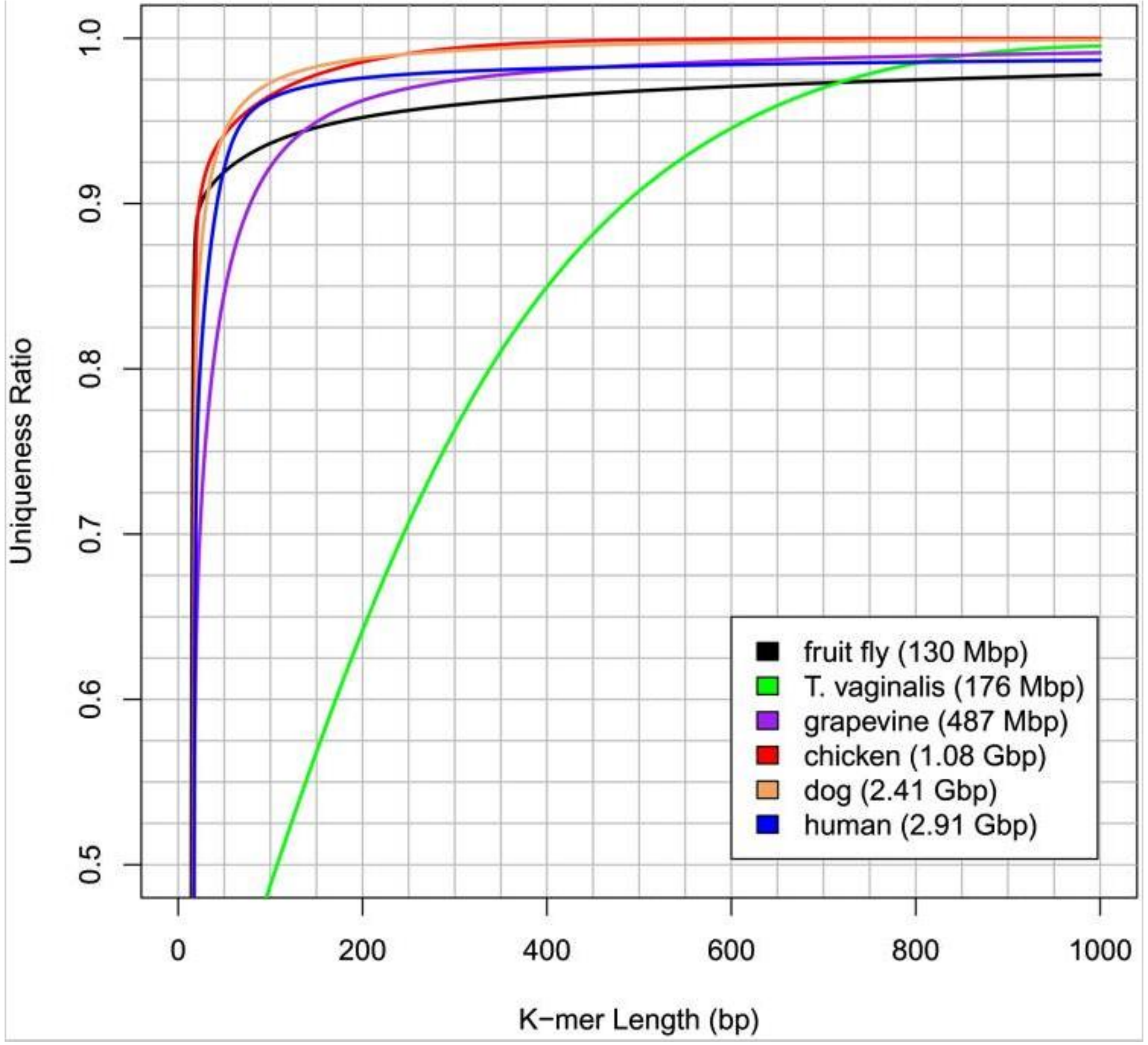
- DNA sequence reads may fit together in more than one way because of repetitive sequences within the genome.

- Assembly methods aim to create the most complete reconstruction possible without introducing errors.

- The central challenge of genome assembly is resolving repetitive sequences.

How can we measure the repetitiveness among species?

Uniqueness ratio for varying read lengths in six genomes



The figure shows how much of each genome would be covered by *k*-mers (reads) that occur exactly once.

Among the multicellular species, dog and chicken are the least repetitive while fly is the most repetitive.

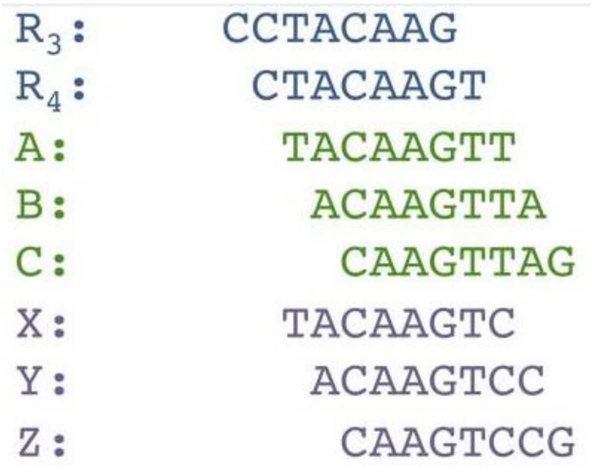
The percentage of a genome covered uniquely increases rapidly as read length increases to 50 bp and above.

But the rate of increase varies due to the variable repeat lengths in different species.

Source: https://doi.org/10.1101/gr.101360.109

Assembly methods

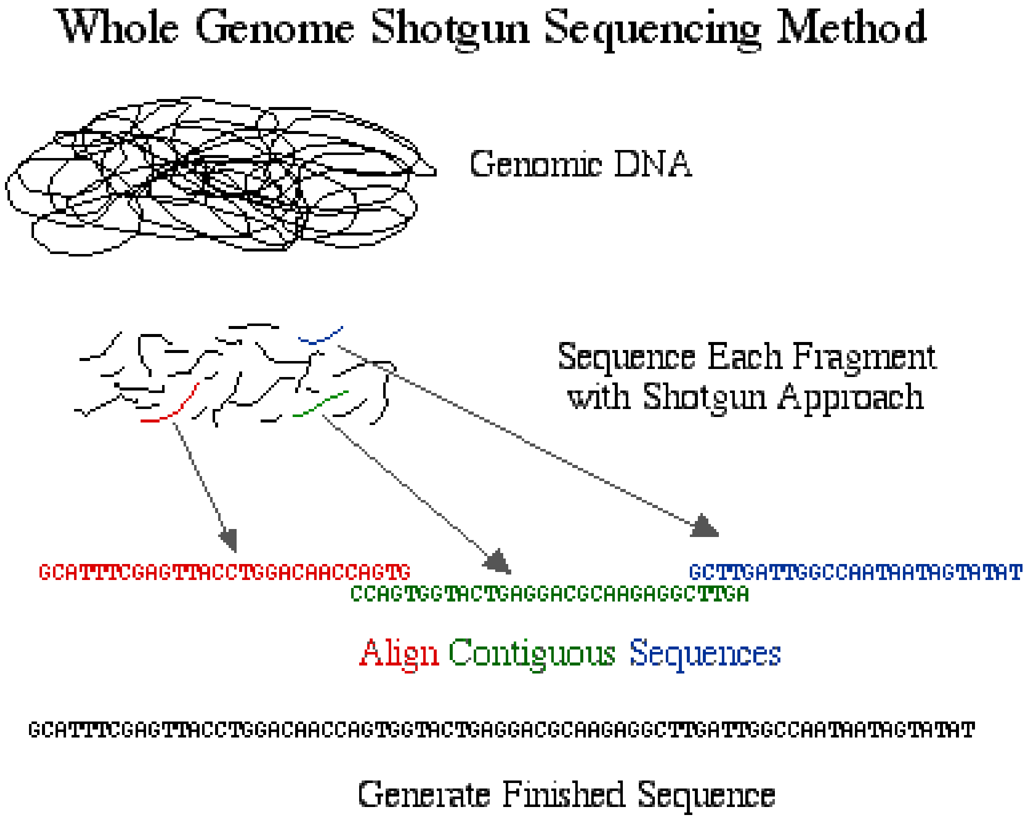
- Early genome assemblers used a simple “greedy” algorithm, in which all pairs of reads are compared with each other, and the ones that **overlap** most are merged first.
- To allow for sequencing errors, assemblers compute these overlaps with a variant of the **Smith-Waterman** algorithm.
- Once all overlaps are computed, the reads with the longest overlap are concatenated to form a contig (contiguous sequence).
- This simple merging process will accurately reconstruct the simplest genomes, but fails for repetitive sequences longer than the read length.



```
R3:  CCTACAAG
R4:   CTACAAGT
A:     TACAAGTT
B:      ACAAGTTA
C:       CAAGTTAG
X:     TACAAGTC
Y:      ACAAGTCC
Z:       CAAGTCCG
```

Large-scale shotgun assembly

- Several assemblers have been developed to assemble large, repetitive genomes from long (Sanger) reads, including the Celera Assembler.


Whole Genome Shotgun Sequencing Method
Genomic DNA
Sequence Each Fragment with Shotgun Approach
GCATTTCGAGTTACCTGGACAACCAGTG
CCAGTGGTACTGAGGACGCAAGAGGCTTGA
GCTTGATTGGCCAATAATAGTATAT
Align Contiguous Sequences
GCATTTCGAGTTACCTGGACAACCAGTGGTACTGAGGACGCAAGAGGCTTGATTGGCCAATAATAGTATAT
Generate Finished Sequence

Celera’s “unique” approach

Source: https://www.ocf.berkeley.edu

Short read assembly

- In principle, assemblers created for long reads should also function for short reads.
- They need to find overlap and build a consensus.
- They even try to accommodate short reads by fine-tune different parameters (overlaps, seed reads, coverage).
- New generation of genome assemblers has been developed specifically to address the challenges of assembling short reads.
- Rather than using an overlap graph, all assemblers use **de Bruijn graph** algorithm.

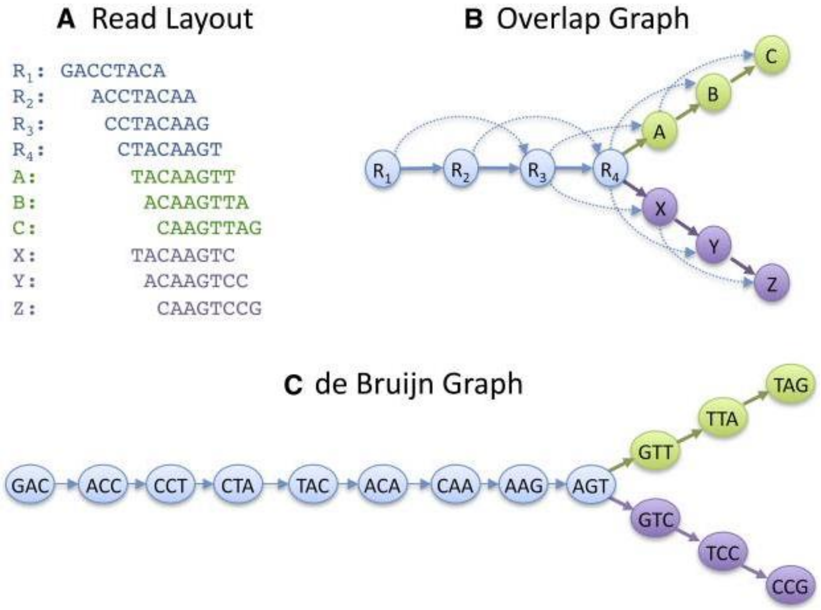
Two approaches to read assembly

Layout - Overlap - Consensus:

- Construct overlap graph directly from reads, eliminating redundant reads.
- Trace path for assembly.
- String graphs assemblers

De Bruijn graph-based assemblers:

- Construct k-mer graph from reads
- Original reads are discarded
- Trace path in graph for assembly



- Differences between an overlap graph and a de Bruijn graph for assembly.
- In both approaches, repeat sequences create a fork in the graph.

Source: https://doi.org/10.1101/gr.101360.109

Limitations of de Bruijn graphs

- Sequencing error complicates the de Bruijn graph (errors create their own graph nodes), but many errors are easily recognized by their structure in the graph : **dead-end** “tips” in the graph.

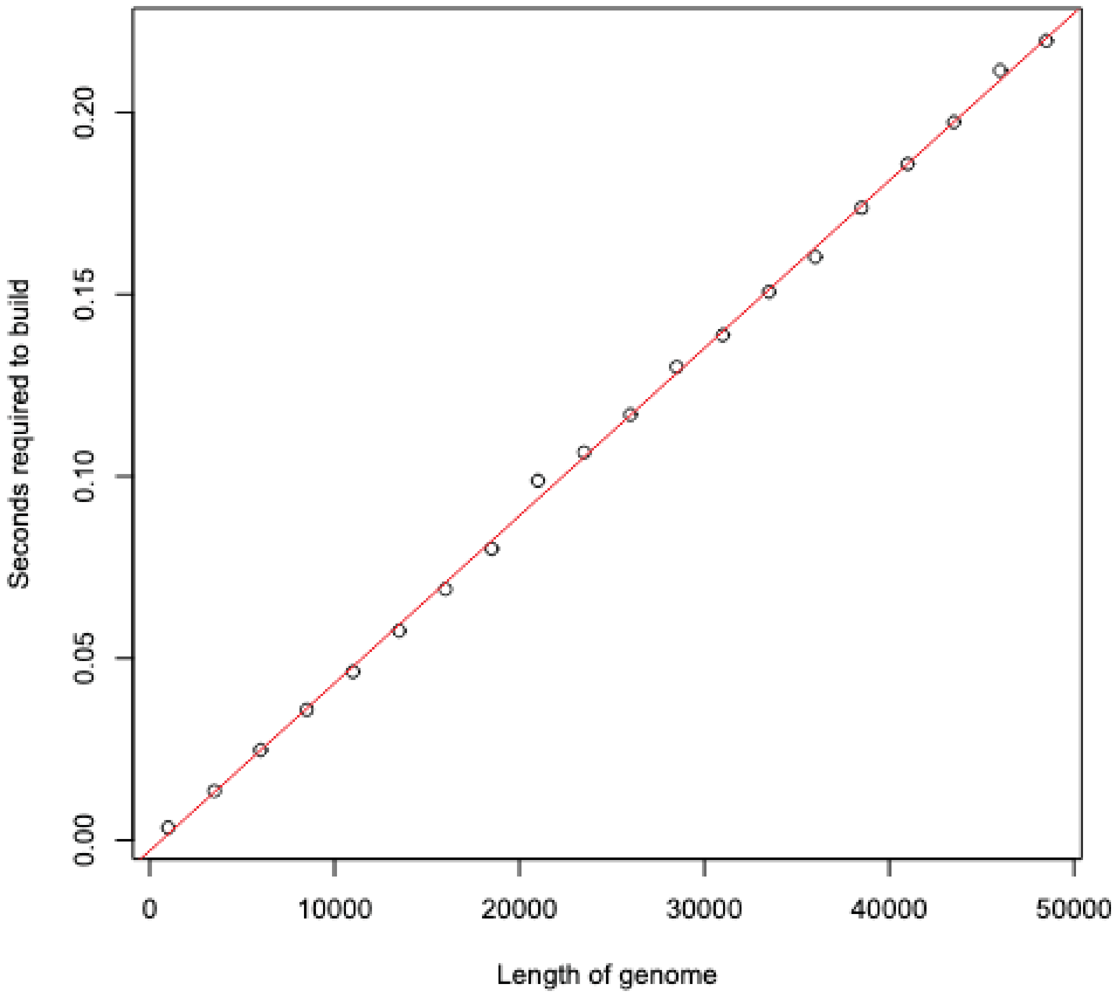
- The main drawback to the de Bruijn approach is the loss of information caused by decomposing a read into a path of *k*-mers.
    - can’t resolve repeats as well as overlap graph.

- Another potential drawback of the de Bruijn approach is that the de Bruijn graph can require an enormous amount of computer space (RAM).

- In theory, the size of the de Bruijn graph depends only on the size of the genome, including polymorphic alleles.

- Single most important benefit of De Bruijn graph is speed and simplicity.

Applying de Bruijn graphs



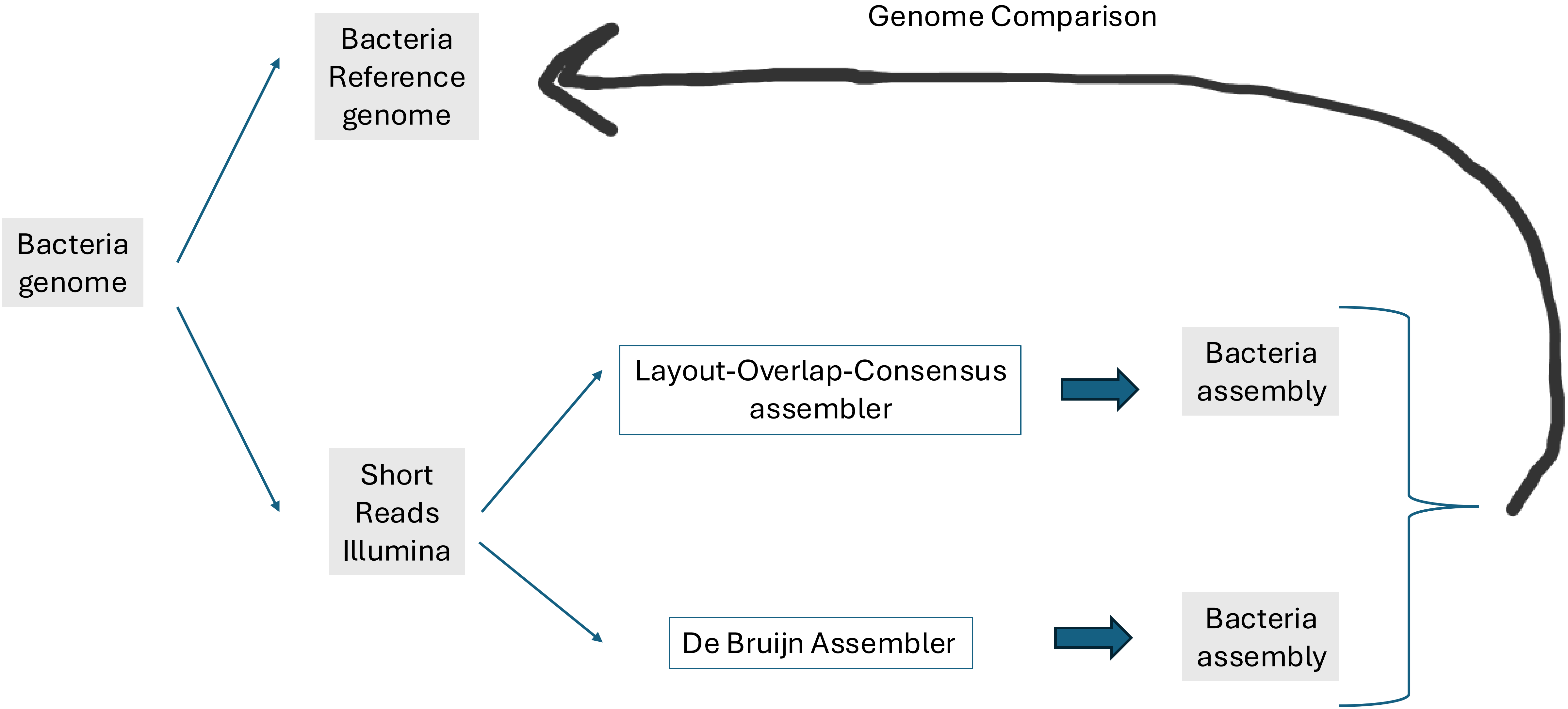
- Timed de Bruijn graph construction applied to progressively longer prefixes of **lambda phage genome** (50 Kb).

- *k* = 14

- *O(N)* expectation appears to work in practice.

Source: Ben Langmead, JHU

Lab12: Genome assembly


Genome Comparison
Bacteria Reference genome
Bacteria genome
Layout-Overlap-Consensus assembler
Bacteria assembly
Short Reads Illumina
De Bruijn Assembler
Bacteria assembly