Graph Algorithms in Bioinformatics

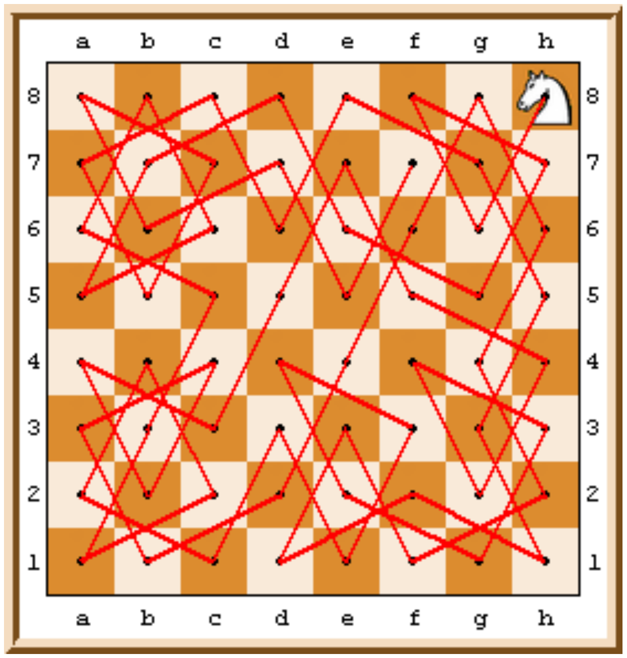
Outline

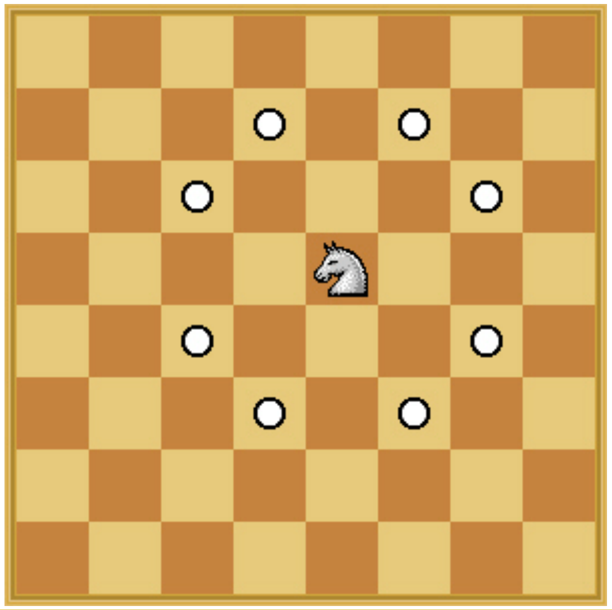
1. Introduction to Graph Theory
2. The Hamiltonian & Eulerian Cycle Problems
3. Basic Biological Applications of Graph Theory
4. DNA Sequencing
5. Shortest Superstring & Traveling Salesman Problems
6. Sequencing by Hybridization
7. Fragment Assembly & Repeats in DNA
8. Fragment Assembly Algorithms

Section 1: Introduction to Graph Theory

Knight Tours

- **Knight Tour Problem**: Given an 8 x 8 chessboard, is it possible to find a path for a knight that visits every square exactly once and returns to its starting square?

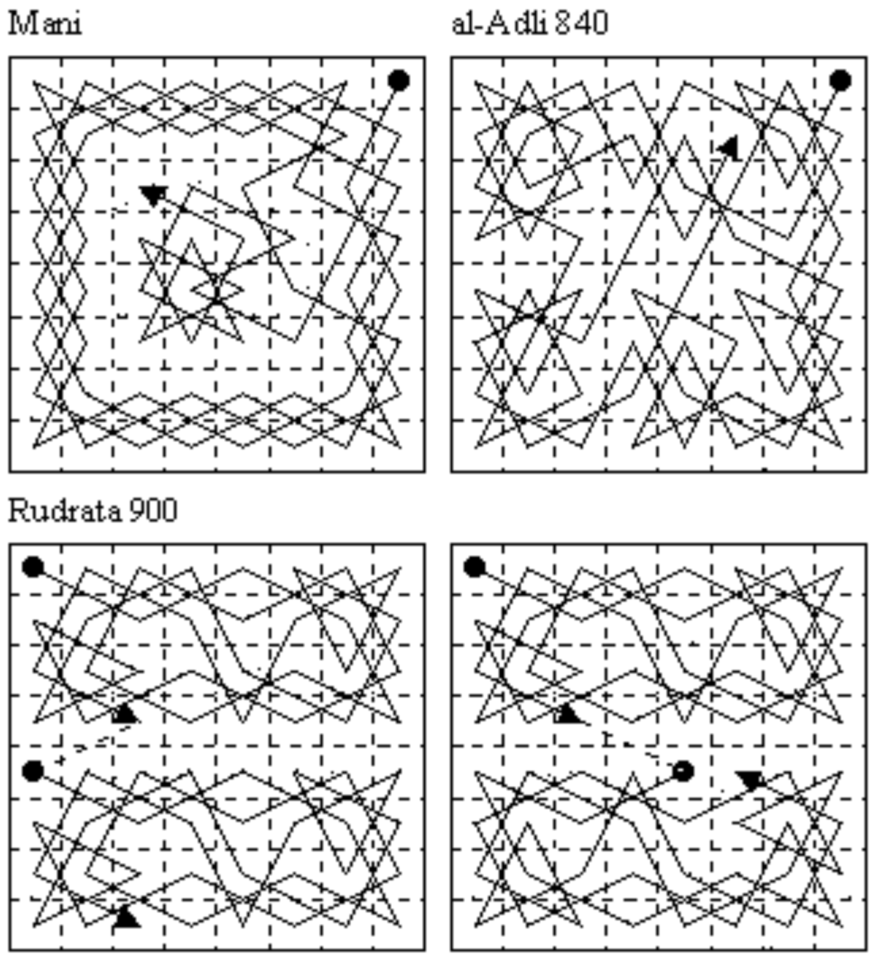




- **Note**: In chess, a knight may move only by jumping two spaces in one direction, followed by a jump one space in a perpendicular direction.

http://www.chess-poster.com/english/laws_of_chess.htm

9th Century: Knight Tours Discovered



18th Century: *N* x *N* Knight Tour Problem

- **1759**: Berlin Academy of Sciences proposes a 4000 francs prize for the solution of the more general problem of finding a knight tour on an *N* x *N* chessboard.

- **1766**: The problem is solved by Leonhard Euler (pronounced "*Oiler*").
  - The prize was never awarded since Euler was Director of Mathematics at Berlin Academy and was deemed ineligible.

Leonhard Euler

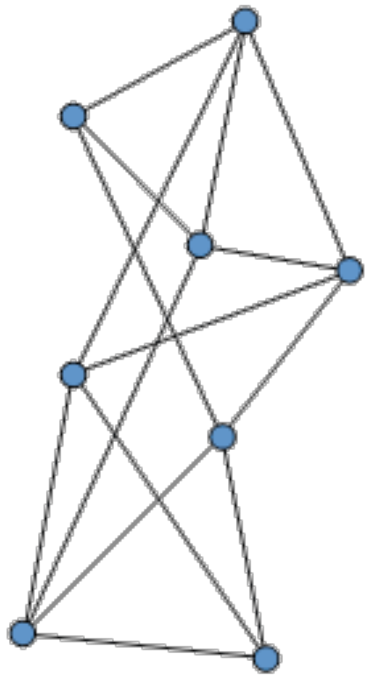
http://commons.wikimedia.org/wiki/File:Leonhard_Euler_by_Handmann.png

Introduction to Graph Theory

- A **graph** is a collection (V, E) of two sets:
  - V is simply a set of objects, which we call the **vertices** of G.
  - E is a set of pairs of vertices which we call the **edges** of G.

Introduction to Graph Theory

- A **graph** is a collection (V, E) of two sets:
  - V is simply a set of objects, which we call the **vertices** of G.
  - E is a set of pairs of vertices which we call the **edges** of G.
- Simpler: Think of G as a network:

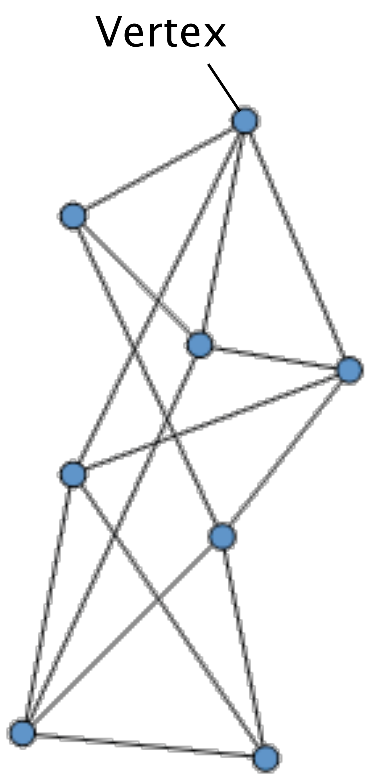


http://uh.edu/engines/epi2467.htm

Introduction to Graph Theory

- A **graph** is a collection (V, E) of two sets:
  - V is simply a set of objects, which we call the **vertices** of G.
  - E is a set of pairs of vertices which we call the **edges** of G.

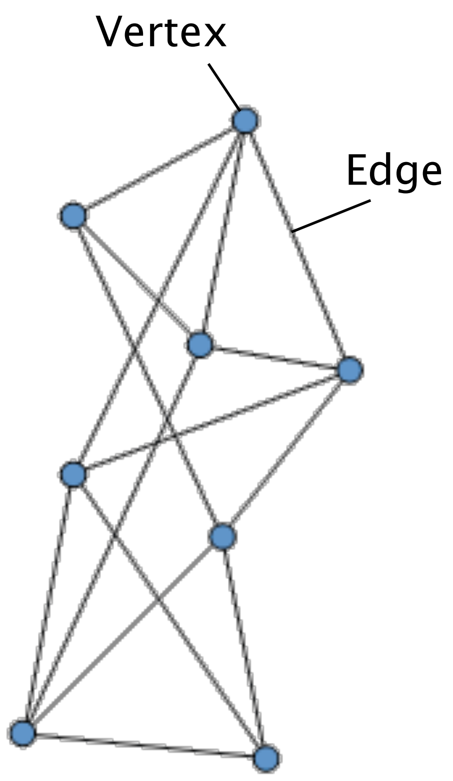
- Simpler: Think of G as a network:
  - Nodes = vertices



http://uh.edu/engines/epi2467.htm

Introduction to Graph Theory

- A **graph** is a collection (V, E) of two sets:
  - V is simply a set of objects, which we call the **vertices** of G.
  - E is a set of pairs of vertices which we call the **edges** of G.
- Simpler: Think of G as a network:
  - Nodes = vertices
  - Edges = segments connecting the nodes

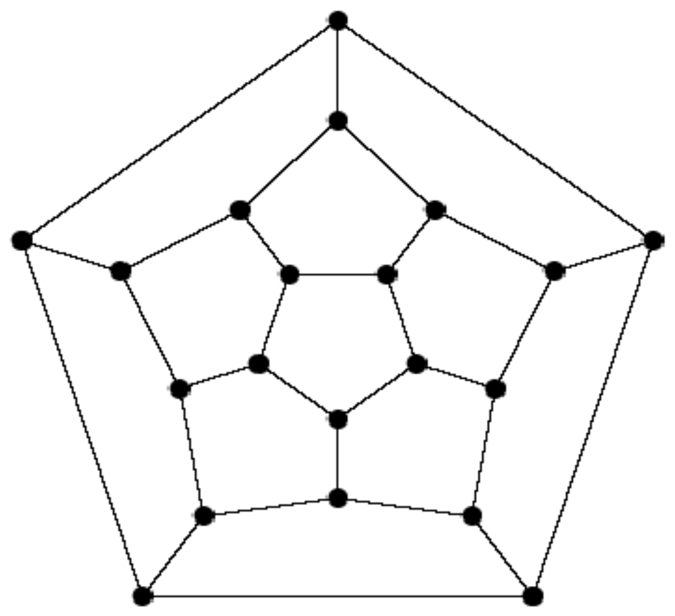


http://uh.edu/engines/epi2467.htm

Section 2: The Hamiltonian & Eulerian Cycle Problems

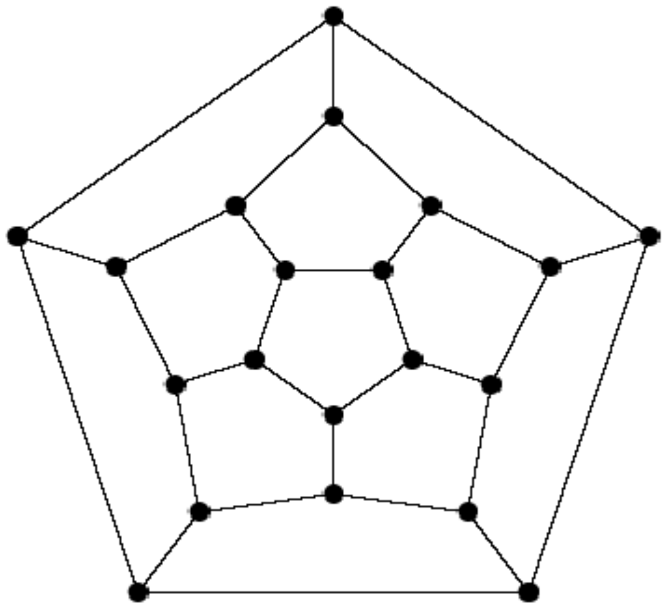
Hamiltonian Cycle Problem

- Input: A graph G = (V, E)
- Output: A **Hamiltonian cycle** in G, which is a cycle that visits every vertex exactly once.



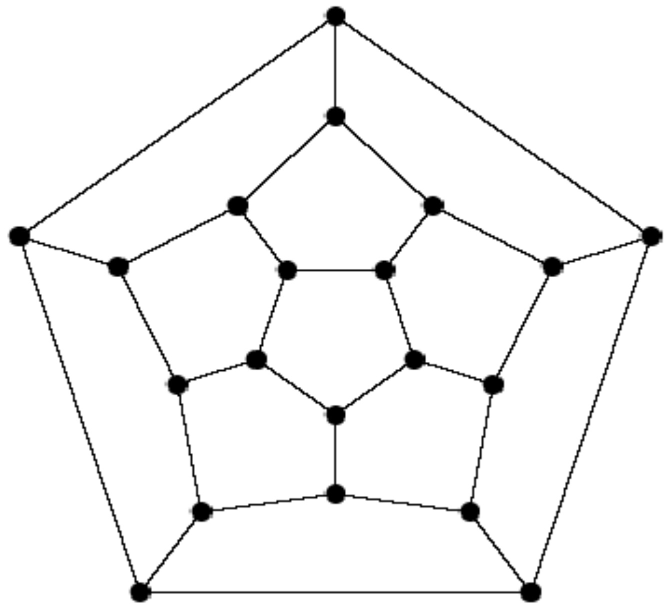
Hamiltonian Cycle Problem

- Input: A graph G = (V, E)
- Output: A **Hamiltonian cycle** in G, which is a cycle that visits every vertex exactly once.
- Example: In 1857, William Rowan Hamilton asked whether the graph to the right has such a cycle.



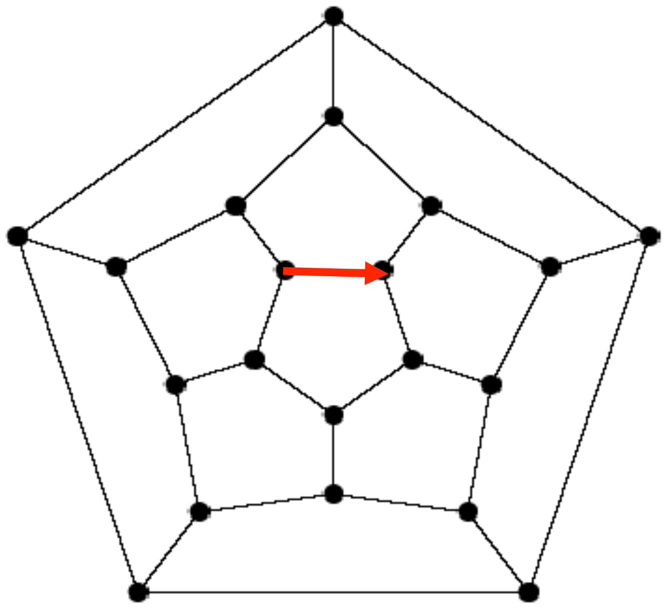
Hamiltonian Cycle Problem

- Input: A graph G = (V, E)
- Output: A **Hamiltonian cycle** in G, which is a cycle that visits every vertex exactly once.
- Example: In 1857, William Rowan Hamilton asked whether the graph to the right has such a cycle.
- Do you see a Hamiltonian cycle?



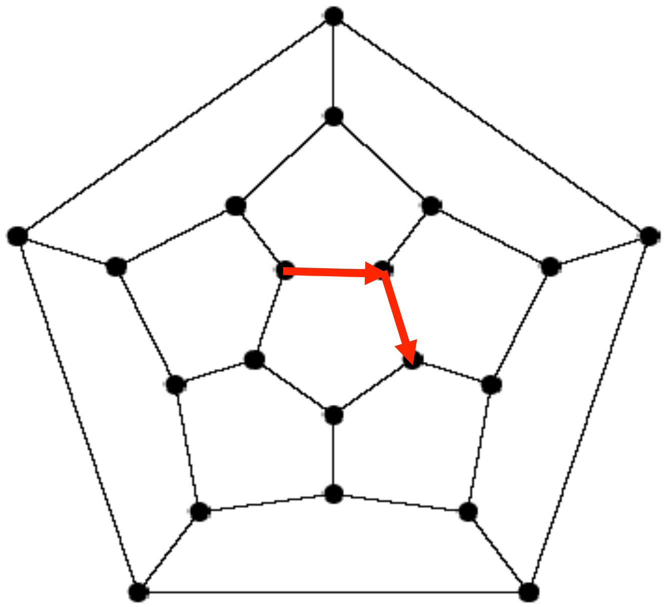
Hamiltonian Cycle Problem

- Input: A graph G = (V, E)
- Output: A **Hamiltonian cycle** in G, which is a cycle that visits every vertex exactly once.
- Example: In 1857, William Rowan Hamilton asked whether the graph to the right has such a cycle.
- Do you see a Hamiltonian cycle?



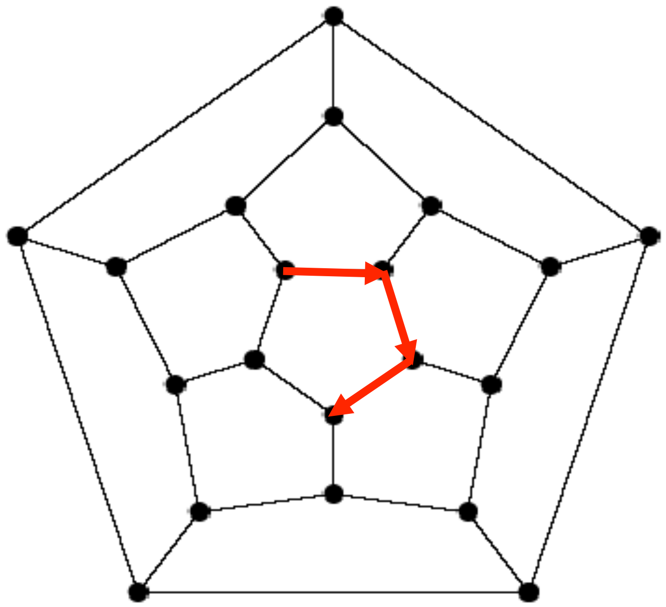
Hamiltonian Cycle Problem

- Input: A graph G = (V, E)
- Output: A **Hamiltonian cycle** in G, which is a cycle that visits every vertex exactly once.
- Example: In 1857, William Rowan Hamilton asked whether the graph to the right has such a cycle.
- Do you see a Hamiltonian cycle?



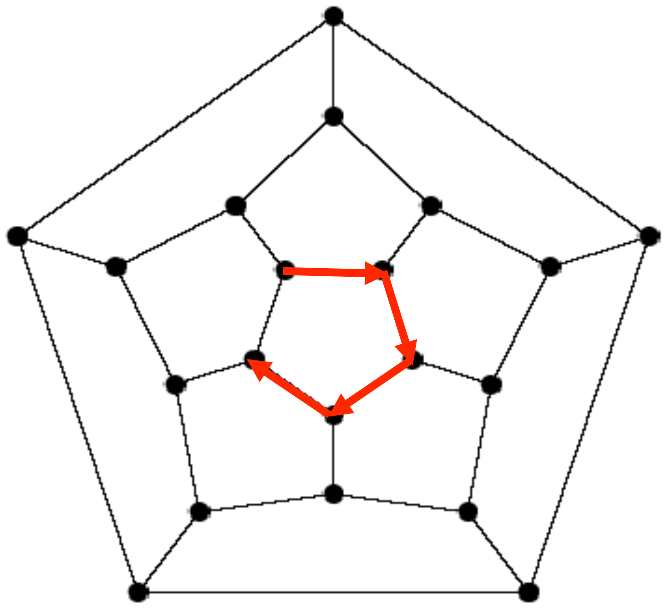
Hamiltonian Cycle Problem

- Input: A graph G = (V, E)
- Output: A **Hamiltonian cycle** in G, which is a cycle that visits every vertex exactly once.
- Example: In 1857, William Rowan Hamilton asked whether the graph to the right has such a cycle.
- Do you see a Hamiltonian cycle?



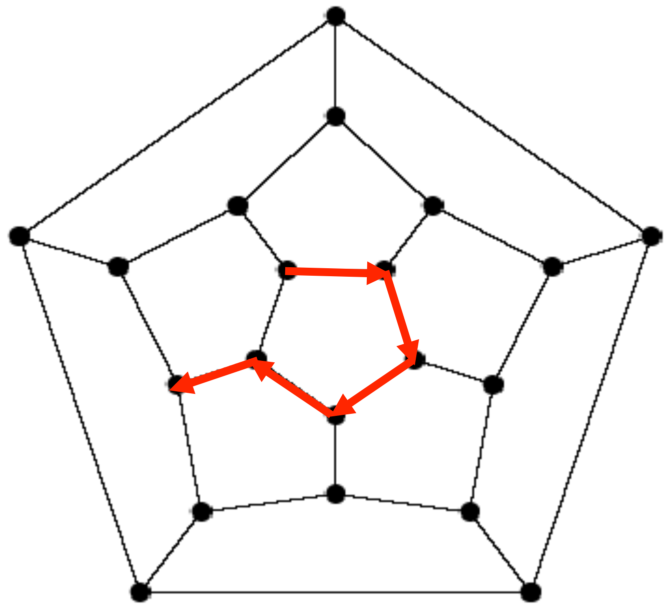
Hamiltonian Cycle Problem

- Input: A graph G = (V, E)
- Output: A **Hamiltonian cycle** in G, which is a cycle that visits every vertex exactly once.
- Example: In 1857, William Rowan Hamilton asked whether the graph to the right has such a cycle.
- Do you see a Hamiltonian cycle?



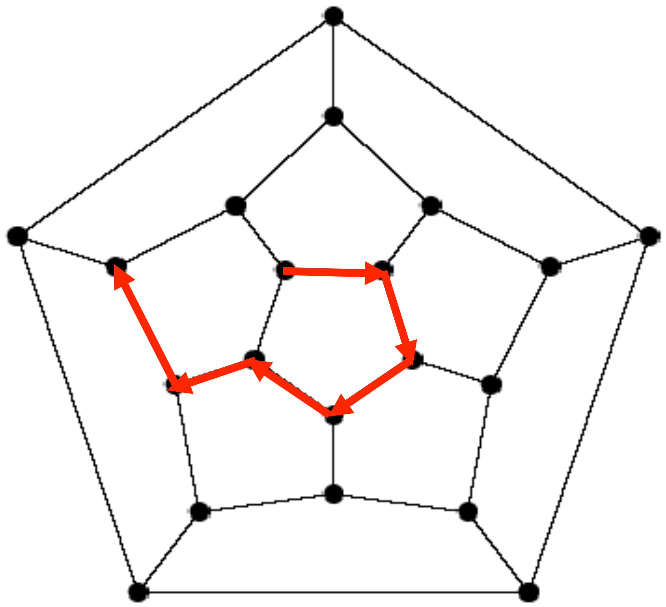
Hamiltonian Cycle Problem

- Input: A graph G = (V, E)
- Output: A **Hamiltonian cycle** in G, which is a cycle that visits every vertex exactly once.
- Example: In 1857, William Rowan Hamilton asked whether the graph to the right has such a cycle.
- Do you see a Hamiltonian cycle?



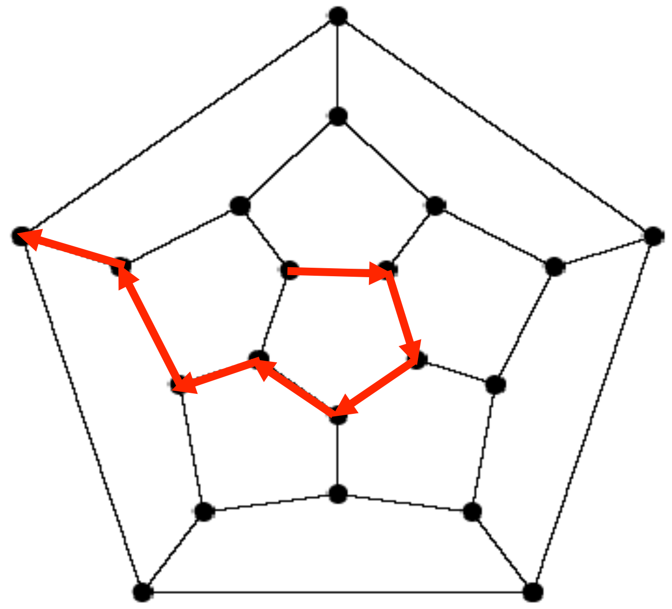
Hamiltonian Cycle Problem

- Input: A graph G = (V, E)
- Output: A **Hamiltonian cycle** in G, which is a cycle that visits every vertex exactly once.
- Example: In 1857, William Rowan Hamilton asked whether the graph to the right has such a cycle.
- Do you see a Hamiltonian cycle?



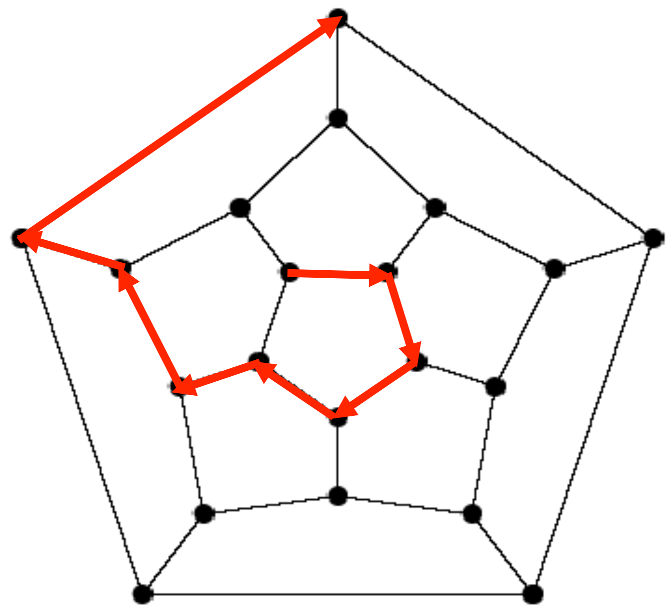
Hamiltonian Cycle Problem

- Input: A graph G = (V, E)
- Output: A **Hamiltonian cycle** in G, which is a cycle that visits every vertex exactly once.
- Example: In 1857, William Rowan Hamilton asked whether the graph to the right has such a cycle.
- Do you see a Hamiltonian cycle?



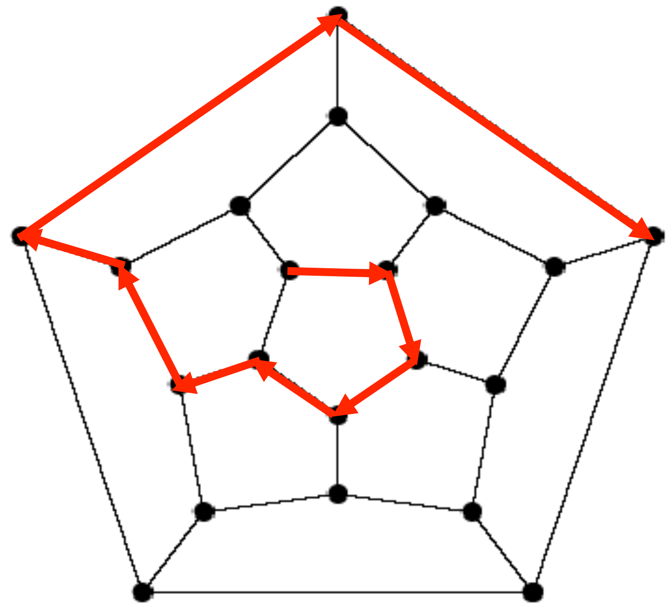
Hamiltonian Cycle Problem

- Input: A graph G = (V, E)
- Output: A **Hamiltonian cycle** in G, which is a cycle that visits every vertex exactly once.
- Example: In 1857, William Rowan Hamilton asked whether the graph to the right has such a cycle.
- Do you see a Hamiltonian cycle?



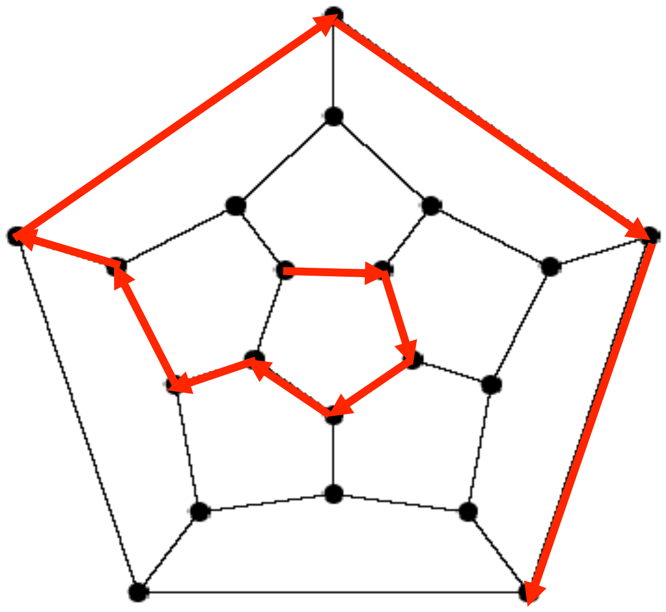
Hamiltonian Cycle Problem

- Input: A graph G = (V, E)
- Output: A **Hamiltonian cycle** in G, which is a cycle that visits every vertex exactly once.
- Example: In 1857, William Rowan Hamilton asked whether the graph to the right has such a cycle.
- Do you see a Hamiltonian cycle?



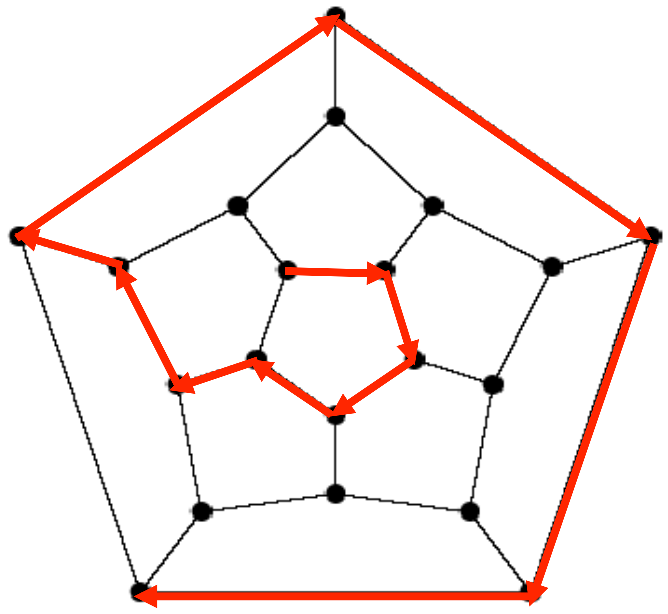
Hamiltonian Cycle Problem

- Input: A graph G = (V, E)
- Output: A **Hamiltonian cycle** in G, which is a cycle that visits every vertex exactly once.
- Example: In 1857, William Rowan Hamilton asked whether the graph to the right has such a cycle.
- Do you see a Hamiltonian cycle?



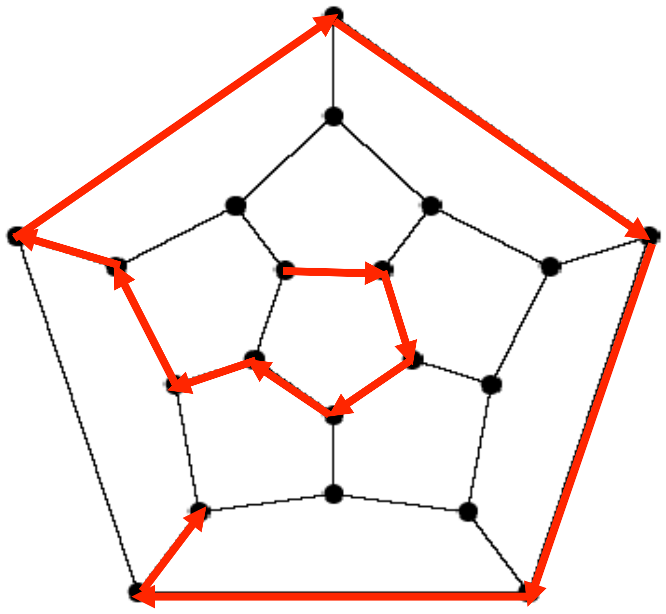
Hamiltonian Cycle Problem

- Input: A graph G = (V, E)
- Output: A **Hamiltonian cycle** in G, which is a cycle that visits every vertex exactly once.
- Example: In 1857, William Rowan Hamilton asked whether the graph to the right has such a cycle.
- Do you see a Hamiltonian cycle?



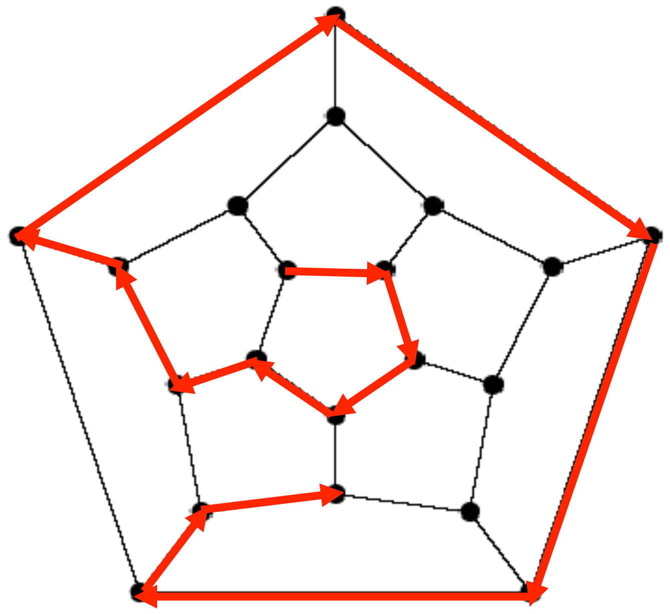
Hamiltonian Cycle Problem

- Input: A graph G = (V, E)
- Output: A **Hamiltonian cycle** in G, which is a cycle that visits every vertex exactly once.
- Example: In 1857, William Rowan Hamilton asked whether the graph to the right has such a cycle.
- Do you see a Hamiltonian cycle?



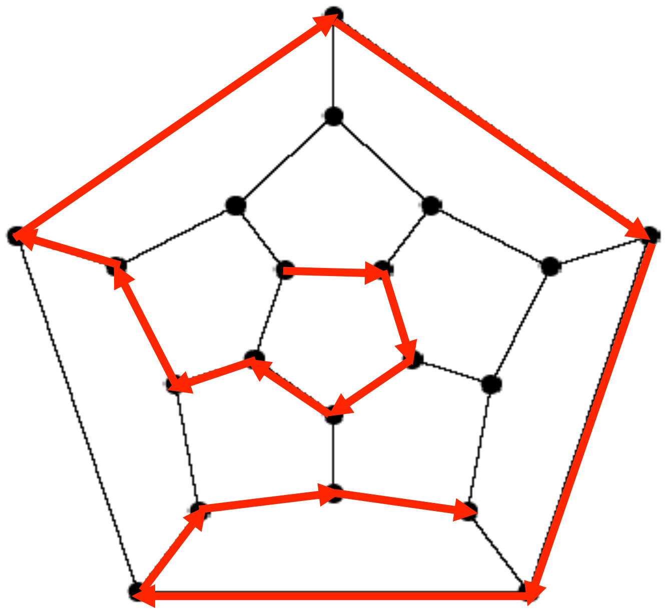
Hamiltonian Cycle Problem

- Input: A graph G = (V, E)
- Output: A **Hamiltonian cycle** in G, which is a cycle that visits every vertex exactly once.
- Example: In 1857, William Rowan Hamilton asked whether the graph to the right has such a cycle.
- Do you see a Hamiltonian cycle?



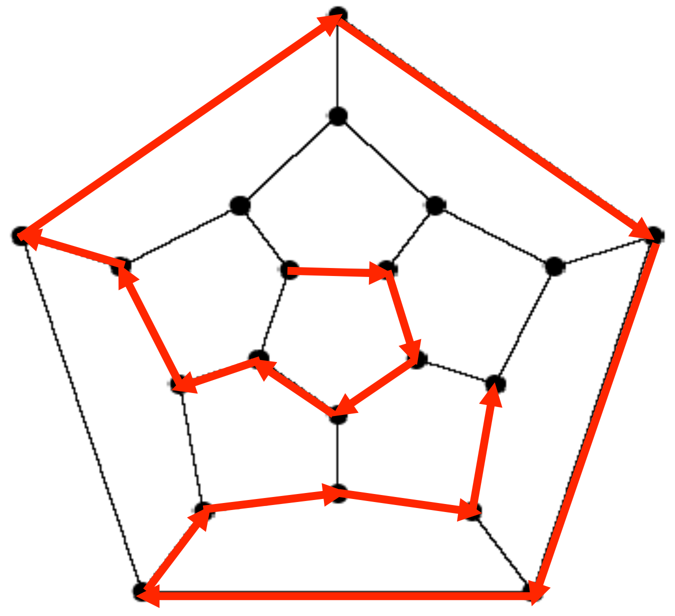
Hamiltonian Cycle Problem

- Input: A graph G = (V, E)
- Output: A **Hamiltonian cycle** in G, which is a cycle that visits every vertex exactly once.
- Example: In 1857, William Rowan Hamilton asked whether the graph to the right has such a cycle.
- Do you see a Hamiltonian cycle?



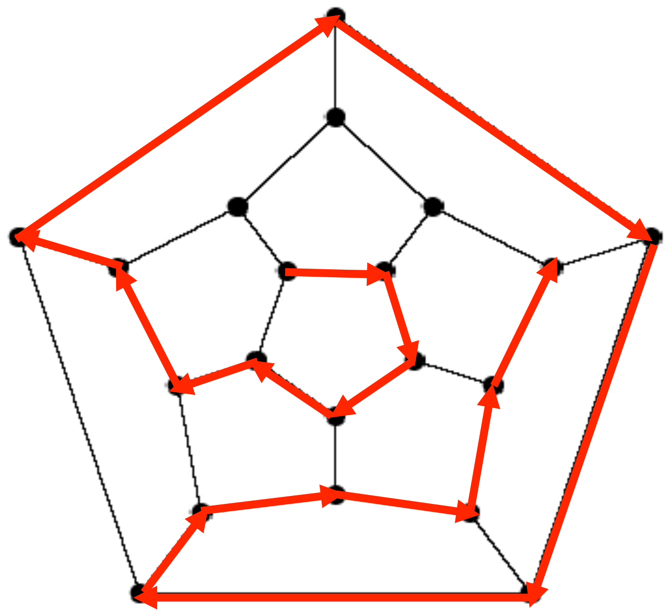
Hamiltonian Cycle Problem

- Input: A graph G = (V, E)
- Output: A **Hamiltonian cycle** in G, which is a cycle that visits every vertex exactly once.
- Example: In 1857, William Rowan Hamilton asked whether the graph to the right has such a cycle.
- Do you see a Hamiltonian cycle?



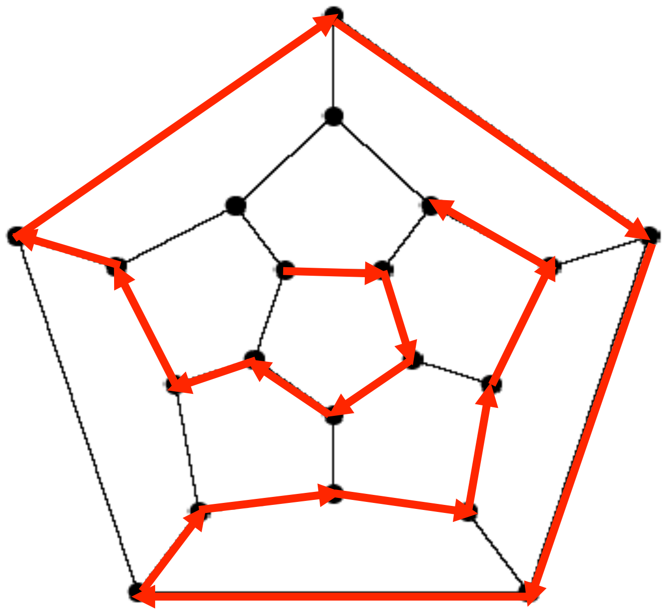
Hamiltonian Cycle Problem

- Input: A graph G = (V, E)
- Output: A **Hamiltonian cycle** in G, which is a cycle that visits every vertex exactly once.
- Example: In 1857, William Rowan Hamilton asked whether the graph to the right has such a cycle.
- Do you see a Hamiltonian cycle?



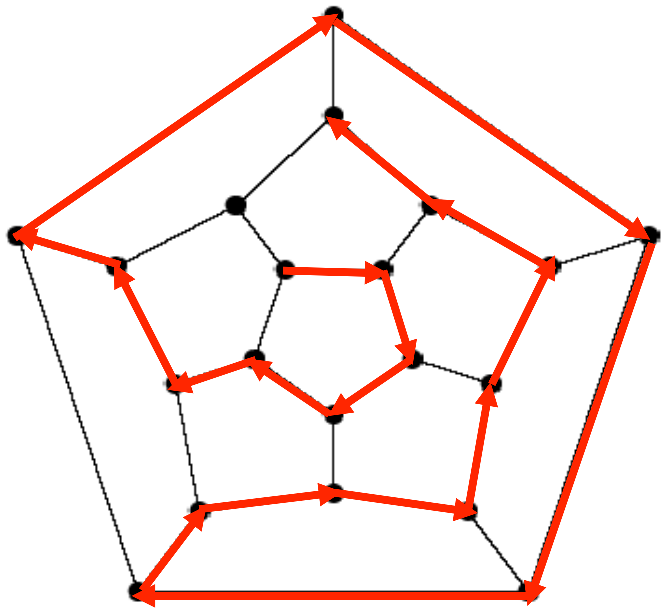
Hamiltonian Cycle Problem

- Input: A graph G = (V, E)
- Output: A **Hamiltonian cycle** in G, which is a cycle that visits every vertex exactly once.
- Example: In 1857, William Rowan Hamilton asked whether the graph to the right has such a cycle.
- Do you see a Hamiltonian cycle?



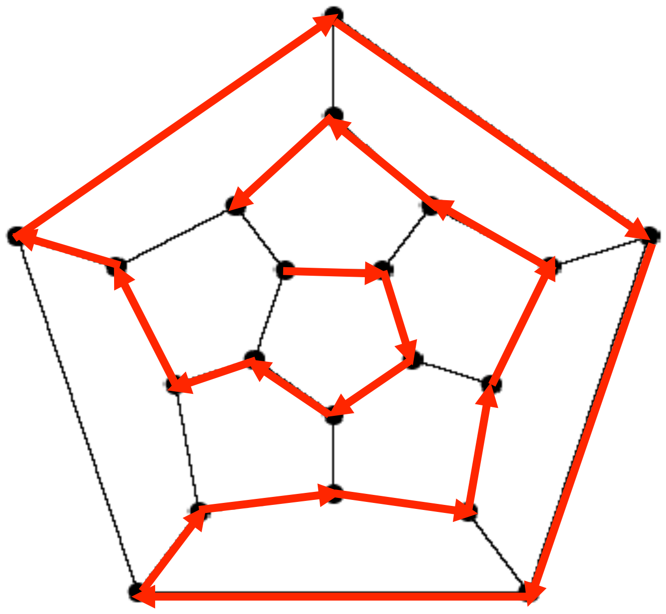
Hamiltonian Cycle Problem

- Input: A graph G = (V, E)
- Output: A **Hamiltonian cycle** in G, which is a cycle that visits every vertex exactly once.
- Example: In 1857, William Rowan Hamilton asked whether the graph to the right has such a cycle.
- Do you see a Hamiltonian cycle?



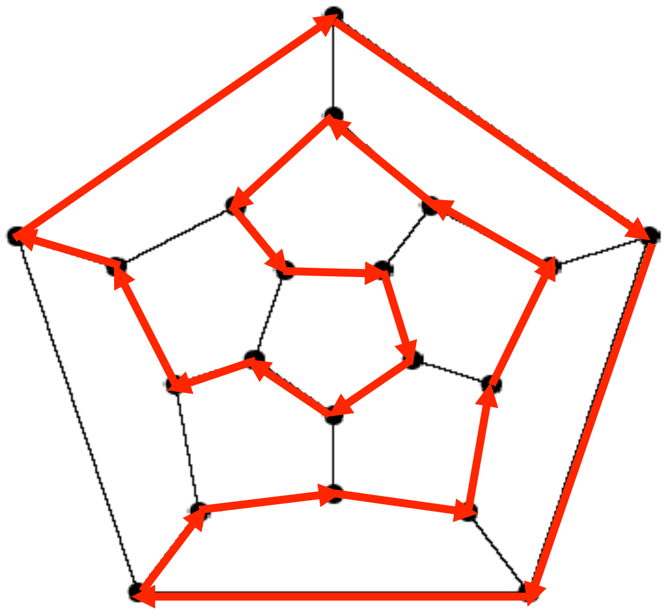
Hamiltonian Cycle Problem

- Input: A graph G = (V, E)
- Output: A **Hamiltonian cycle** in G, which is a cycle that visits every vertex exactly once.
- Example: In 1857, William Rowan Hamilton asked whether the graph to the right has such a cycle.
- Do you see a Hamiltonian cycle?



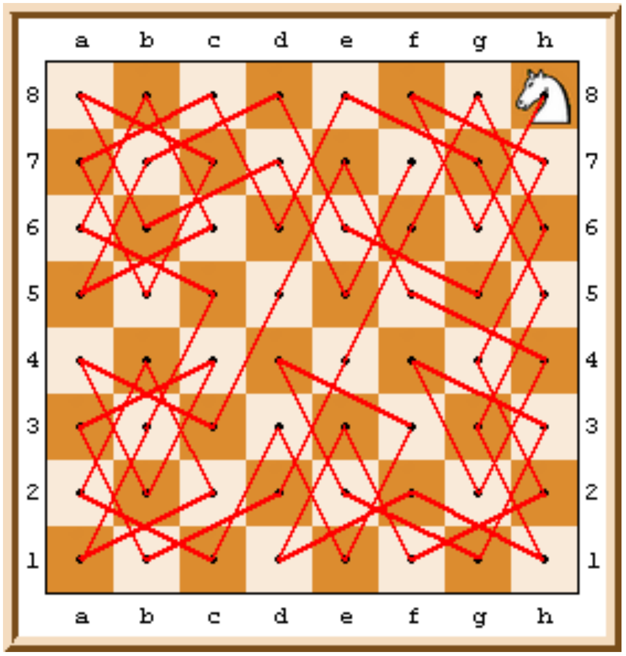
Hamiltonian Cycle Problem

- Input: A graph G = (V, E)
- Output: A **Hamiltonian cycle** in G, which is a cycle that visits every vertex exactly once.
- Example: In 1857, William Rowan Hamilton asked whether the graph to the right has such a cycle.
- Do you see a Hamiltonian cycle?



Knight Tours Revisited

- Let us form a graph G = (V, E) as follows:
  - V = the squares of a chessboard
  - E = the set of edges (*v*, *w*) where *v* and *w* are squares on the chessboard and a knight can jump from *v* to *w* in a single move.



- Hence, a knight tour is just a Hamiltonian Cycle in this graph!

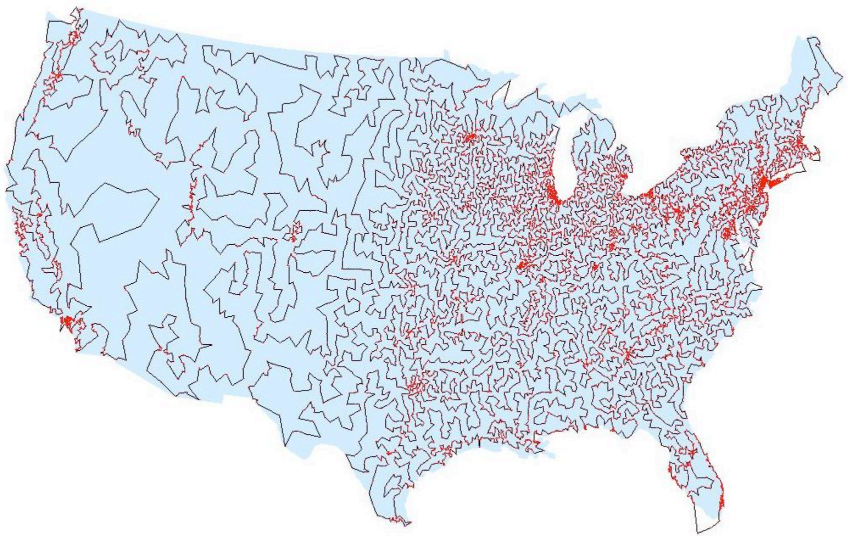
Hamiltonian Cycle Problem

- **Theorem**: The Hamiltonian Cycle Problem is NP-Complete.

- This result explains why knight tours were so difficult to find; there is no known quick method to find them!

Hamiltonian Cycle Problem as TSP

- Recall the Traveling Salesman Problem (TSP):
  - *n* cities
  - Cost of traveling from *i* to *j* is given by c(*i*, *j*)
  - Goal: Find the tour of all the cities of lowest total cost.
  - Example at right: One busy salesman!



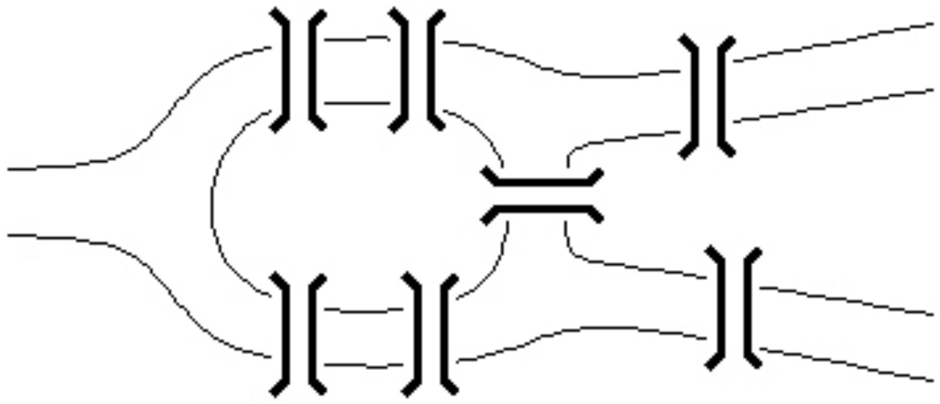
- So we might like to think of the Hamiltonian Cycle Problem as a TSP with all costs = 1, where we have some edges missing (there doesn't always exist a flight between all pairs of cities).

http://www.ima.umn.edu/public-lecture/tsp/index.html

The Bridges of Konigsberg

- The city of Konigsberg, Prussia (today: Kaliningrad, Russia) was made up of both banks of a river, as well as two islands.

- The riverbanks and the islands were connected with bridges, as follows:



- The residents wanted to know if they could take a walk from anywhere in the city, cross each bridge exactly once, and wind up where they started.

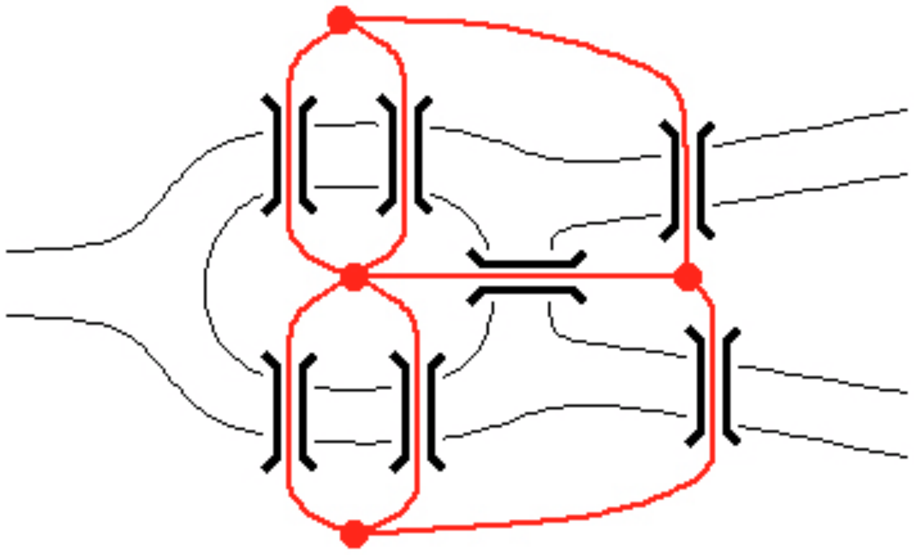
http://www.math.uwaterloo.ca/navigation/ideas/Zeno/zenocando.shtml

The Bridges of Konigsberg

- **1735**: Enter Euler...his idea: compress each land area down to a single point, and each bridge down to a segment connecting two points.

The Bridges of Konigsberg

- **1735**: Enter Euler...his idea: compress each land area down to a single point, and each bridge down to a segment connecting two points.
    - This is just a graph!

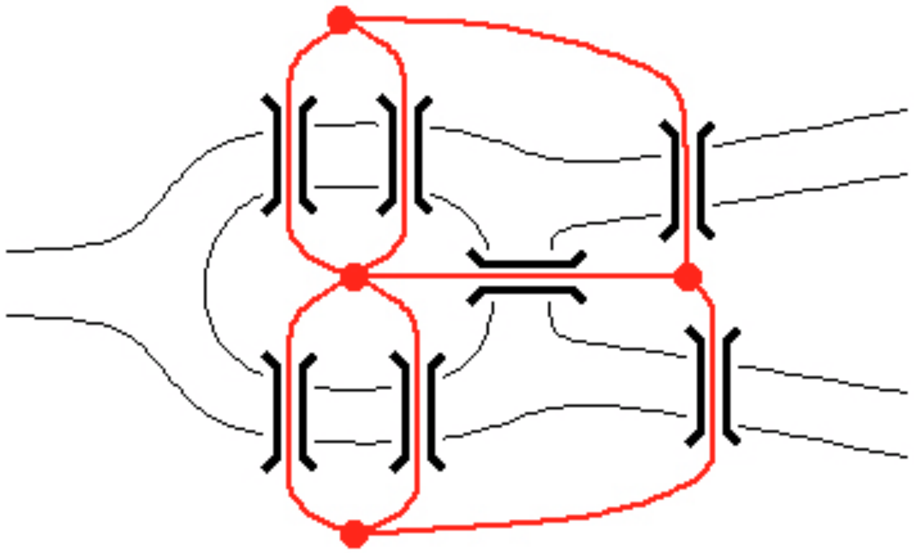


http://www.math.uwaterloo.ca/navigation/ideas/Zeno/zenocando.shtml

The Bridges of Konigsberg

- **1735**: Enter Euler...his idea: compress each land area down to a single point, and each bridge down to a segment connecting two points.
    - This is just a graph!

- What we are looking for, then, is a cycle in this graph which covers each edge exactly once.

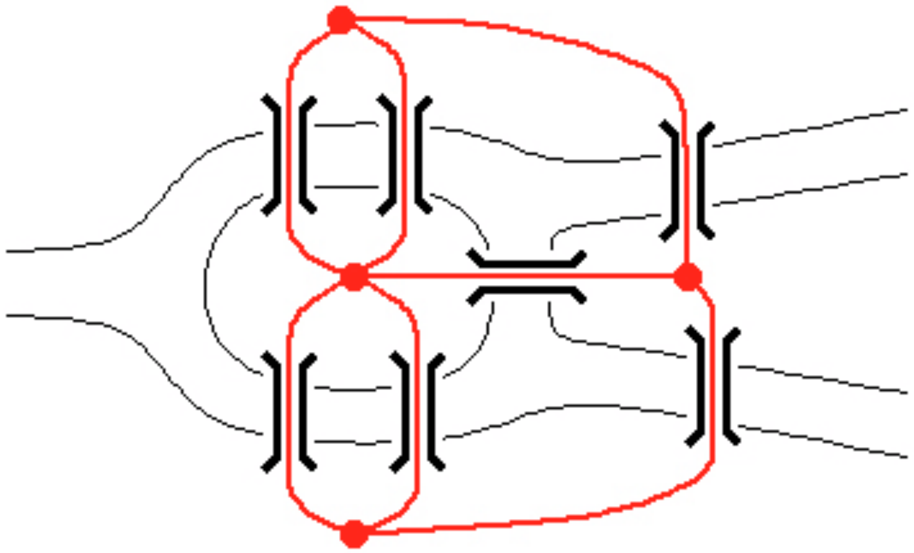


http://www.math.uwaterloo.ca/navigation/ideas/Zeno/zenocando.shtml

The Bridges of Konigsberg

- **1735**: Enter Euler...his idea: compress each land area down to a single point, and each bridge down to a segment connecting two points.
  - This is just a graph!

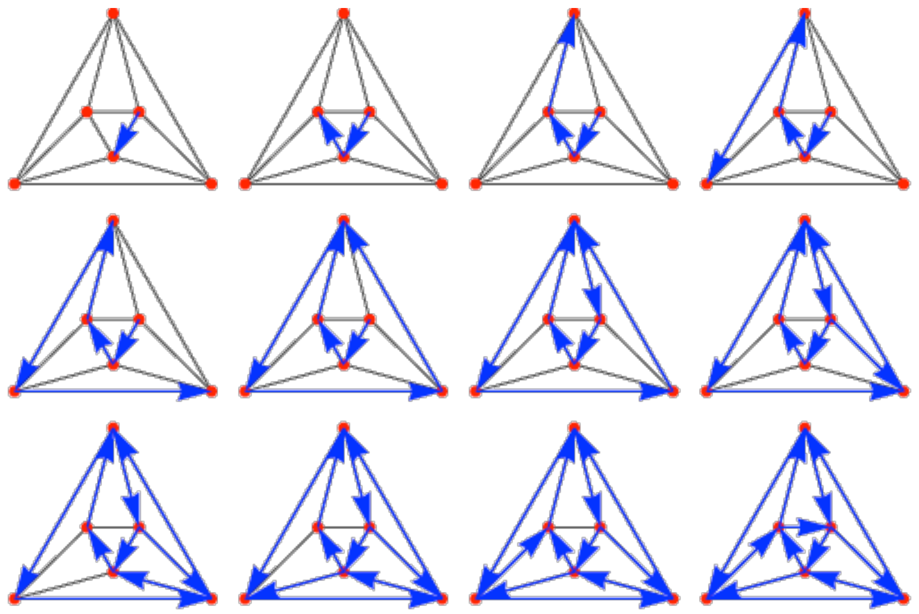
- What we are looking for, then, is a cycle in this graph which covers each edge exactly once.



- Using this setup, Euler showed that such a cycle cannot exist.

Eulerian Cycle Problem

- Input: A graph G = (V, E).
- Output: A cycle in G that touches every edge in E (called an **Eulerian cycle**), if one exists.
- Example: At right is a demonstration of an Eulerian cycle.



http://mathworld.wolfram.com/EulerianCycle.html

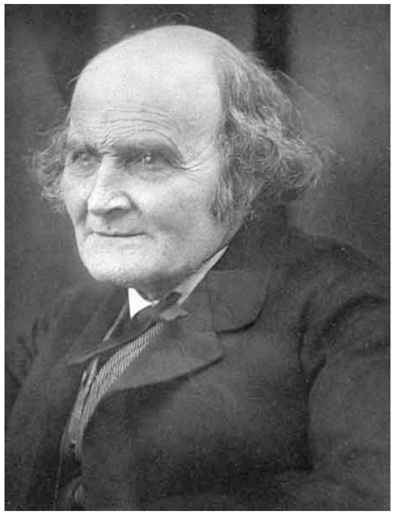
Eulerian Cycle Problem

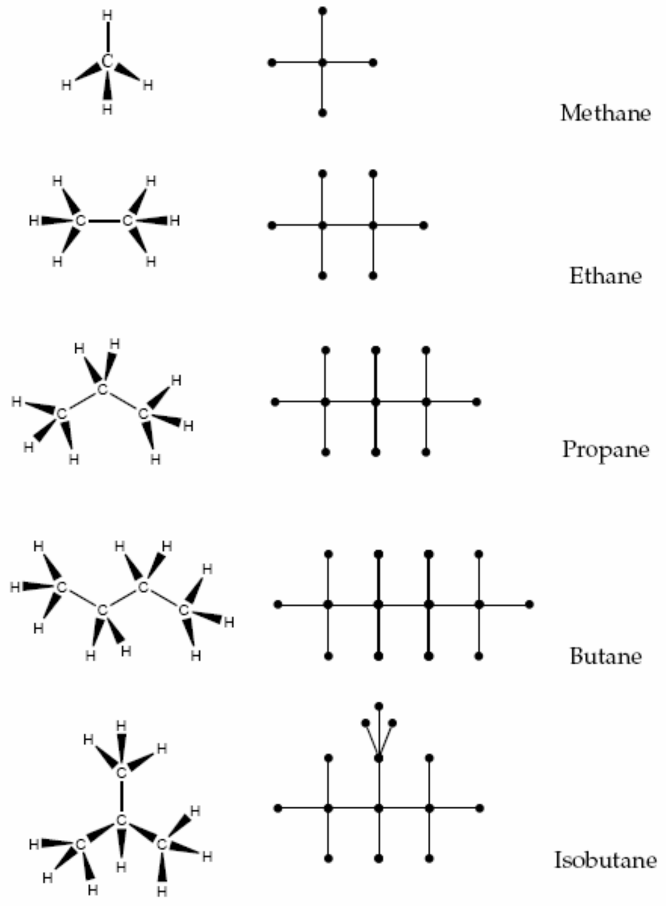
- **Theorem**: The Eulerian Cycle Problem can be solved in linear time.
- So whereas finding a Hamiltonian cycle quickly becomes intractable for an arbitrary graph, finding an Eulerian cycle is relatively much easier.
- Keep this fact in mind, as it will become essential.

Section 3:
Basic Biological Applications of Graph Theory

Modeling Hydrocarbons with Graphs

- Arthur Cayley studied chemical structures of hydrocarbons in the mid-1800s.
- He used **trees** (acyclic connected graphs) to enumerate structural isomers.


Arthur Cayley

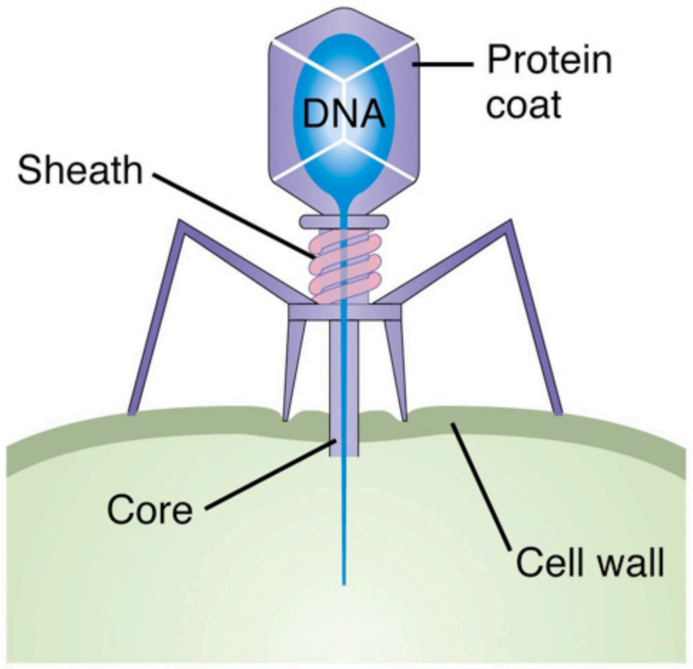


Hydrocarbon Structure

http://www.scientific-web.com/en/Mathematics/Biographies/ArthurCayley01.html

T4 Bacteriophages: Life Finds a Way

- Normally, the T4 bacteriophage kills bacteria
- However, if T4 is mutated (e.g., an important gene is deleted) it gets disabled and loses the ability to kill bacteria
- Suppose a bacterium is infected with two different disabled mutants– would the bacterium still survive?
- Amazingly, a pair of disabled viruses can still kill a bacterium.
- How is this possible?



T4 Bacteriophage

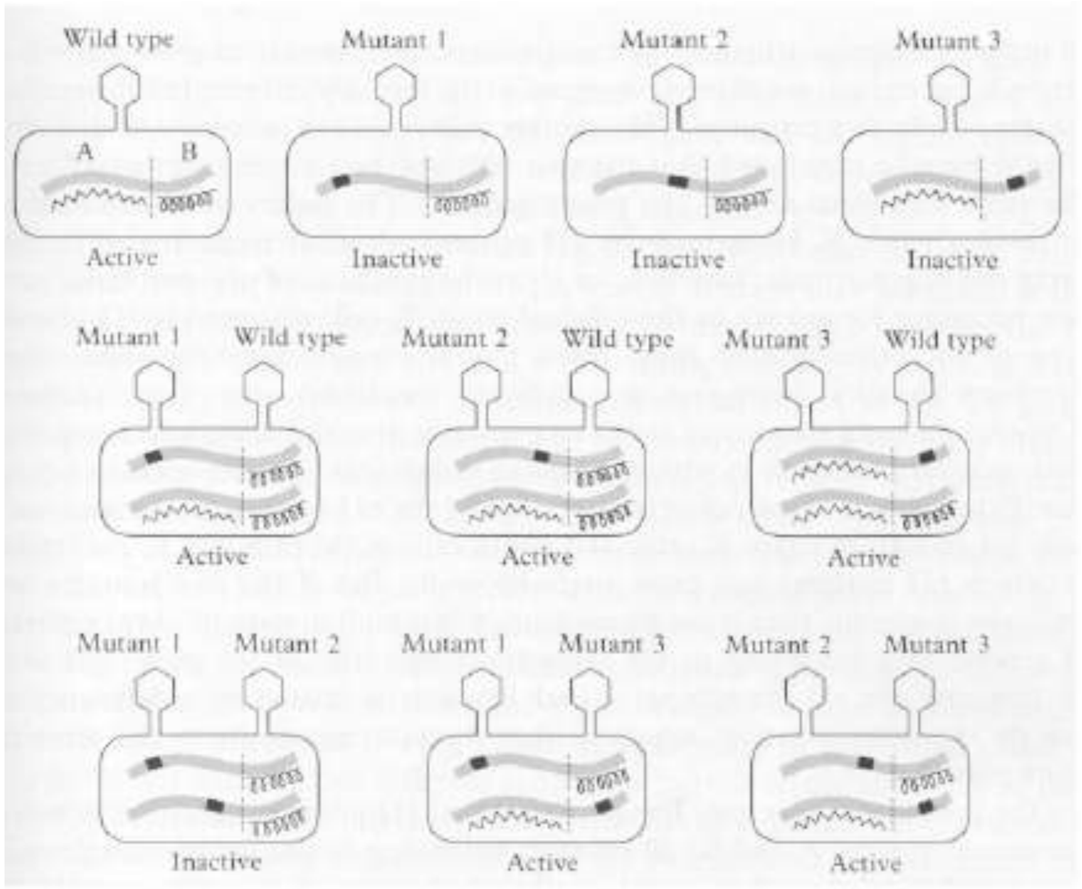
Benzer's Experiment

- **Seymour Benzer's Idea**: Infect bacteria with pairs of mutant T4 bacteriophage (virus).
- Each T4 mutant has an unknown interval deleted from its genome.
- If the two intervals overlap: T4 pair is missing part of its genome and is disabled—bacteria survive.
- If the two intervals do not overlap: T4 pair has its entire genome and is enabled – bacteria are killed.

Seymour Benzer

http://commons.wikimedia.org/wiki/File:Seymour_Benzer.gif

Benzer's Experiment: Illustration

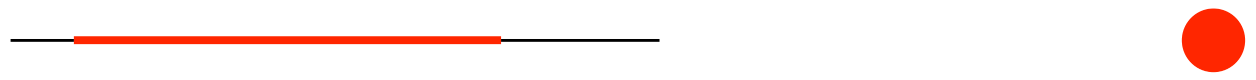


Benzer's Experiment and Graph Theory

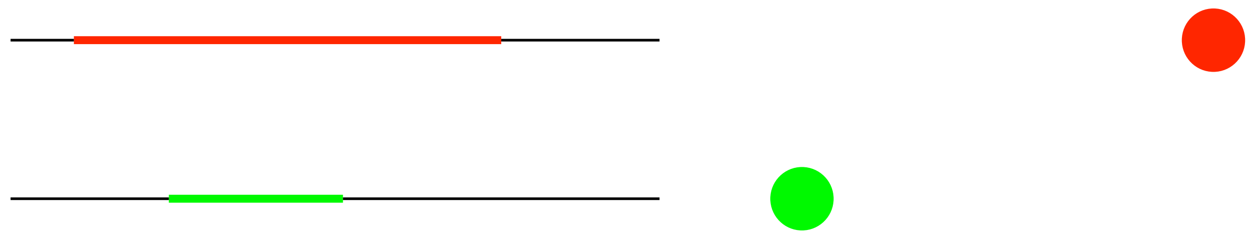
- We construct an **interval graph**:
  - Each T4 mutant forms a vertex.
  - Place an edge between mutant pairs where bacteria survived (i.e., the deleted intervals in the pair of mutants overlap)

- As the next slides show, the interval graph structure reveals whether DNA is *linear* or *branched*.

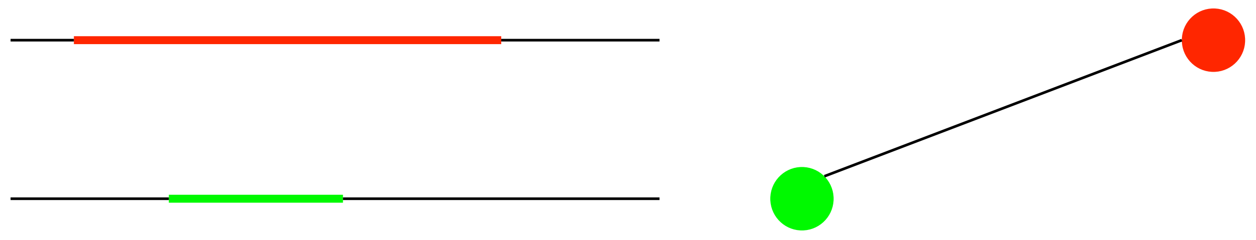
Interval Graph: Linear Genomes



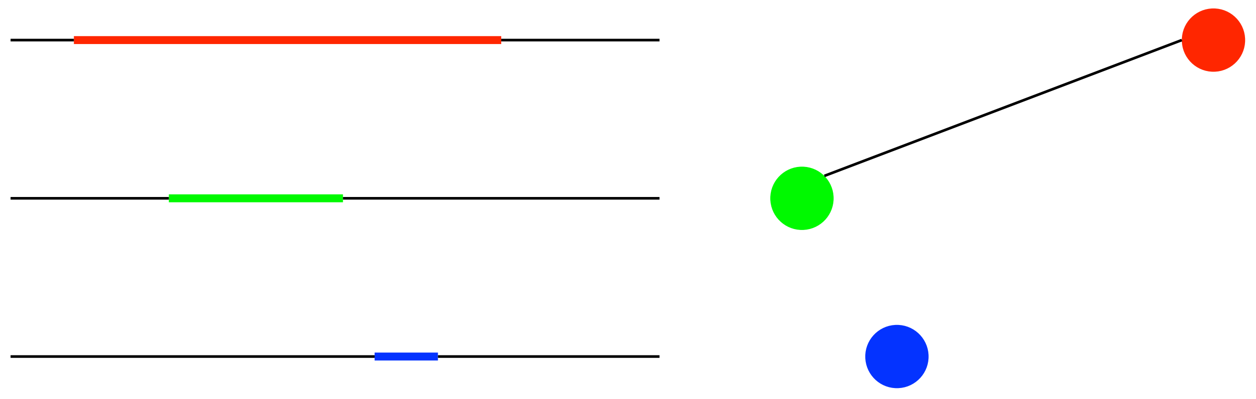
Interval Graph: Linear Genomes



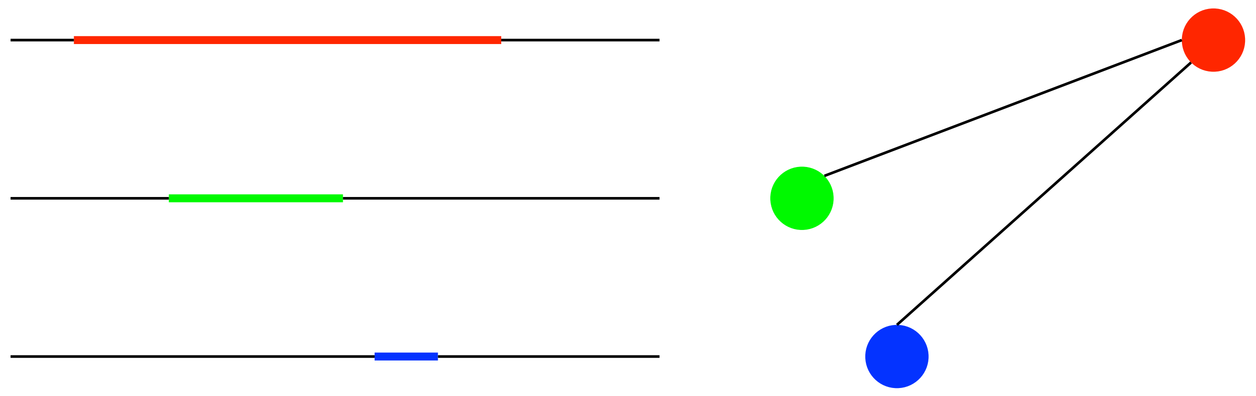
Interval Graph: Linear Genomes



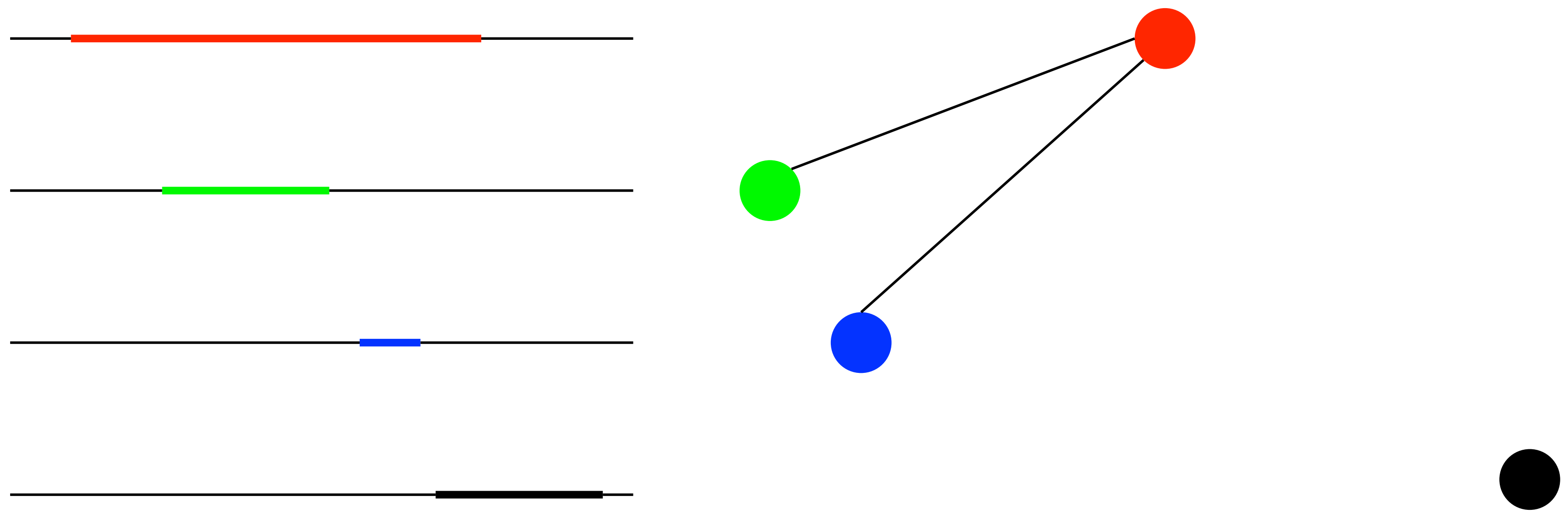
Interval Graph: Linear Genomes



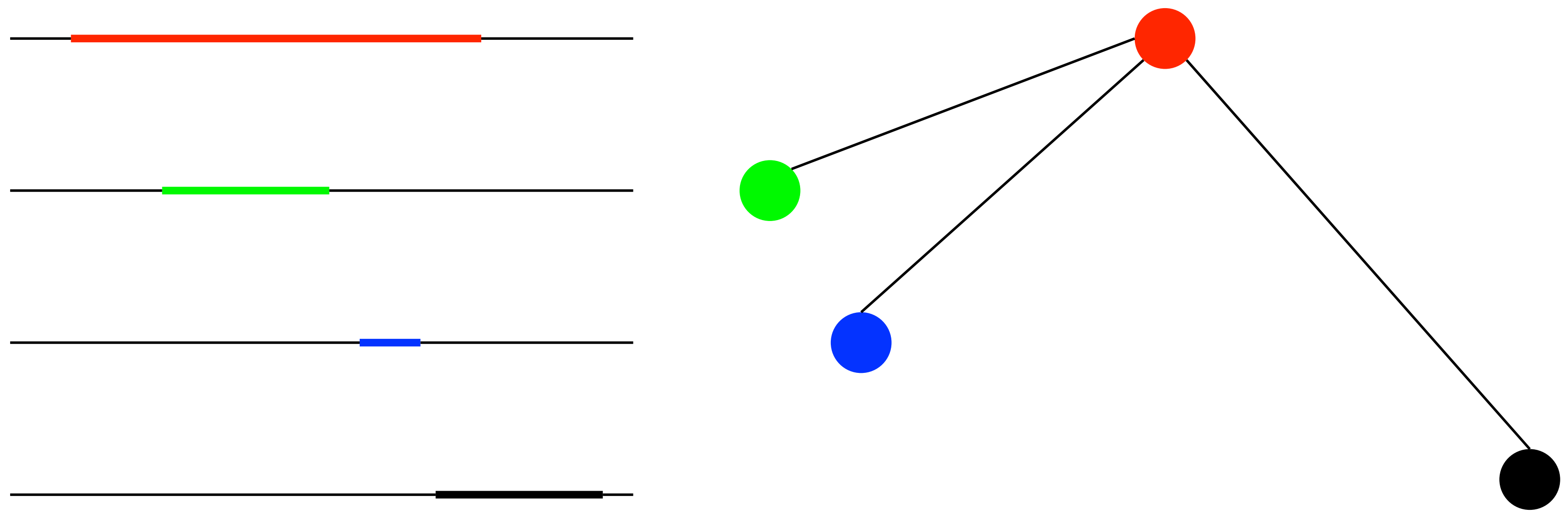
Interval Graph: Linear Genomes



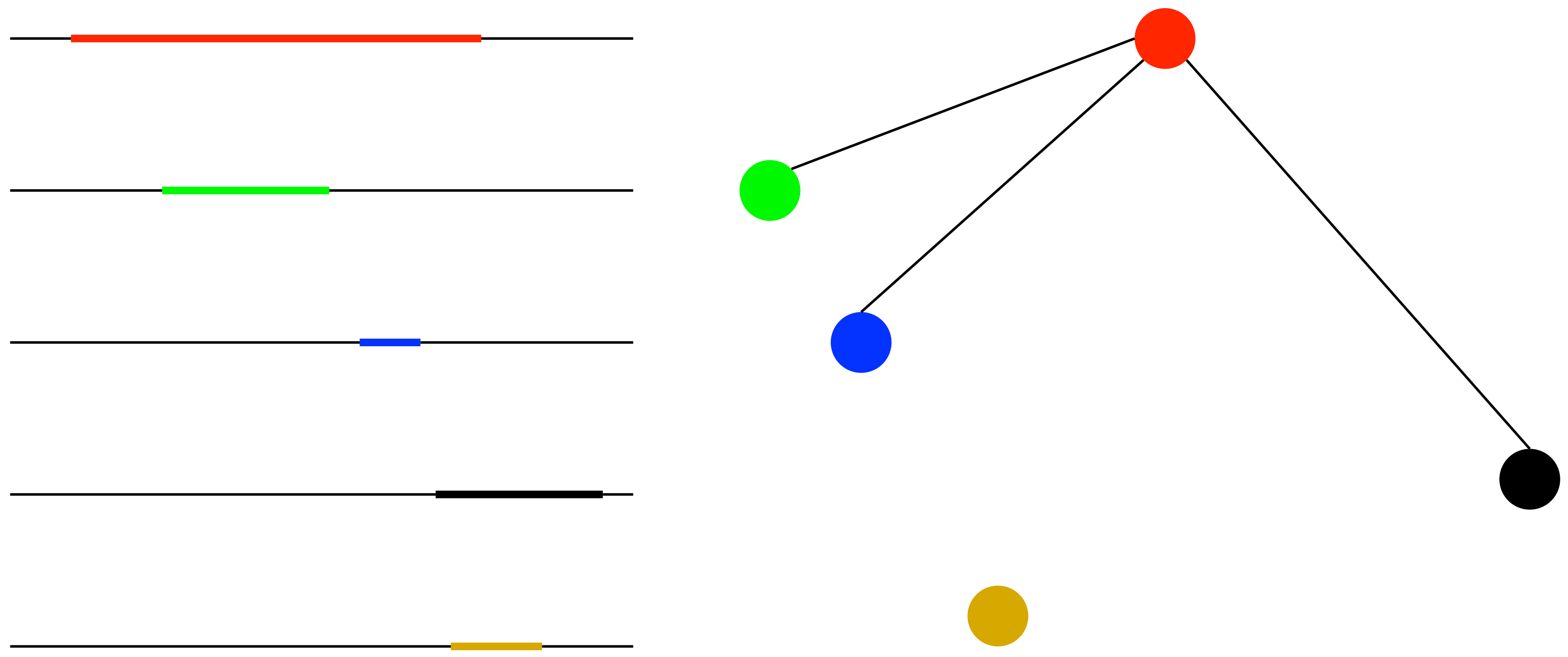
Interval Graph: Linear Genomes



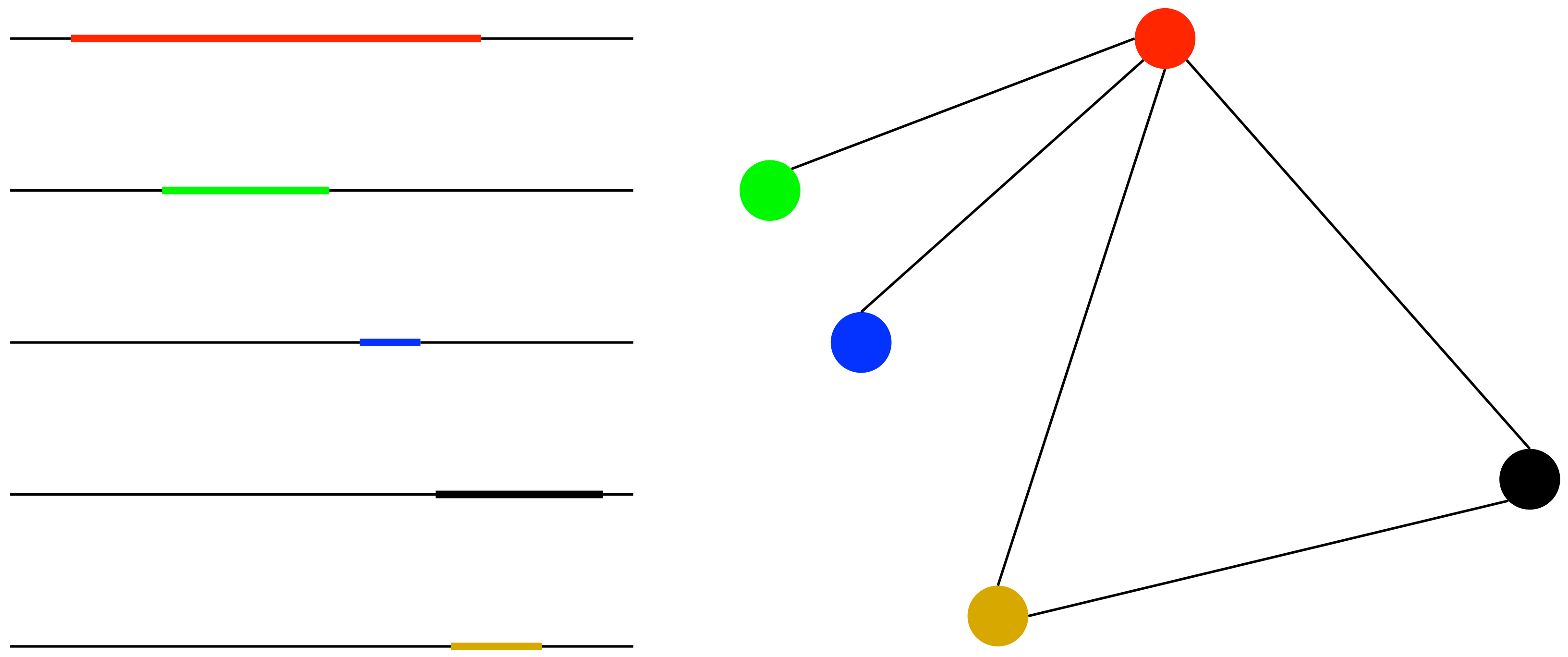
Interval Graph: Linear Genomes



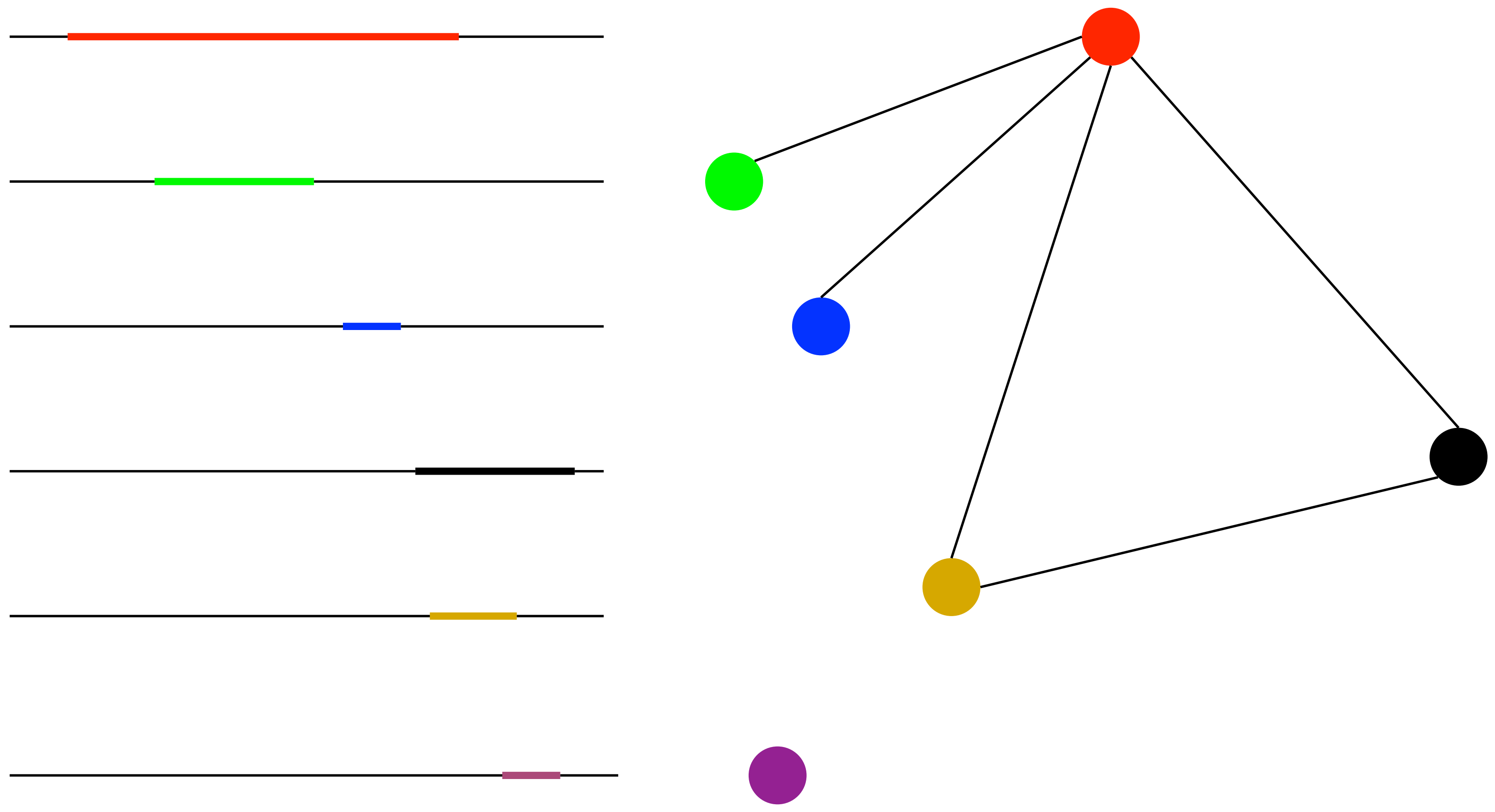
Interval Graph: Linear Genomes



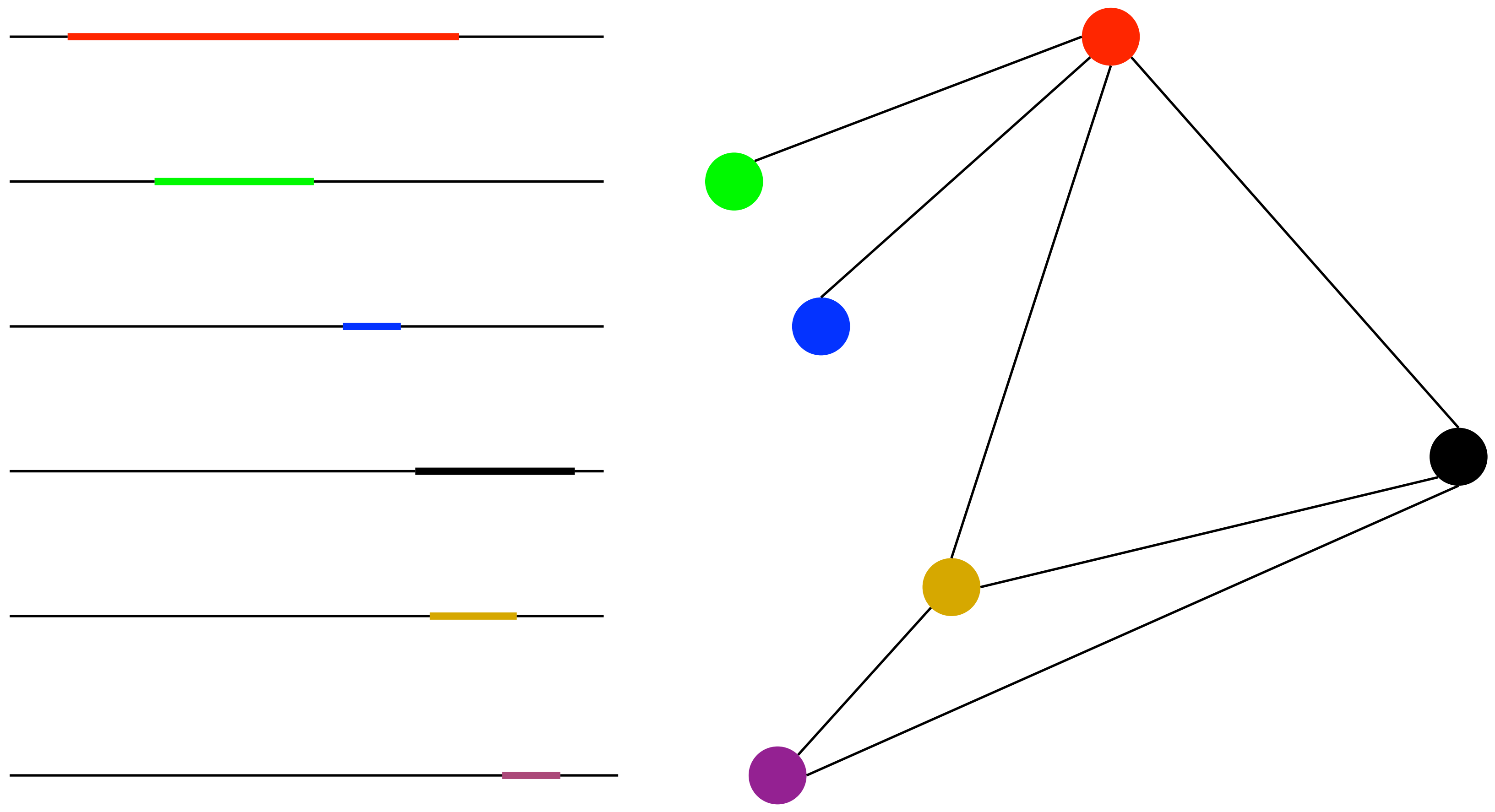
Interval Graph: Linear Genomes



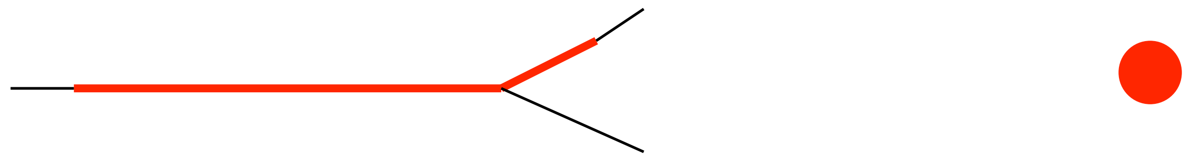
Interval Graph: Linear Genomes



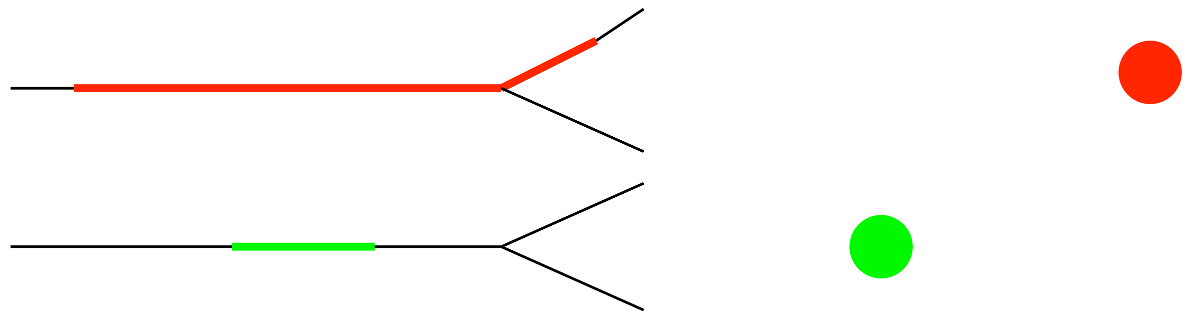
Interval Graph: Linear Genomes



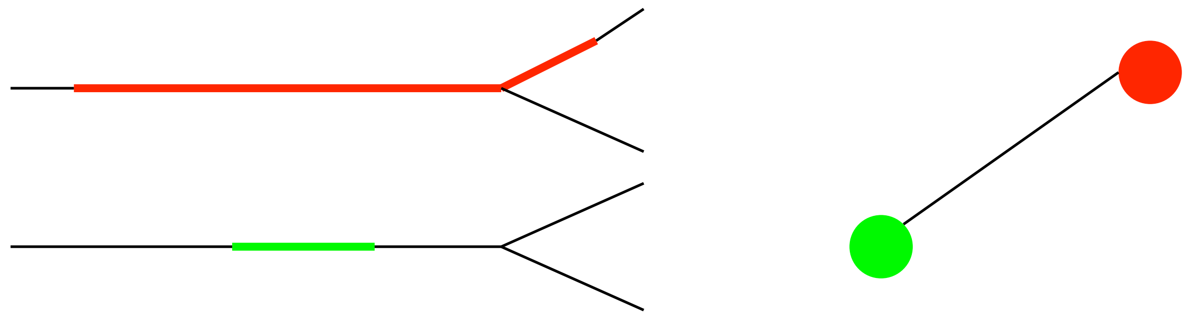
Interval Graph: Branched Genomes



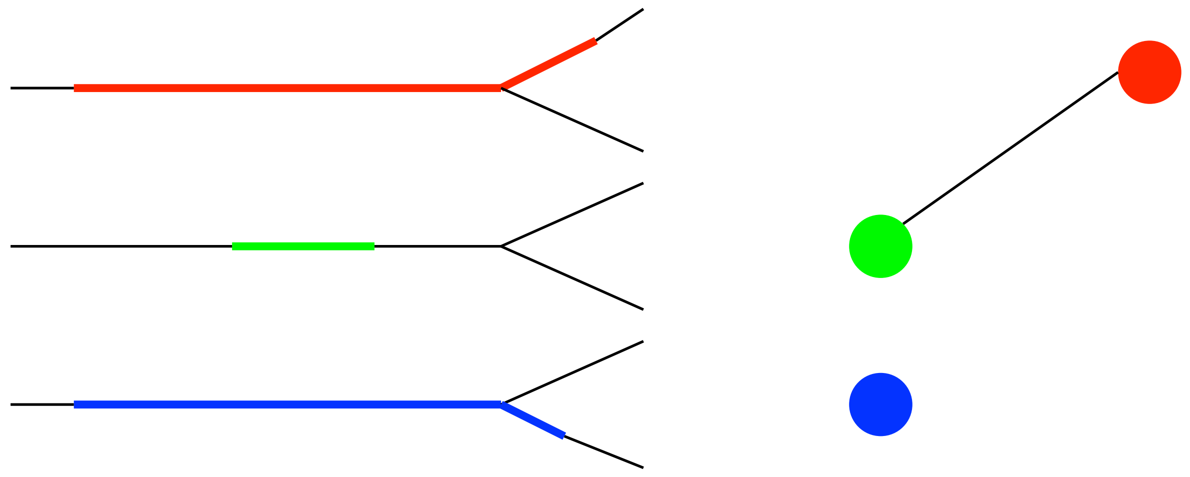
Interval Graph: Branched Genomes



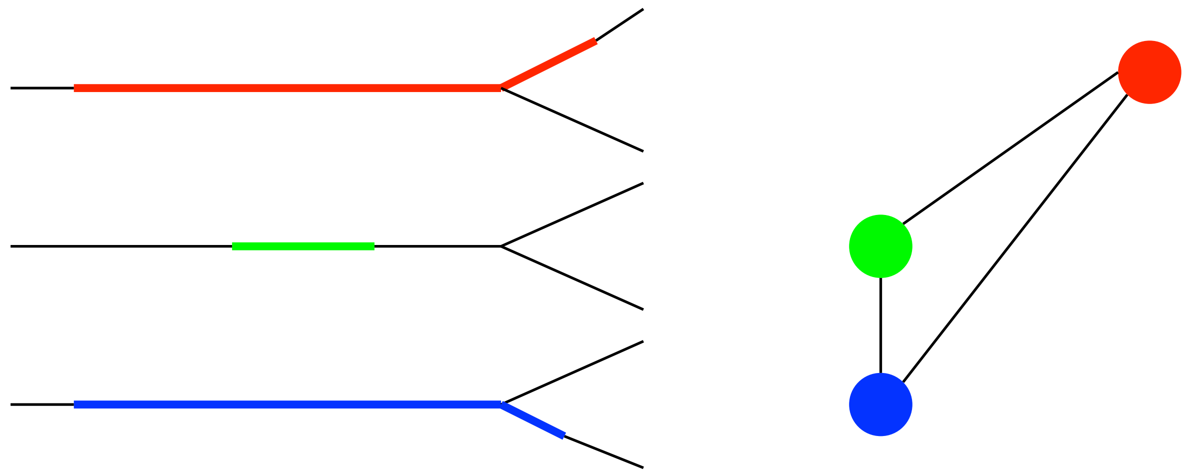
Interval Graph: Branched Genomes



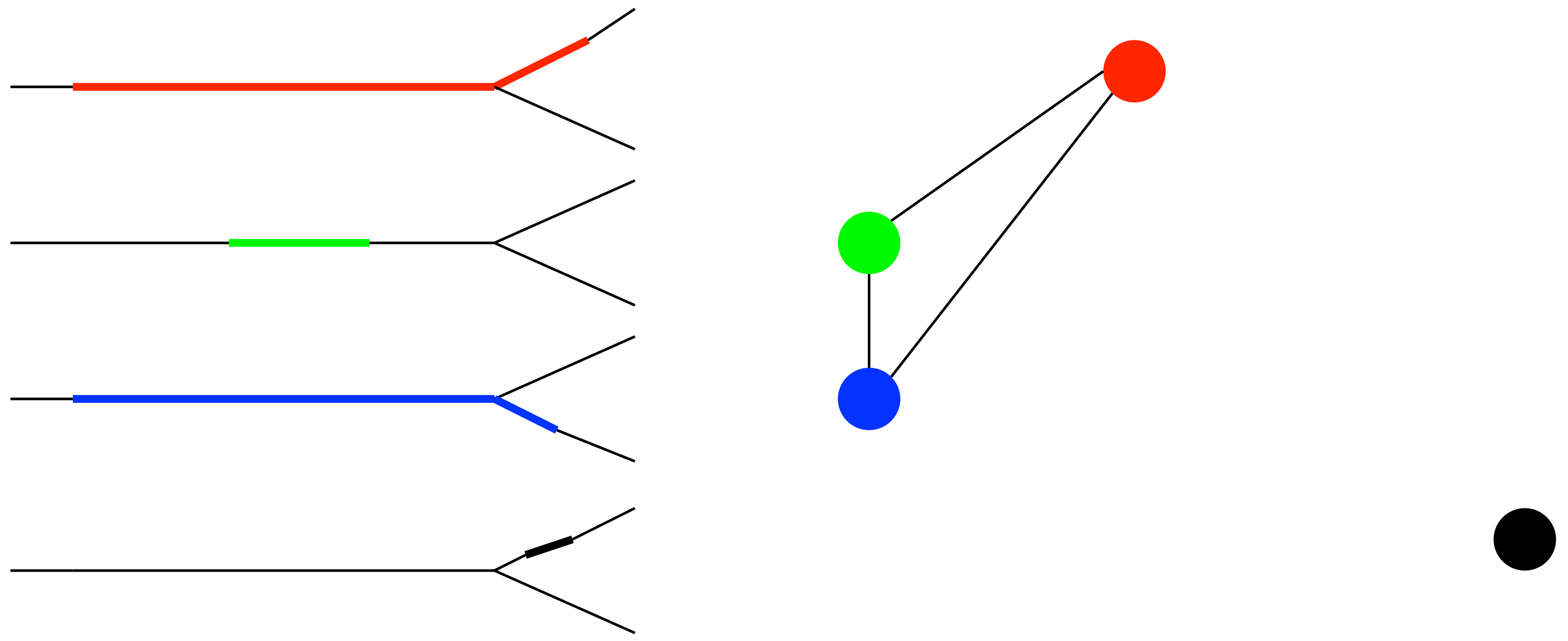
Interval Graph: Branched Genomes



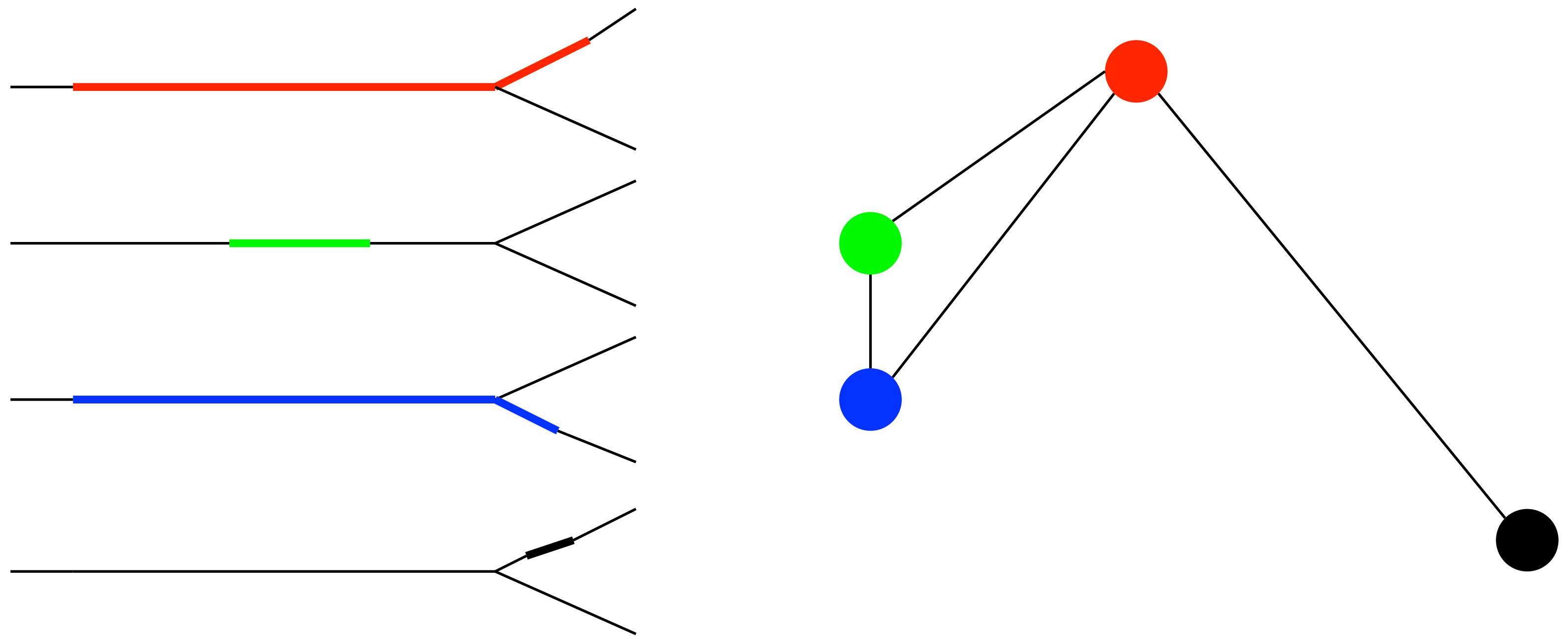
Interval Graph: Branched Genomes



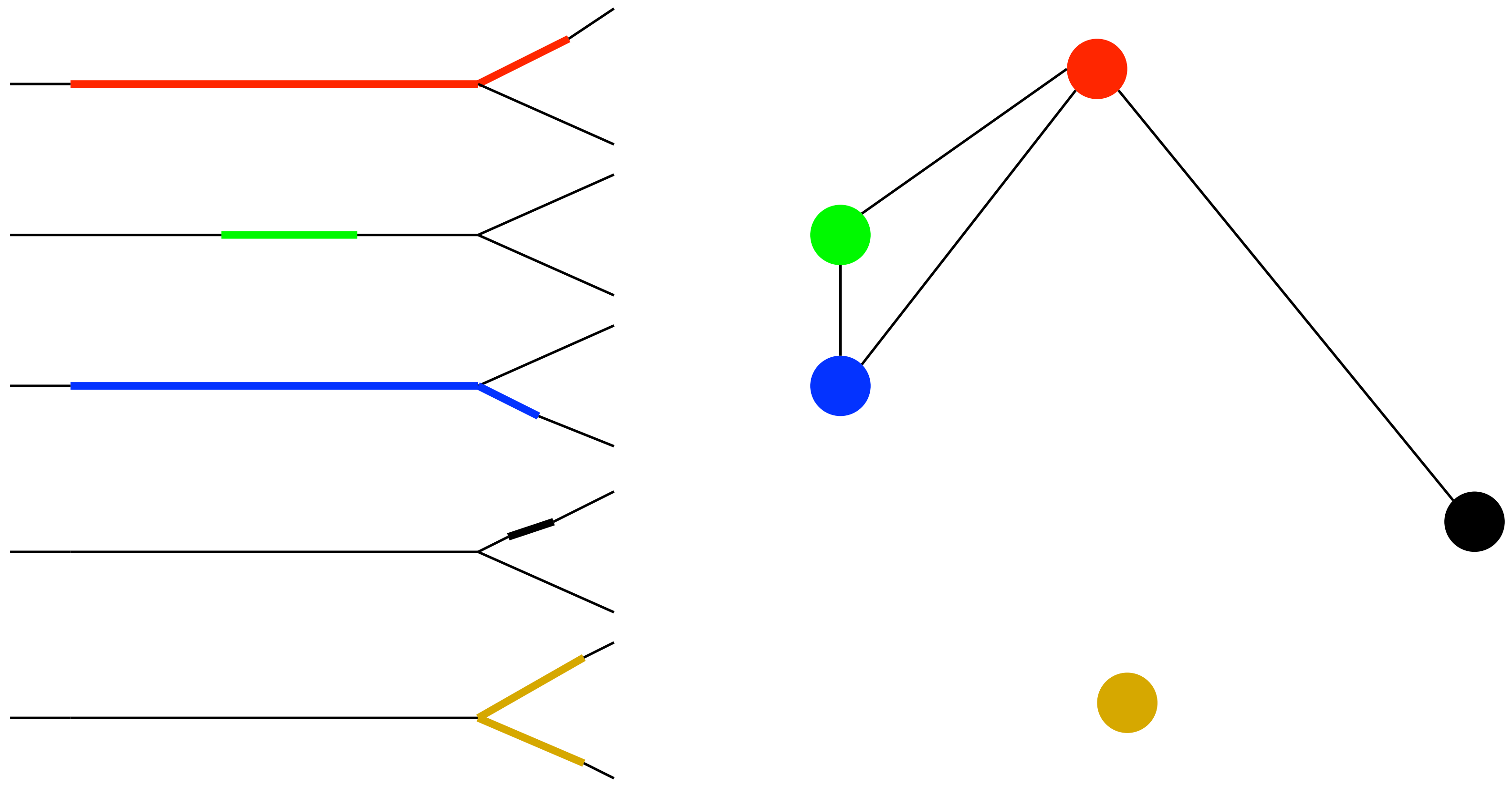
Interval Graph: Branched Genomes



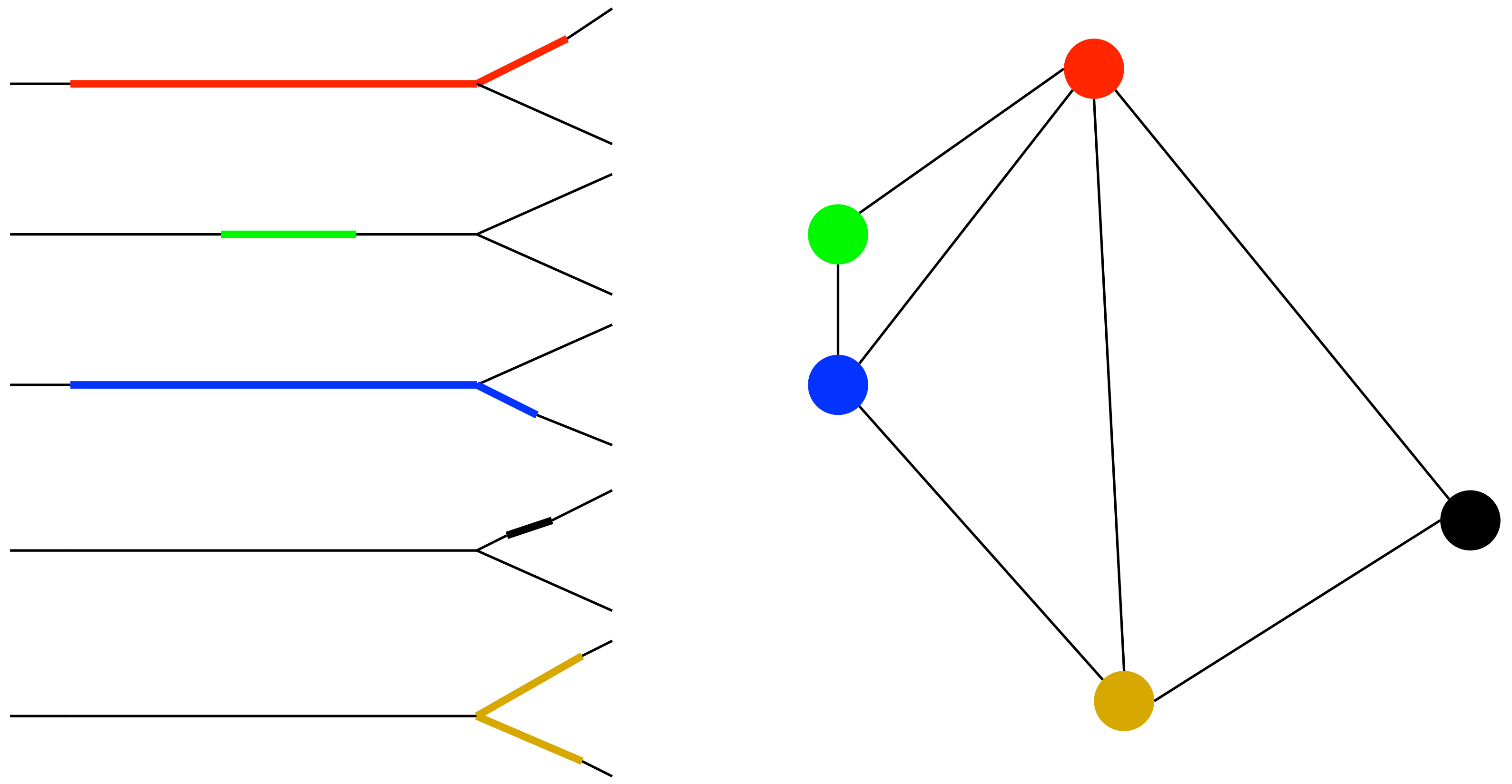
Interval Graph: Branched Genomes



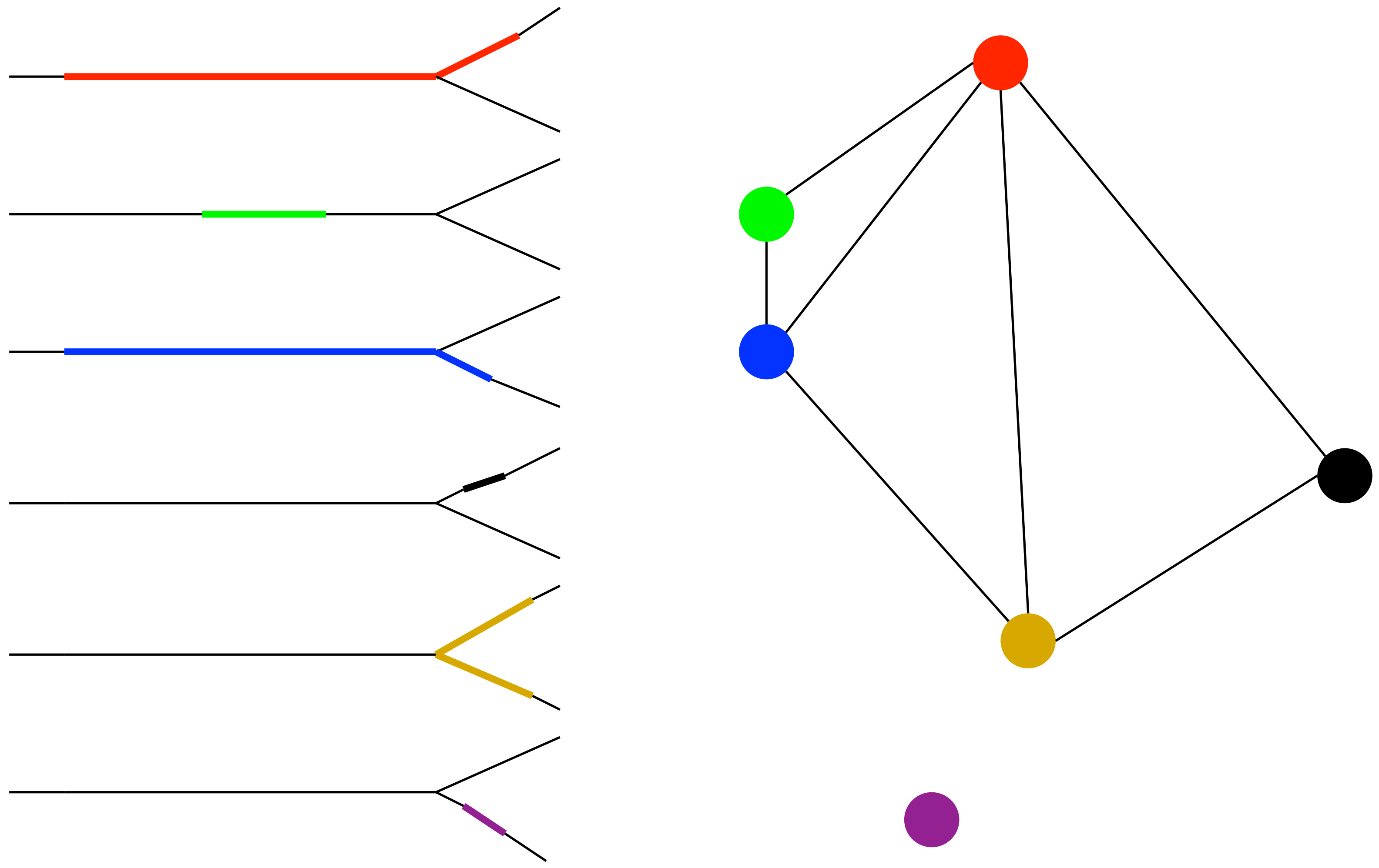
Interval Graph: Branched Genomes



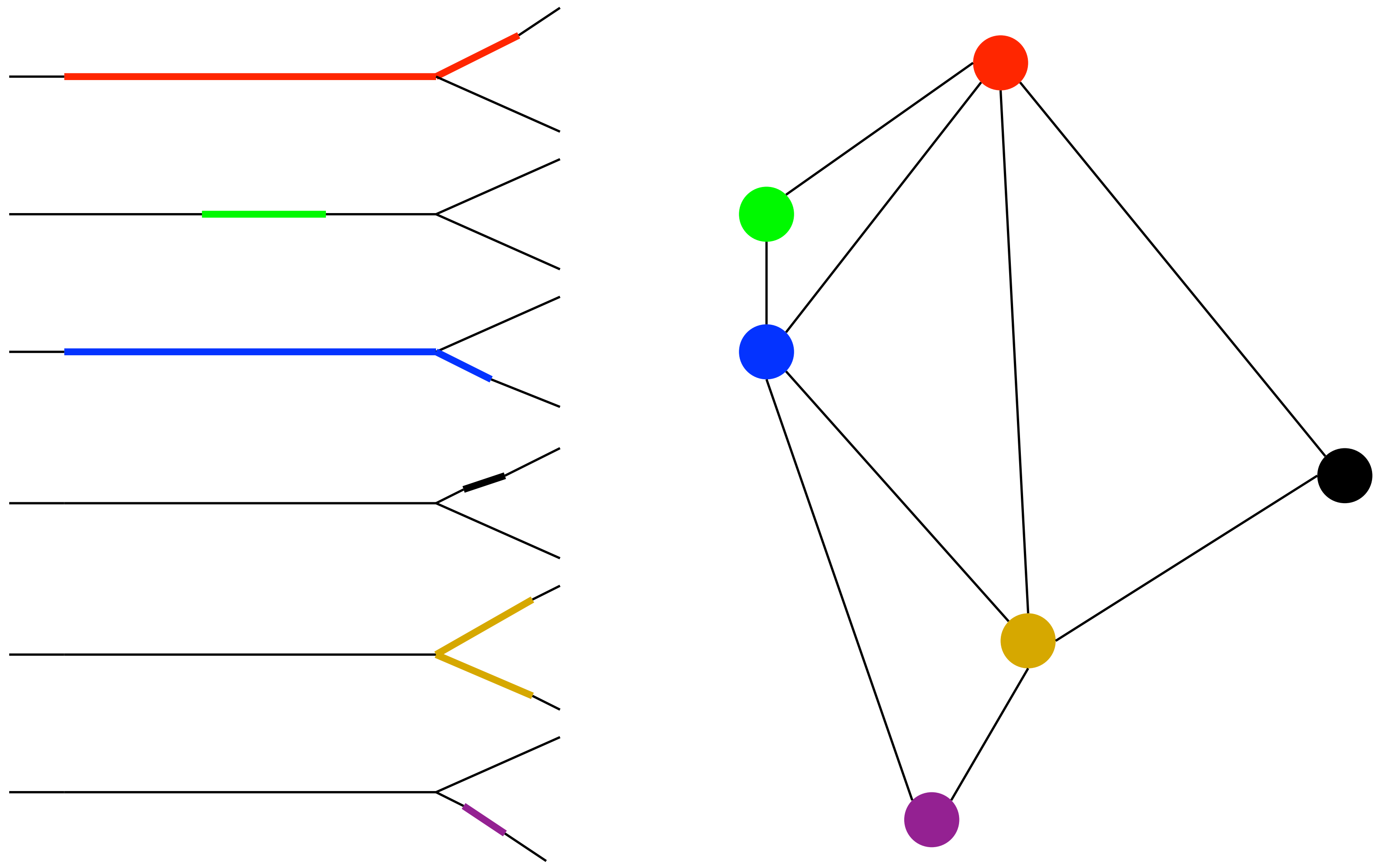
Interval Graph: Branched Genomes



Interval Graph: Branched Genomes

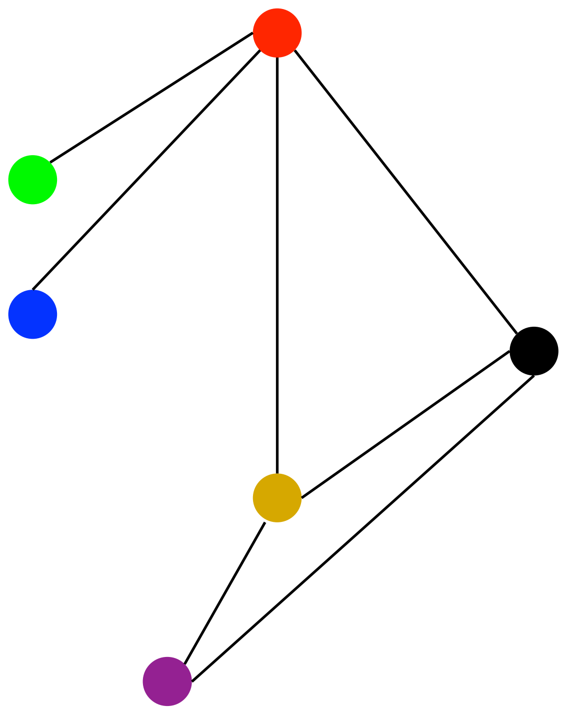


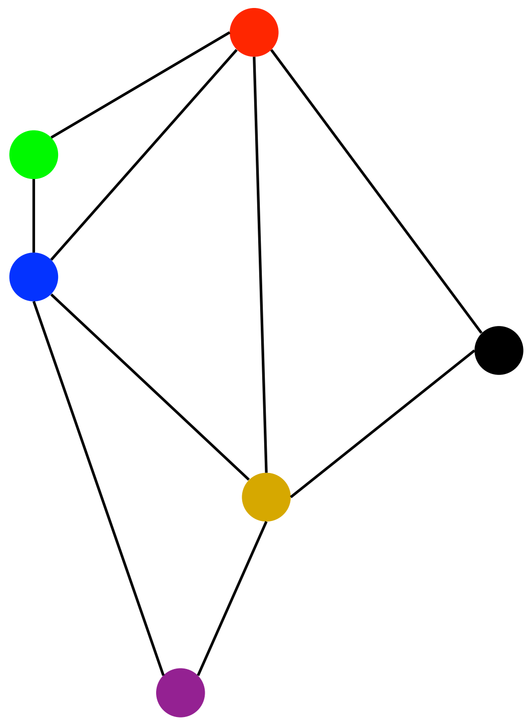
Interval Graph: Branched Genomes



Linear vs. Branched Genomes: Interval Graphs

- Simply by comparing the structure of the two interval graphs, Benzer showed that genomes cannot be branched!


Linear Genome


Branched Genome

Section 4: DNA Sequencing

DNA Sequencing: History

- **Sanger Method** (1977): Labeled ddNTPs terminate DNA copying at random points.
- **Gilbert Method** (1977): Chemical method to cleave DNA at specific points (G, G+A, T+C, C).

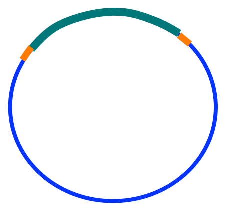
Frederick Sanger

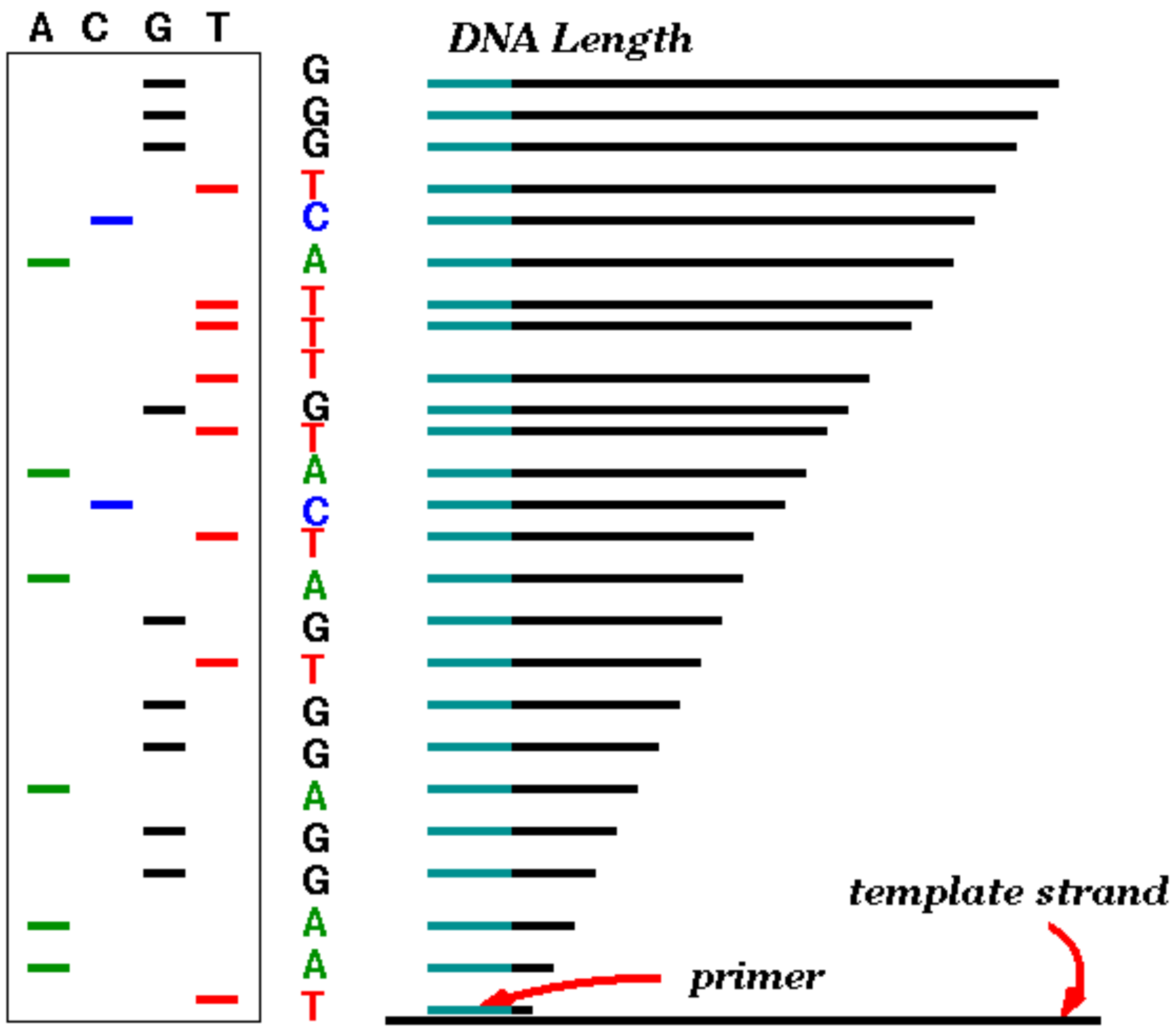
- Both methods generate labeled fragments of varying lengths that are further electrophoresed.

Walter Gilbert

Sanger Method: Generating Read

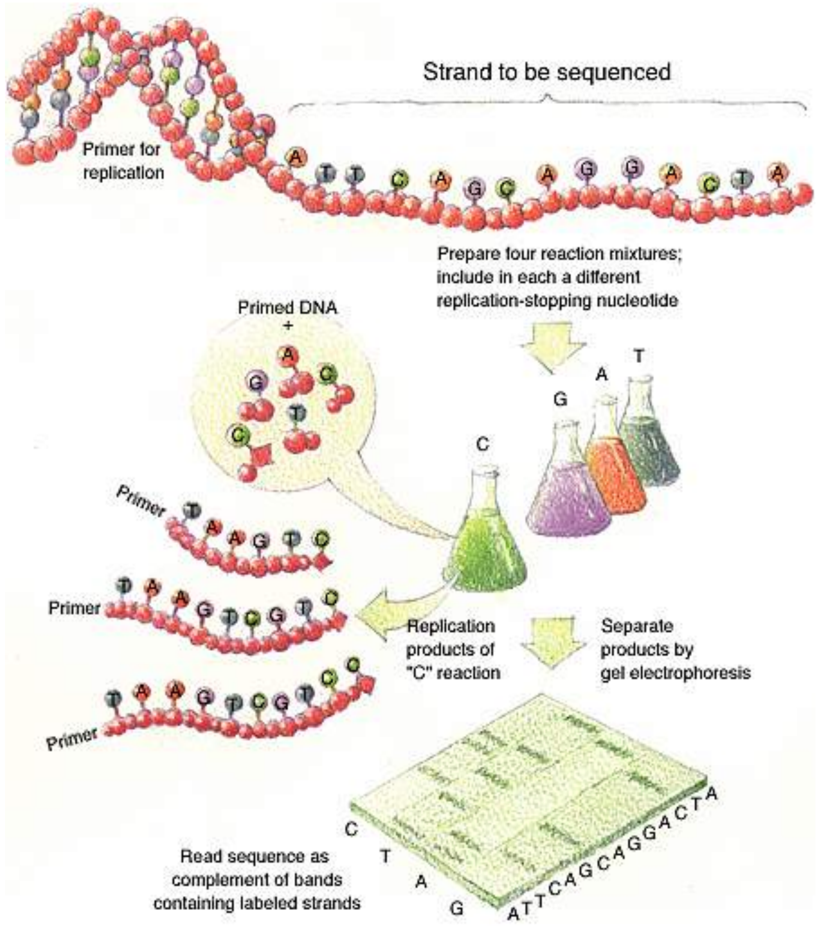
1. Start at primer (restriction site).
2. Grow DNA chain.
3. Include ddNTPs.
4. Stop reaction at all possible points.
5. Separate products by length, using gel electrophoresis.





Sanger Method: Sequencing

- Shear DNA into millions of small fragments.
- Read 500 – 700 nucleotides at a time from the small fragments.


Strand to be sequenced
Primer for replication
A T T C A G C A G G A C T A
Prepare four reaction mixtures; include in each a different replication-stopping nucleotide
Primed DNA +
A C G T
G A T C
Primer
Replication products of "C" reaction
Separate products by gel electrophoresis
Read sequence as complement of bands containing labeled strands
C T A G
ATTCAGCAGGACTA

Fragment Assembly

- **Computational Challenge:** assemble individual short fragments (“reads”) into a single genomic sequence (“superstring”).

- Until late 1990s the so called “shotgun fragment assembly” of the human genome was viewed as an intractable problem, because it required so much work for a large genome.

- Our computational challenge leads to the formal problem at the beginning of the next section.

Section 5: Shortest Superstring & Traveling Salesman Problems

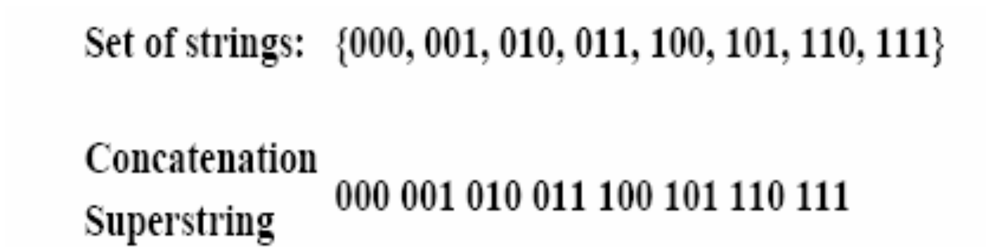
Shortest Superstring Problem (SSP)

- Problem: Given a set of strings, find a shortest string that contains all of them.
- Input: Strings $s_1, s_2, \ldots., s_n$
- Output: A "superstring" s that contains all strings $s_1, s_2, \ldots., s_n$ as substrings, such that the length of s is minimized.

SSP: Example

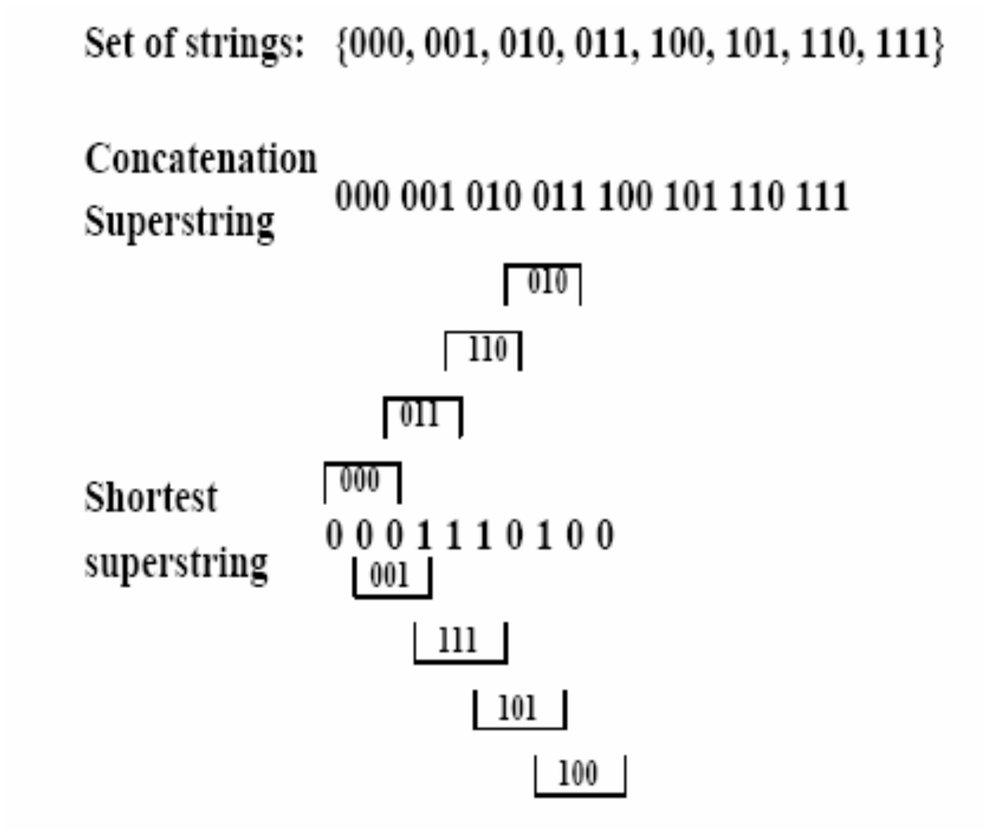
Set of strings: {000, 001, 010, 011, 100, 101, 110, 111}

SSP: Example



Set of strings:	{000, 001, 010, 011, 100, 101, 110, 111}
Concatenation Superstring	000 001 010 011 100 101 110 111

SSP: Example

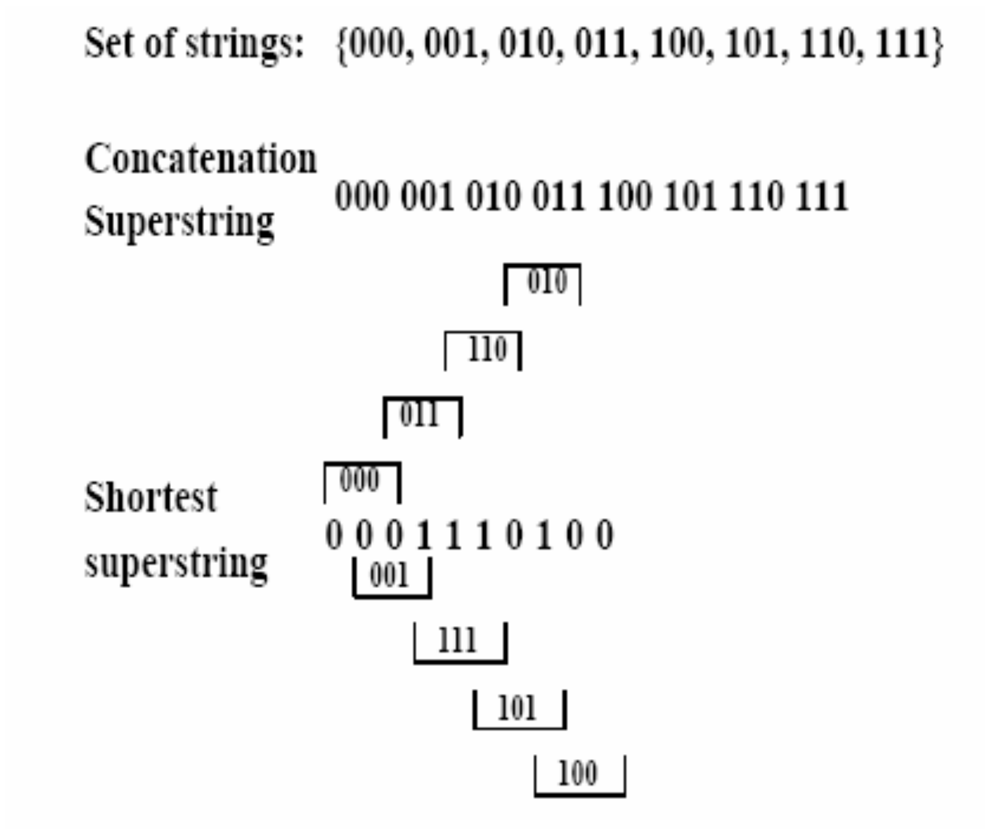


Set of strings: {000, 001, 010, 011, 100, 101, 110, 111}

Concatenation Superstring 000 001 010 011 100 101 110 111

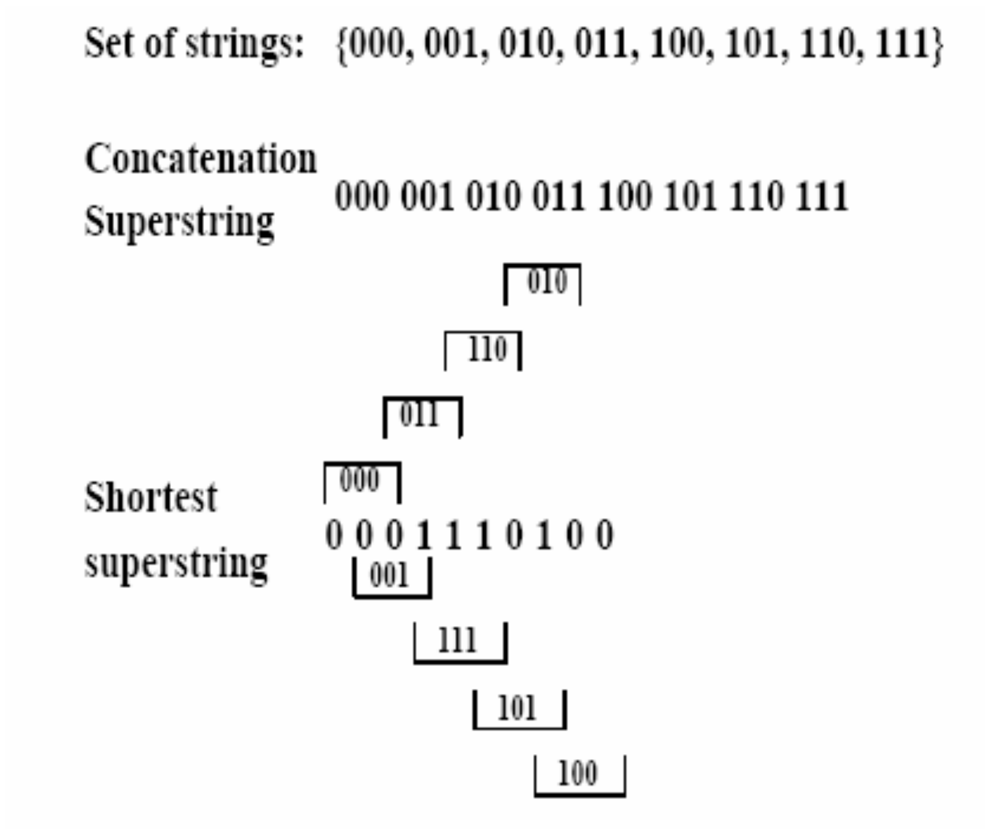
Shortest superstring 0 0 0 1 1 1 0 1 0 0

SSP: Example



- So our greedy guess of concatenating all the strings together turns out to be substantially suboptimal (length 24 vs. 10).

SSP: Example



- So our greedy guess of concatenating all the strings together turns out to be substantially suboptimal (length 24 vs. 10).

- **Note**: The strings here are just the integers from 1 to 8 in base-2 notation.

SSP: Issues

- **Complexity**: NP-complete (in a few slides).

- Also, this formulation does not take into account the possibility of sequencing errors, and it is difficult to adapt to handle that consideration.

The Overlap Function

- Given strings s_i and s_j , define *overlap*(s_i , s_j) as the length of the longest prefix of s_j that matches a suffix of s_i .

The Overlap Function

- Given strings s_i and s_j , define *overlap*(s_i, s_j) as the length of the longest prefix of s_j that matches a suffix of s_i .

- Example:
  - s_1 = aaaggcatcaaatctaaaggcatcaaa
  - s_2 = aagcatcaaatctaaaggcatcaaa

The Overlap Function

- Given strings s_i and s_j , define *overlap*(s_i, s_j) as the length of the longest prefix of s_j that matches a suffix of s_i .

- Example:
  - s_1 = aaaggcatcaaatctaaaggcatcaaa
  - s_2 = aagcatcaaatctaaaggcatcaaa

```
aaaggcatcaaatctaaaggcatcaaa
               aaaggcatcaaatctaaaggcatcaaa
```

The Overlap Function

- Given strings s_i and s_j , define *overlap*(s_i , s_j) as the length of the longest prefix of s_j that matches a suffix of s_i .

- Example:
  - s_1 = aaaggcatcaaatctaaaggcatcaaa
  - s_2 = aagcatcaaatctaaaggcatcaaa

```
aaaggcatcaaatctaaaggcatcaaa
               aaaggcatcaaatctaaaggcatcaaa
```

  - Therefore, *overlap*(s_1 , s_2) = 12.

Why is SSP an NP-Complete Problem?

- Construct a graph G as follows:
  - The n vertices represent the n strings s_1, s_2, …., s_n.
  - For every pair of vertices s_i and s_j, insert an edge of length *overlap*(s_i, s_j) connecting the vertices.
- Then finding the shortest superstring will correspond to finding the shortest Hamiltonian path in G.
- But this is the **Traveling Salesman Problem** (TSP), which we know to be NP-complete.
  - Hence SSP must also be NP-Complete!

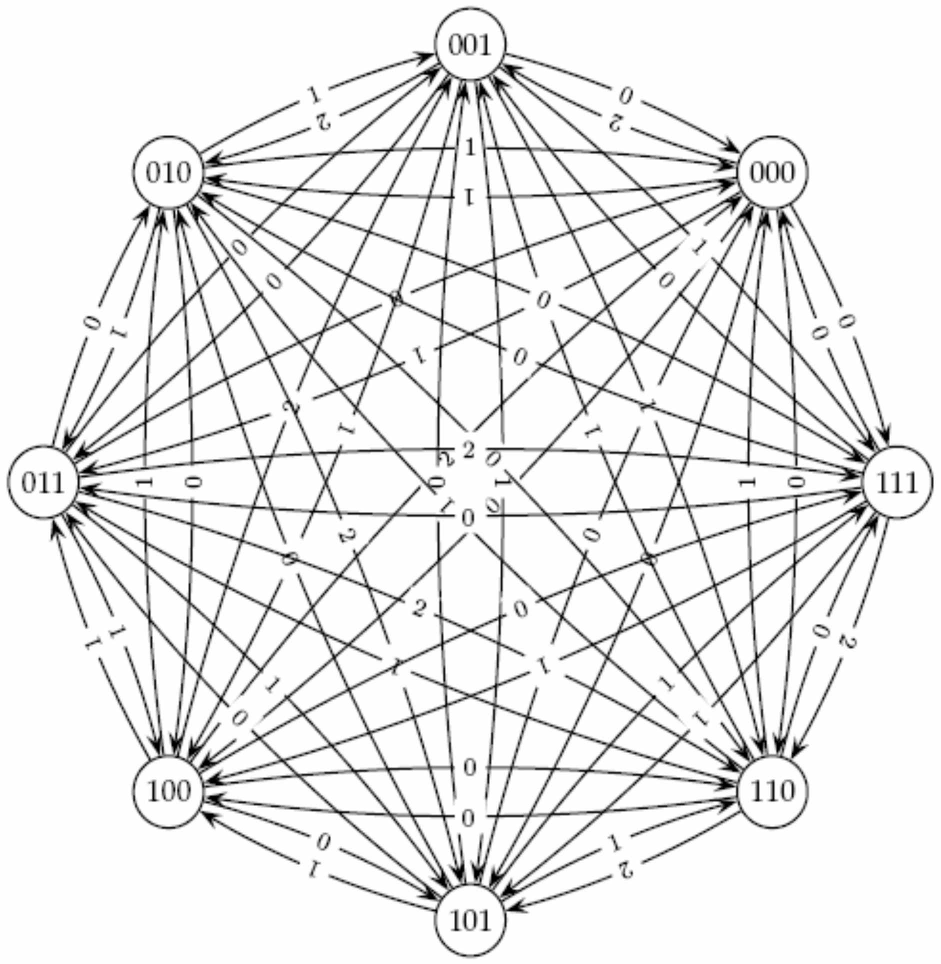
- **Note**: We also need to show that any TSP can be formulated as a SSP (not difficult).

Reducing SSP to TSP: Example 1

- Take our previous set of strings S = {000, 001, 010, 011, 100, 101, 110, 111}.

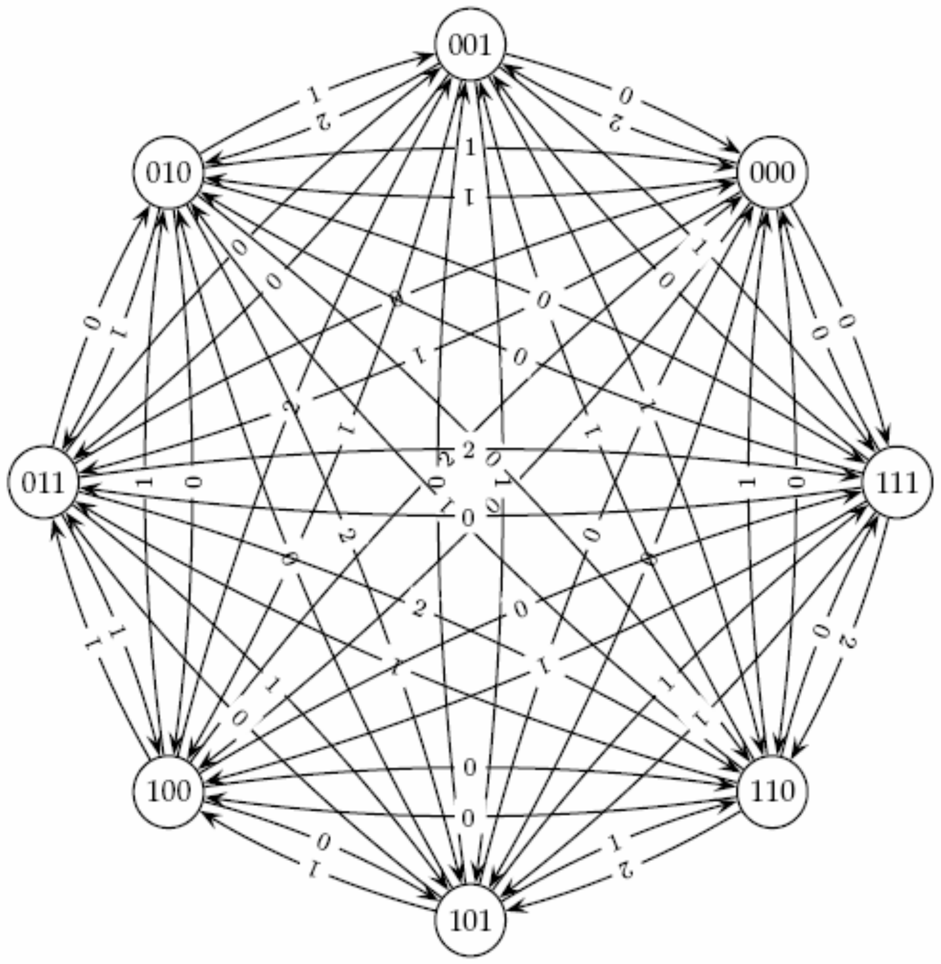
Reducing SSP to TSP: Example 1

- Take our previous set of strings S = {000, 001, 010, 011, 100, 101, 110, 111}.
- Then the graph for S is given at right.



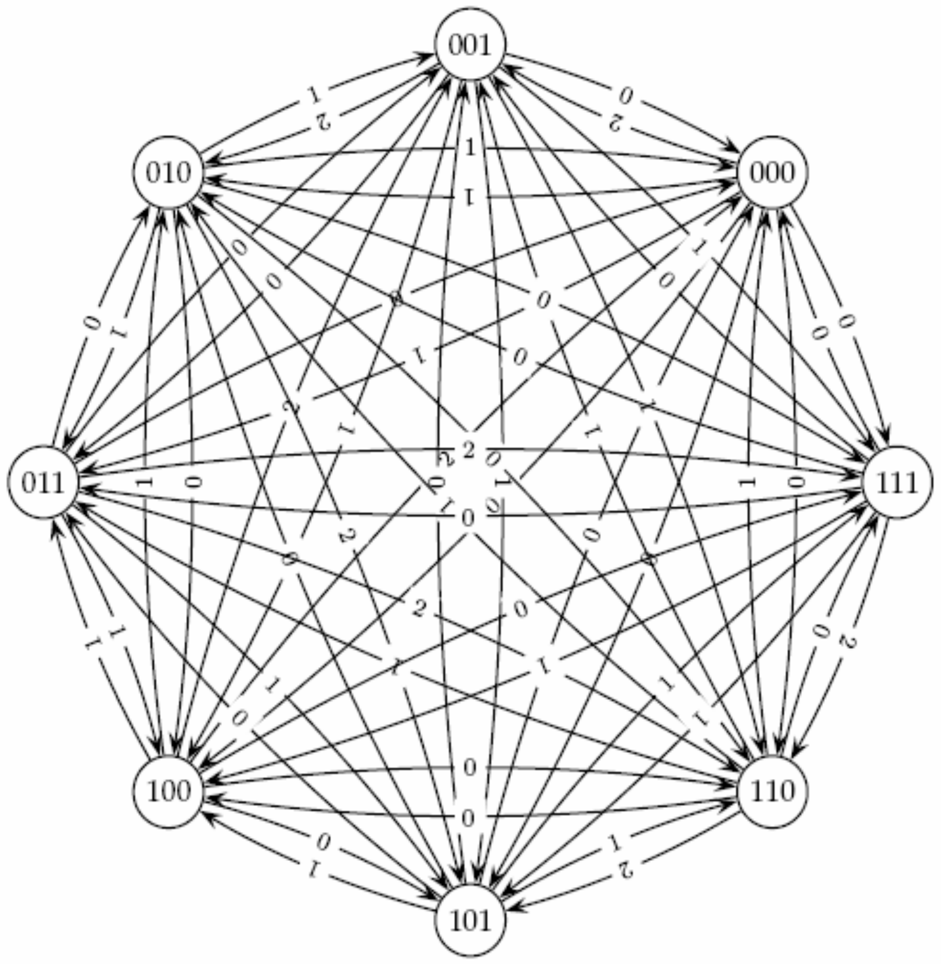
Reducing SSP to TSP: Example 1

- Take our previous set of strings S = {000, 001, 010, 011, 100, 101, 110, 111}.
- Then the graph for S is given at right.
- One minimal Hamiltonian path gives our previous superstring, 0001110100.



Reducing SSP to TSP: Example 1

- Take our previous set of strings S = {000, 001, 010, 011, 100, 101, 110, 111}.
- Then the graph for S is given at right.
- One minimal Hamiltonian path gives our previous superstring, 0001110100.
- Check that this works!

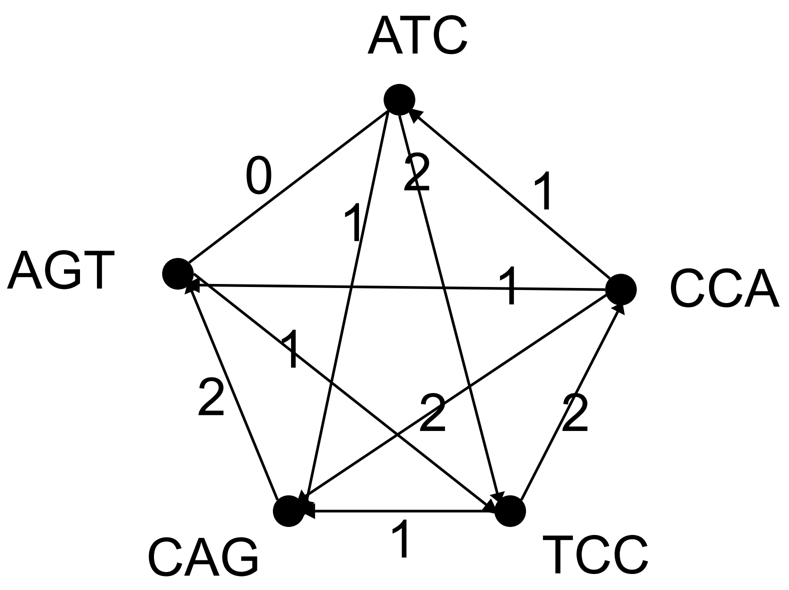


Reducing SSP to TSP: Example 2

- S = {ATC, CCA, CAG, TCC, AGT}

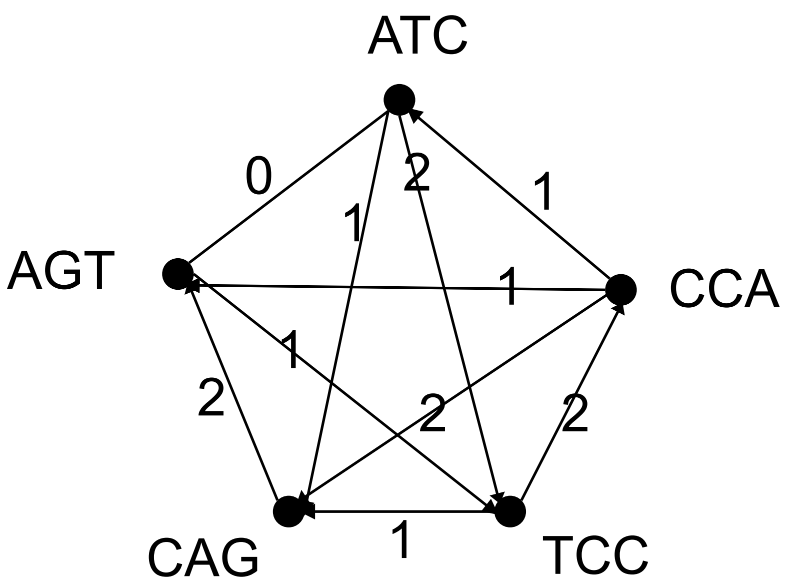
Reducing SSP to TSP: Example 2

- S = {ATC, CCA, CAG, TCC, AGT}
- The graph is provided at right.



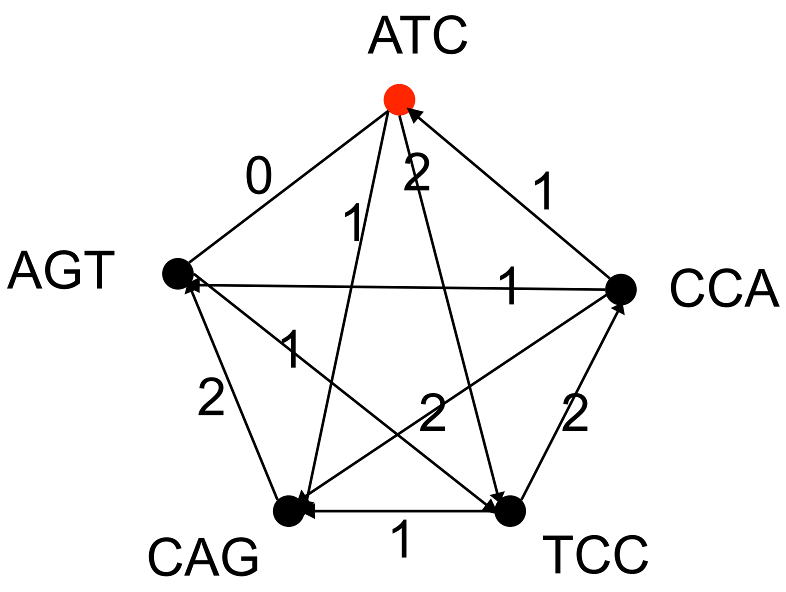
Reducing SSP to TSP: Example 2

- S = {ATC, CCA, CAG, TCC, AGT}
- The graph is provided at right.
- A minimal Hamiltonian path gives as shortest superstring ATCCAGT.



Reducing SSP to TSP: Example 2

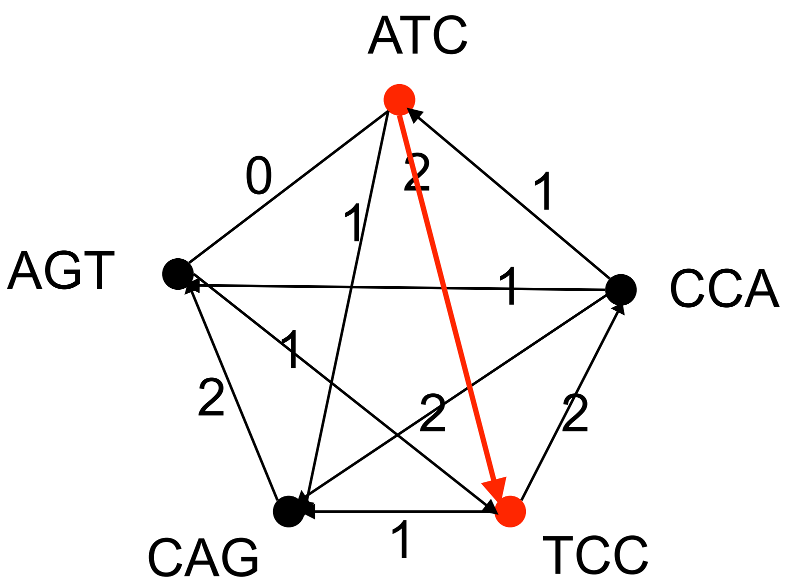
- S = {ATC, CCA, CAG, TCC, AGT}
- The graph is provided at right.
- A minimal Hamiltonian path gives as shortest superstring ATCCAGT.



ATC

Reducing SSP to TSP: Example 2

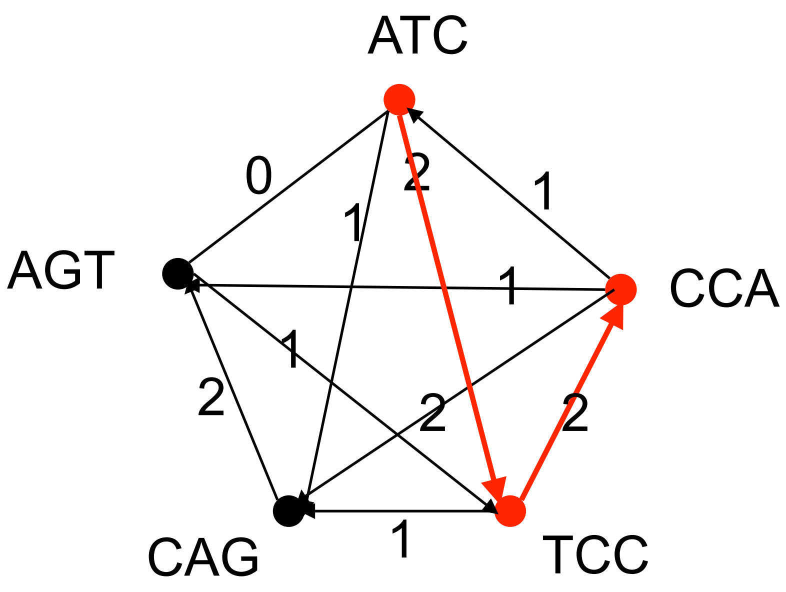
- S = {ATC, CCA, CAG, TCC, AGT}
- The graph is provided at right.
- A minimal Hamiltonian path gives as shortest superstring ATCCAGT.



ATCC

Reducing SSP to TSP: Example 2

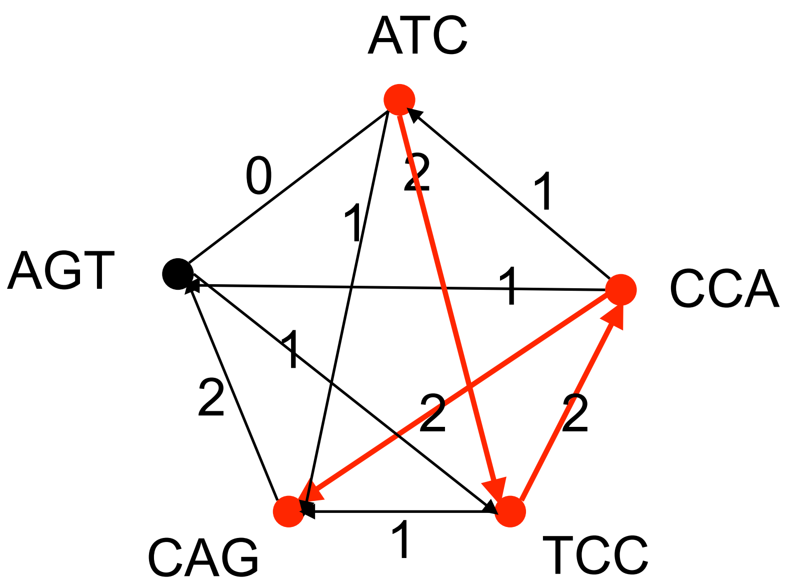
- S = {ATC, CCA, CAG, TCC, AGT}
- The graph is provided at right.
- A minimal Hamiltonian path gives as shortest superstring ATCCAGT.



ATCCA

Reducing SSP to TSP: Example 2

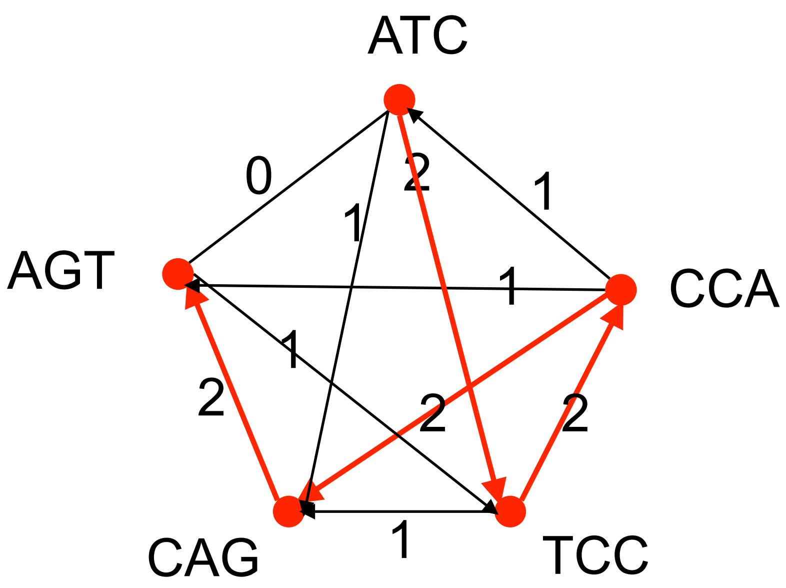
- S = {ATC, CCA, CAG, TCC, AGT}
- The graph is provided at right.
- A minimal Hamiltonian path gives as shortest superstring ATCCAGT.



ATCCAG

Reducing SSP to TSP: Example 2

- S = {ATC, CCA, CAG, TCC, AGT}
- The graph is provided at right.
- A minimal Hamiltonian path gives as shortest superstring ATCCAGT.



ATCCAGT

Section 6: Sequencing By Hybridization

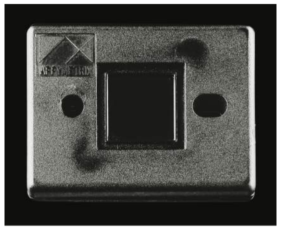
Sequencing by Hybridization (SBH): History

- **1988:** SBH is suggested as an an alternative sequencing method. Nobody believes it will ever work.

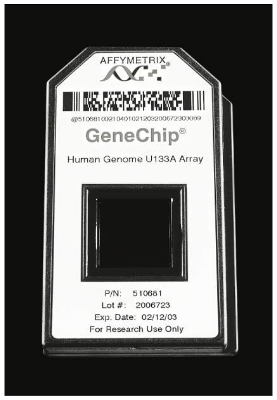
- **1991:** Light directed polymer synthesis is developed by Steve Fodor and colleagues.

- **1994:** Affymetrix develops the first 64-kb DNA microarray.

First microarray prototype (1989)



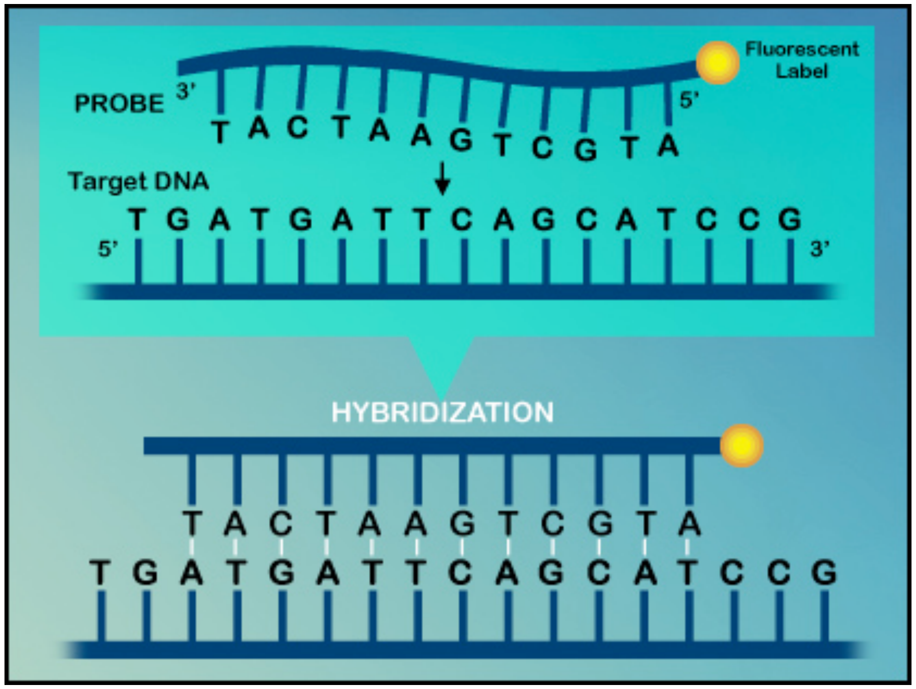
First commercial DNA microarray prototype w/16,000 features (1994)



500,000 features per chip (2002)

How SBH Works

- Attach all possible DNA probes of length *l* to a flat surface, each probe at a distinct known location. This set of probes is called a **DNA array**.
- Apply a solution containing fluorescently labeled DNA fragment to the array.
- The DNA fragment hybridizes with those probes that are complementary to substrings of length *l* of the fragment.



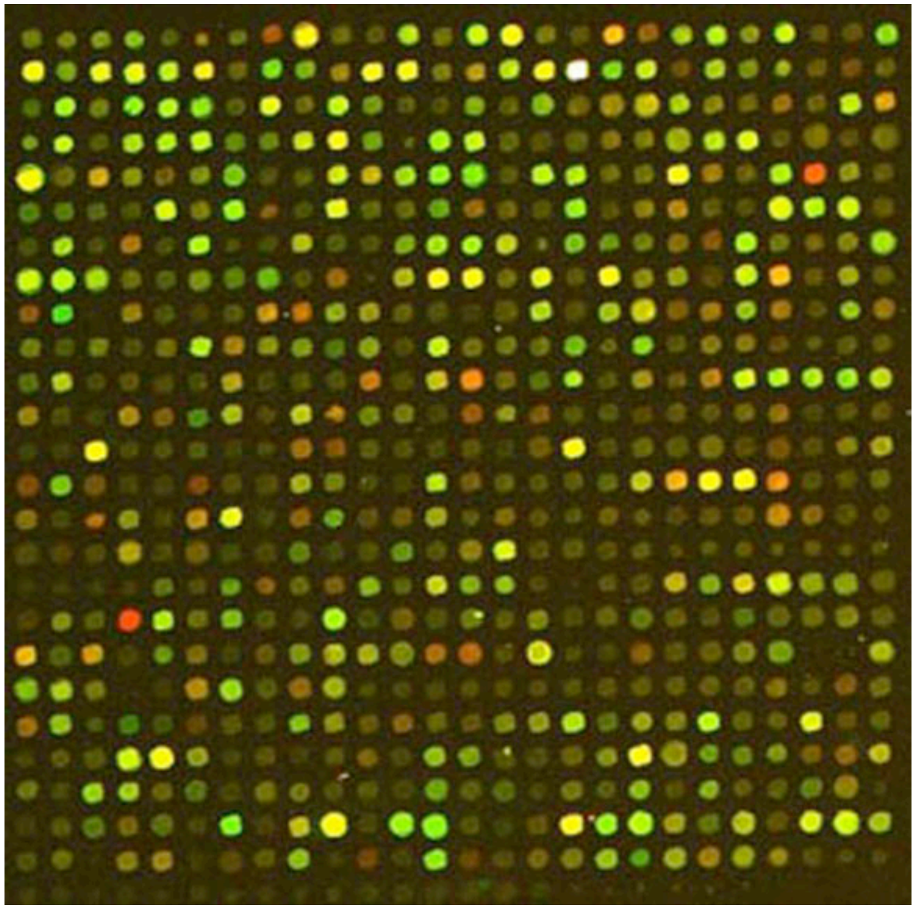
Hybridization of a DNA Probe

http://members.cox.net/amgough/Fanconi-genetics-PGD.htm

How SBH Works

- Using a spectroscopic detector, determine which probes hybridize to the DNA fragment to obtain the *l*–mer composition of the target DNA fragment.

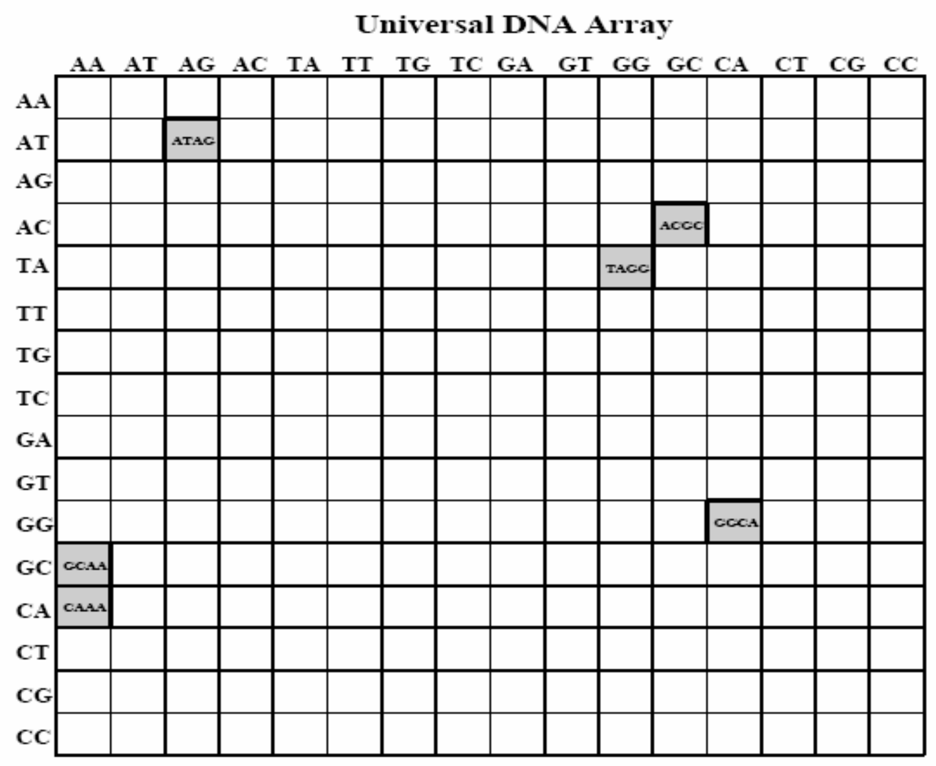
- Reconstruct the sequence of the target DNA fragment from the *l*-mer composition.


DNA Microarray

http://www.wormbook.org/chapters/www_germlinegenomics/germlinegenomics.html

How SBH Works: Example

- Say our DNA fragment hybridizes to indicate that it contains the following substrings: GCAA, CAAA, ATAG, TAGG, ACGC, GGCA.
- Then the most logical explanation is that our fragment is the shortest superstring containing these strings!
- Here the superstring is: ATAGGCAAACGC



Universal DNA Array

	AA	AT	AG	AC	TA	TT	TG	TC	GA	GT	GG	GC	CA	CT	CG	CC
AA																
AT			ATAG													
AG																
AC												ACGC				
TA											TAGG					
TT																
TG																
TC																
GA																
GT																
GG													GGCA			
GC	GCAA															
CA	CAAA															
CT																
CG																
CC																

DNA Microarray Interpreted

l-mer Composition

- ***Spectrum*(s, *l*)**: The *unordered* multiset of all *l*-mers in a string *s* of length *n*.
- The order of individual elements in *Spectrum*(*s, l*) does not matter.

l-mer Composition

- ***Spectrum*(s, *l*)**: The *unordered* multiset of all *l*-mers in a string *s* of length *n*.
- The order of individual elements in *Spectrum*(*s*, *l*) does not matter.
- For *s* = TATGGTGC all of the following are equivalent representations of *Spectrum*(*s*, *3*)*:*

l-mer Composition

- ***Spectrum*(*s, l*)**: The *unordered* multiset of all *l*-mers in a string *s* of length *n*.
- The order of individual elements in *Spectrum*(*s, l*) does not matter.
- For *s* = TATGGTGC all of the following are equivalent representations of *Spectrum*(*s, 3*)*:*

  {TAT, ATG, TGG, GGT, GTG, TGC}

l-mer Composition

- ***Spectrum*(s, *l*)**: The *unordered* multiset of all *l*-mers in a string *s* of length *n*.
- The order of individual elements in *Spectrum*(*s, l*) does not matter.
- For *s* = TATGGTGC all of the following are equivalent representations of *Spectrum*(*s, 3*)*:*

  {TAT, ATG, TGG, GGT, GTG, TGC}

  {ATG, GGT, GTG, TAT, TGC, TGG}

l-mer Composition

- ***Spectrum*(s, *l*)**: The *unordered* multiset of all *l*-mers in a string *s* of length *n*.
- The order of individual elements in *Spectrum*(*s, l*) does not matter.
- For *s* = TATGGTGC all of the following are equivalent representations of *Spectrum*(*s, 3*)*:*

  {TAT, ATG, TGG, GGT, GTG, TGC}

  {ATG, GGT, GTG, TAT, TGC, TGG}

  {TGG, TGC, TAT, GTG, GGT, ATG}

l-mer Composition

- ***Spectrum*(s, *l*)**: The *unordered* multiset of all *l*-mers in a string *s* of length *n*.
- The order of individual elements in *Spectrum*(*s, l*) does not matter.
- For *s* = TATGGTGC all of the following are equivalent representations of *Spectrum*(*s, 3*)*:*

  {TAT, ATG, TGG, GGT, GTG, TGC}

  {ATG, GGT, GTG, TAT, TGC, TGG}

  {TGG, TGC, TAT, GTG, GGT, ATG}
- Which ordering do we choose?

l-mer Composition

- ***Spectrum*(s, l)**: The *unordered* multiset of all *l*-mers in a string *s* of length *n*.
- The order of individual elements in *Spectrum*(*s, l*) does not matter.
- For *s* = TATGGTGC all of the following are equivalent representations of *Spectrum*(*s, 3*):

  {TAT, ATG, TGG, GGT, GTG, TGC}

  {ATG, GGT, GTG, TAT, TGC, TGG}

  {TGG, TGC, TAT, GTG, GGT, ATG}
- Which ordering do we choose? Typically the one that is *lexicographic*, meaning in alphabetical order (think of a phonebook).

Different Sequences, Same Spectrum

- Different sequences may share a common spectrum.

- Example:

$$\left.\begin{array}{l} \mathit{Spectrum}(\text{GTATCT}, 2) = \\ \mathit{Spectrum}(\text{GTCTAT}, 2) = \end{array}\right. \{\text{AT}, \text{CT}, \text{GT}, \text{TA}, \text{TC}\}$$

The SBH Problem

- Problem: Reconstruct a string from its *l*-mer composition
- Input: A set *S*, representing all *l*-mers from an (unknown) string *s*.
- Output: A string *s* such that *Spectrum*(*s*, *l*) = *S*
- **Note**: As we have seen, there may be more than one correct answer. Determining which DNA sequence is actually correct is another matter.

SBH: Hamiltonian Path Approach

- Create a graph G as follows:
  - Create one vertex for each member of *S*.
  - Connect vertex *v* to vertex *w* with a *directed* edge (arrow) if the last *l* – 1 elements of *v* match the first *l* – 1 elements of *w*.

- Then a Hamiltonian path in this graph will correspond to a string *s* such that *Spectrum*(*s*, *l*)!

SBH: Hamiltonian Path Approach

- **Example**:

  S = {ATG TGG TGC GTG GGC GCA GCG CGT}

SBH: Hamiltonian Path Approach

- **Example**:

S = {ATG TGG TGC GTG GGC GCA GCG CGT}

SBH: Hamiltonian Path Approach

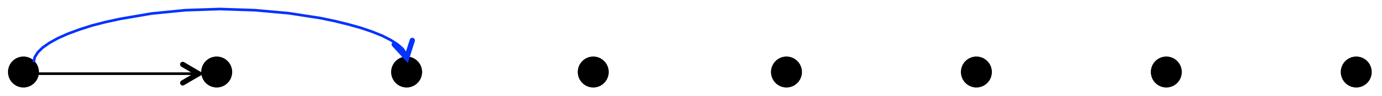
- **Example**:

S = {ATG TGG TGC GTG GGC GCA GCG CGT}

SBH: Hamiltonian Path Approach

- **Example**:

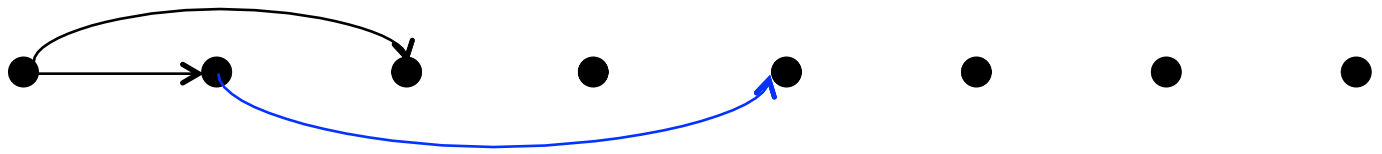
S = {ATG TGG TGC GTG GGC GCA GCG CGT}



SBH: Hamiltonian Path Approach

- **Example**:

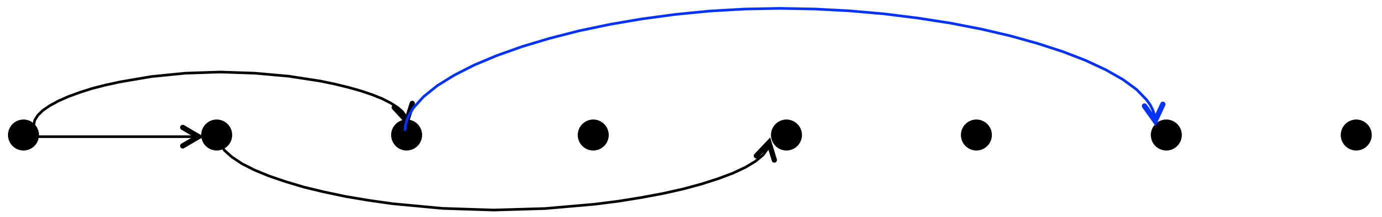
$S = \{\text{ATG} \quad \text{TGG} \quad \text{TGC} \quad \text{GTG} \quad \text{GGC} \quad \text{GCA} \quad \text{GCG} \quad \text{CGT}\}$



SBH: Hamiltonian Path Approach

- **Example**:

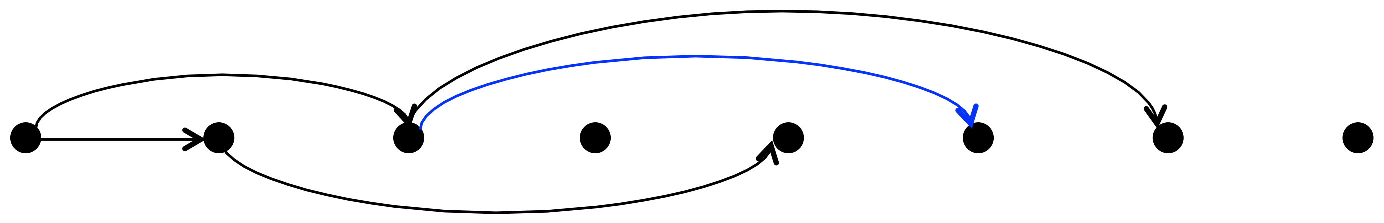
S = {ATG TGG TGC GTG GGC GCA GCG CGT}



SBH: Hamiltonian Path Approach

- **Example**:

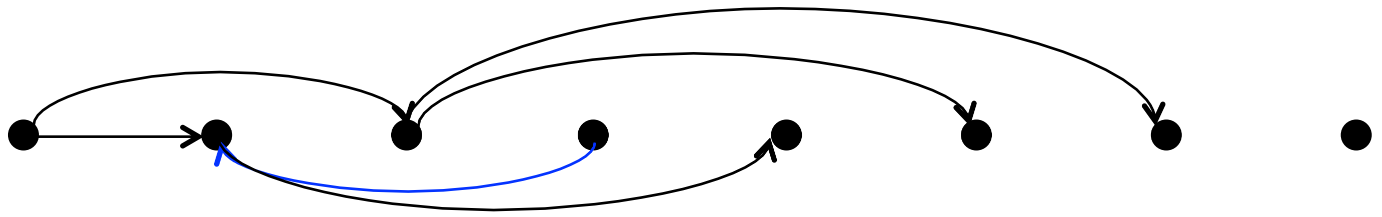
S = {ATG TGG TGC GTG GGC GCA GCG CGT}



SBH: Hamiltonian Path Approach

- **Example**:

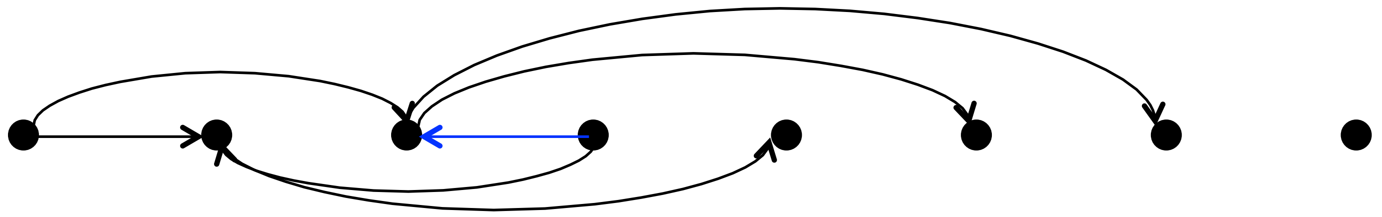
S = {ATG TGG TGC GTG GGC GCA GCG CGT}



SBH: Hamiltonian Path Approach

- **Example**:

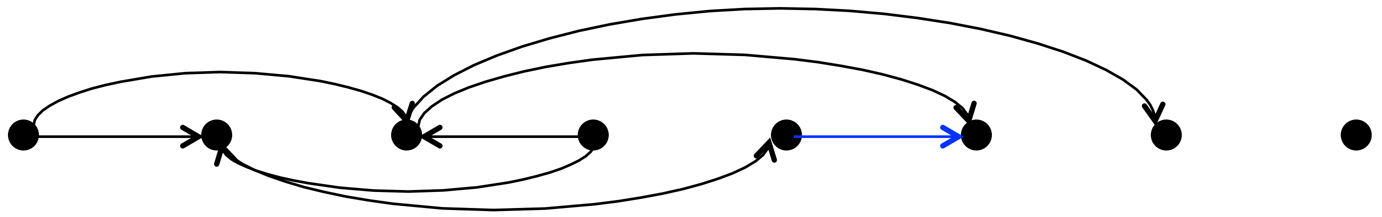
S = {ATG TGG TGC GTG GGC GCA GCG CGT}



SBH: Hamiltonian Path Approach

- **Example**:

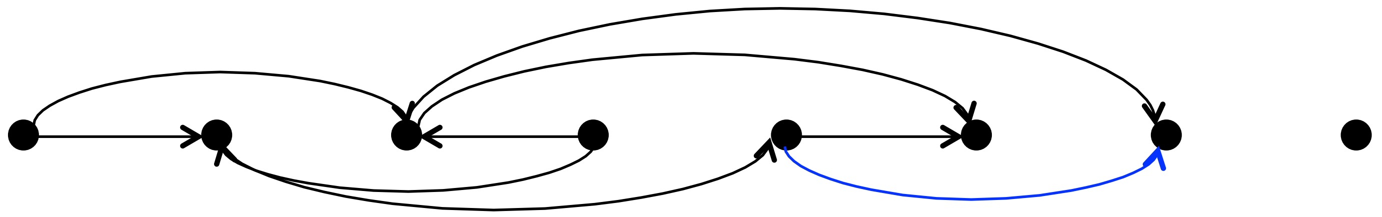
S = {ATG TGG TGC GTG GGC GCA GCG CGT}



SBH: Hamiltonian Path Approach

- **Example**:

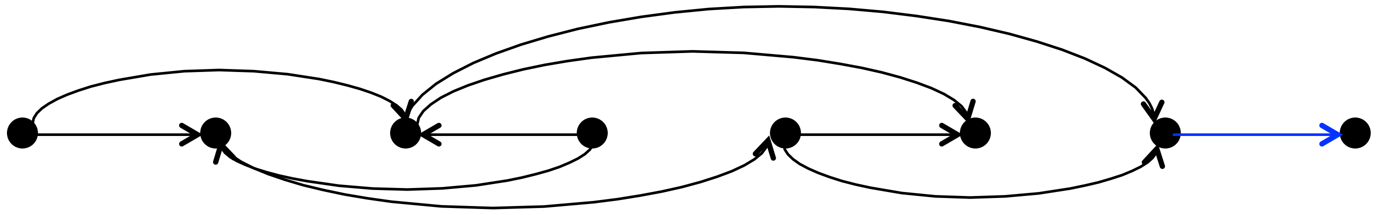
S = {ATG TGG TGC GTG GGC GCA GCG CGT}



SBH: Hamiltonian Path Approach

- **Example**:

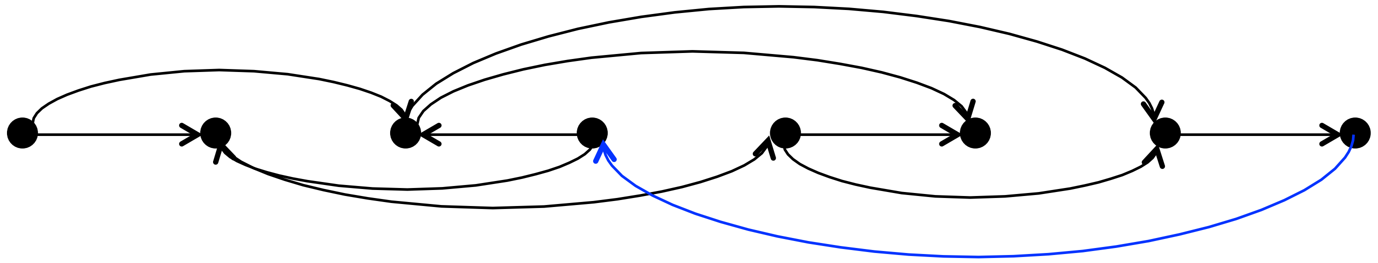
S = {ATG TGG TGC GTG GGC GCA GCG CGT}



SBH: Hamiltonian Path Approach

- **Example**:

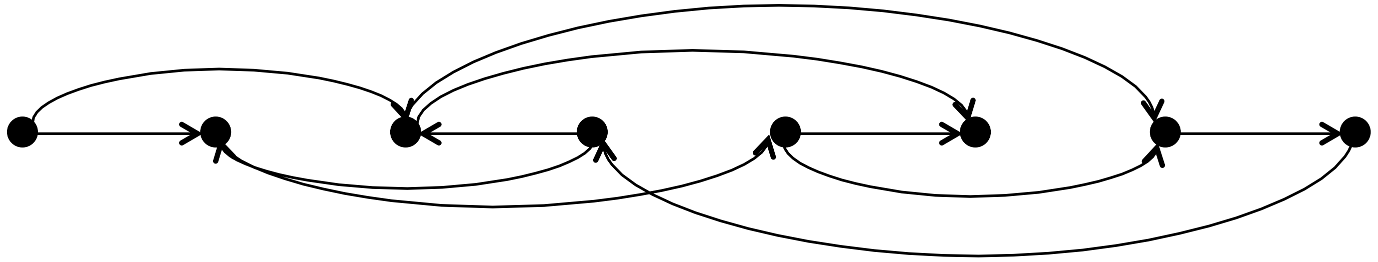
S = {ATG TGG TGC GTG GGC GCA GCG CGT}



SBH: Hamiltonian Path Approach

- **Example**:

S = {ATG TGG TGC GTG GGC GCA GCG CGT}

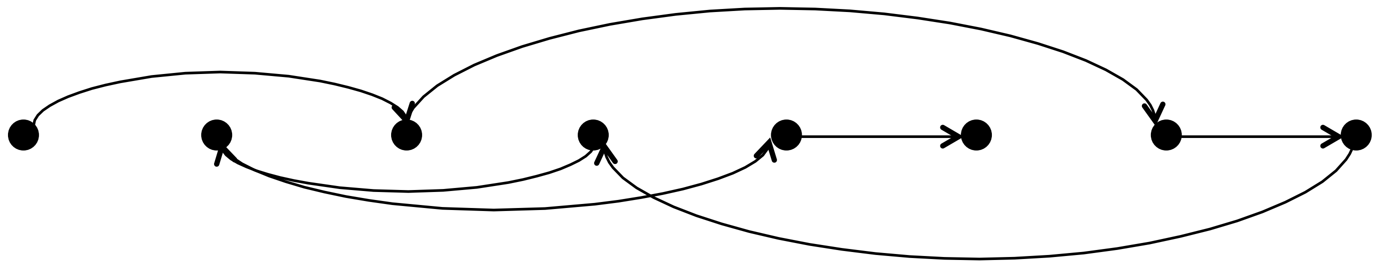


- There are actually two Hamiltonian paths in this graph:

SBH: Hamiltonian Path Approach

- **Example**:

S = {ATG TGG TGC GTG GGC GCA GCG CGT}

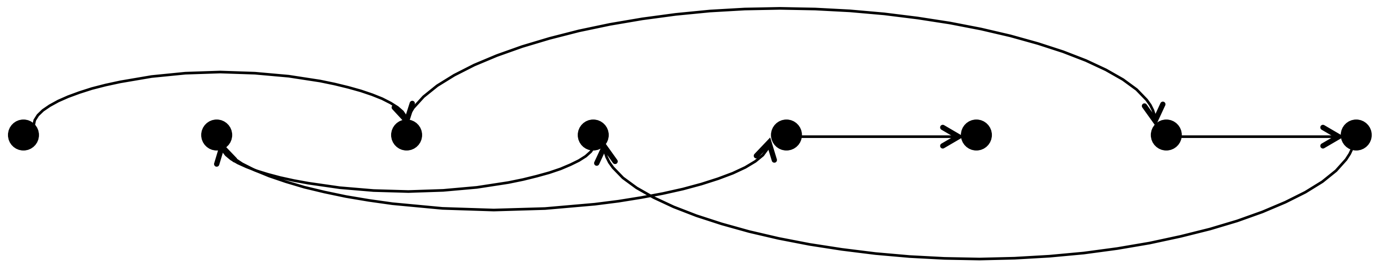


- There are actually two Hamiltonian paths in this graph:
  - Path 1:

SBH: Hamiltonian Path Approach

- **Example**:

S = {ATG TGG TGC GTG GGC GCA GCG CGT}



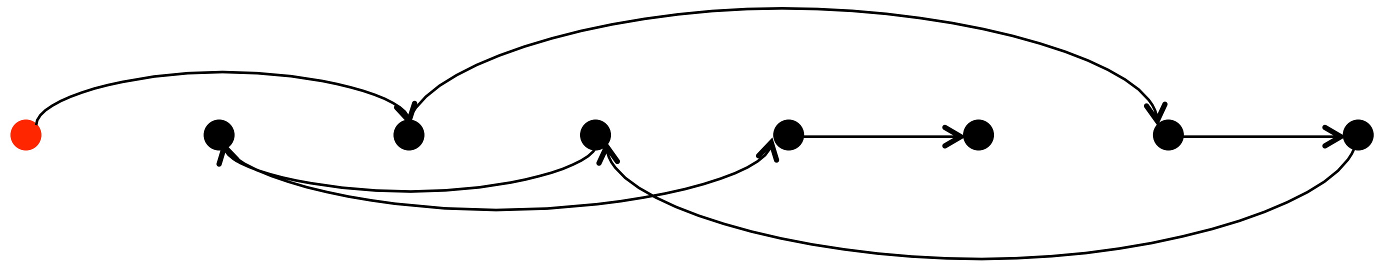
- There are actually two Hamiltonian paths in this graph:
  - Path 1: Gives the string

    $S =$

SBH: Hamiltonian Path Approach

- **Example**:

  S = {ATG TGG TGC GTG GGC GCA GCG CGT}



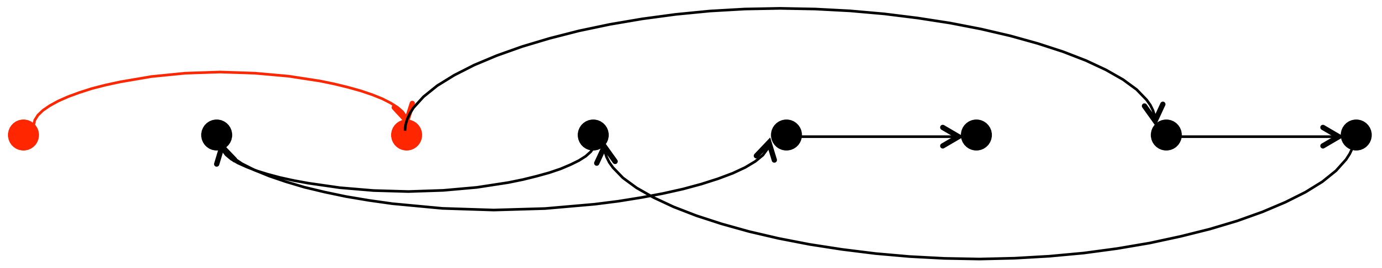
- There are actually two Hamiltonian paths in this graph:
  - Path 1: Gives the string

    S = ATG

SBH: Hamiltonian Path Approach

- **Example**:

S = {ATG TGG TGC GTG GGC GCA GCG CGT}



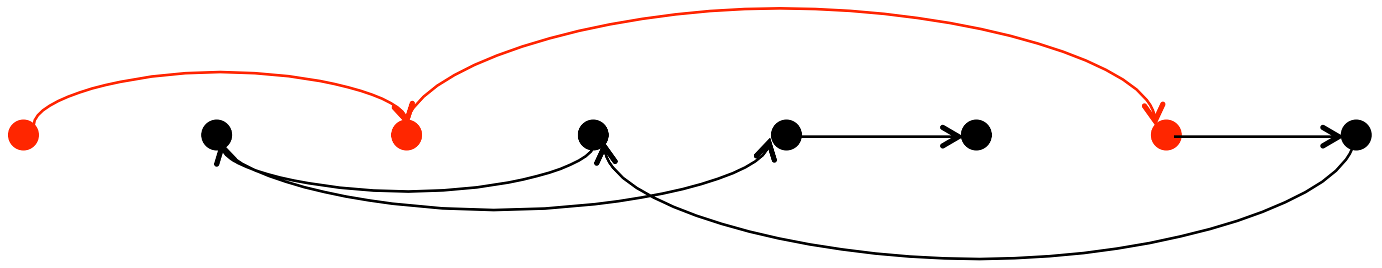
- There are actually two Hamiltonian paths in this graph:
  - Path 1: Gives the string

    S = ATGC

SBH: Hamiltonian Path Approach

- **Example**:

S = {ATG TGG TGC GTG GGC GCA GCG CGT}



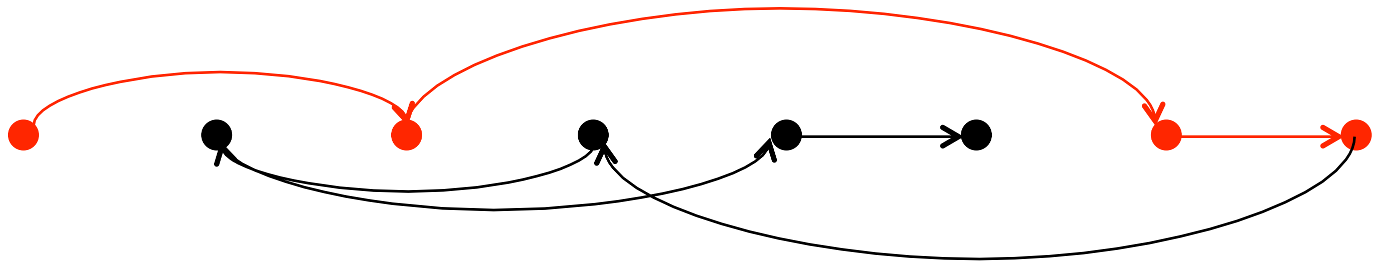
- There are actually two Hamiltonian paths in this graph:
  - Path 1: Gives the string

    S = ATGCG

SBH: Hamiltonian Path Approach

- **Example**:

S = {ATG TGG TGC GTG GGC GCA GCG CGT}



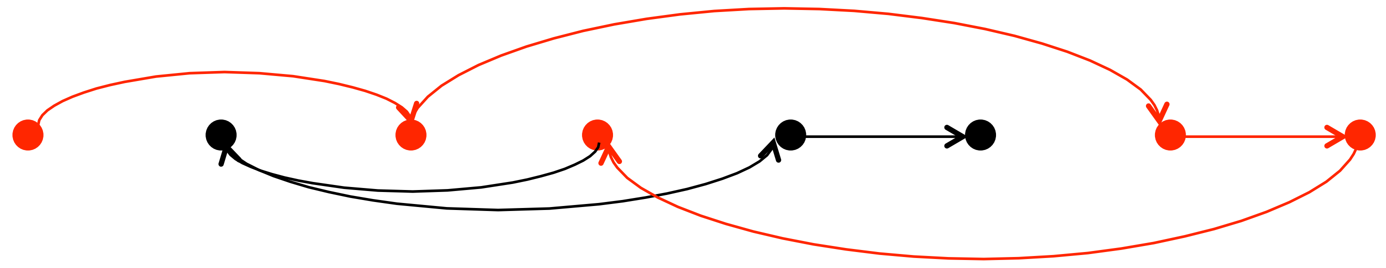
- There are actually two Hamiltonian paths in this graph:
  - Path 1: Gives the string

    S = ATGCGT

SBH: Hamiltonian Path Approach

- **Example**:

S = {ATG TGG TGC GTG GGC GCA GCG CGT}



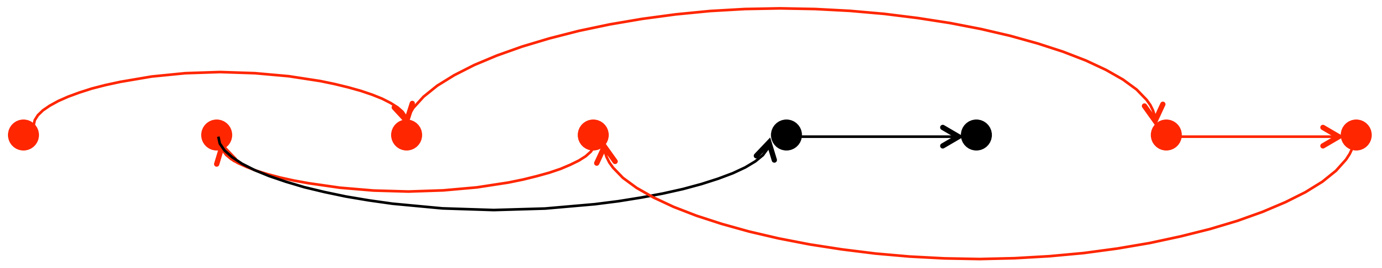
- There are actually two Hamiltonian paths in this graph:
  - Path 1: Gives the string

    S = ATGCGTG

SBH: Hamiltonian Path Approach

- **Example**:

S = {ATG TGG TGC GTG GGC GCA GCG CGT}



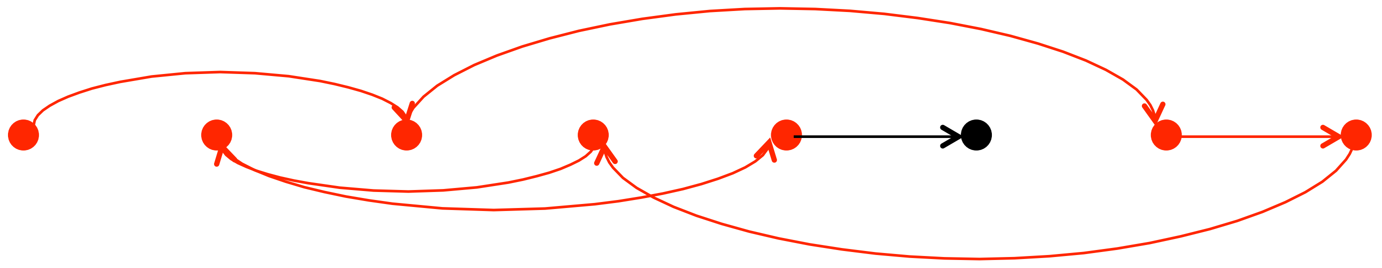
- There are actually two Hamiltonian paths in this graph:
  - Path 1: Gives the string

    S = ATGCGTGG

SBH: Hamiltonian Path Approach

- **Example**:

S = {ATG TGG TGC GTG GGC GCA GCG CGT}



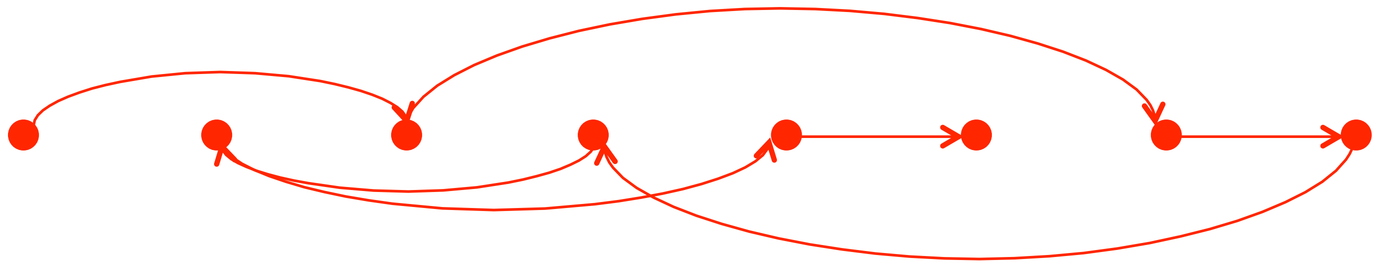
- There are actually two Hamiltonian paths in this graph:
  - Path 1: Gives the string

    S = ATGCGTGGC

SBH: Hamiltonian Path Approach

- **Example**:

S = {ATG TGG TGC GTG GGC GCA GCG CGT}



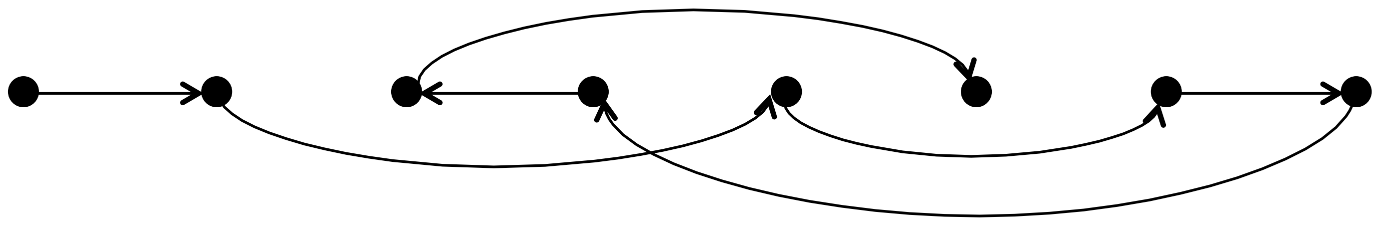
- There are actually two Hamiltonian paths in this graph:
  - Path 1: Gives the string

    S = ATGCGTGGCA

SBH: Hamiltonian Path Approach

- **Example**:

$S = \{$ATG TGG TGC GTG GGC GCA GCG CGT$\}$



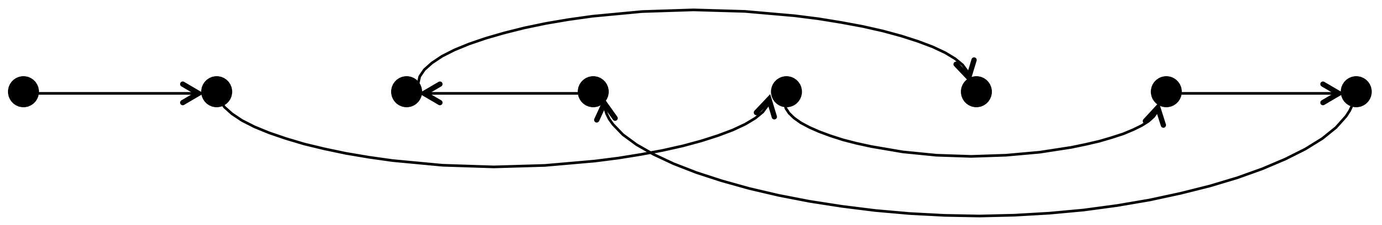
- There are actually two Hamiltonian paths in this graph:
  - Path 1: Gives the string

    S = ATGCGTGGCA
  - Path 2:

SBH: Hamiltonian Path Approach

- **Example**:

  S = {ATG TGG TGC GTG GGC GCA GCG CGT}



- There are actually two Hamiltonian paths in this graph:
  - Path 1: Gives the string

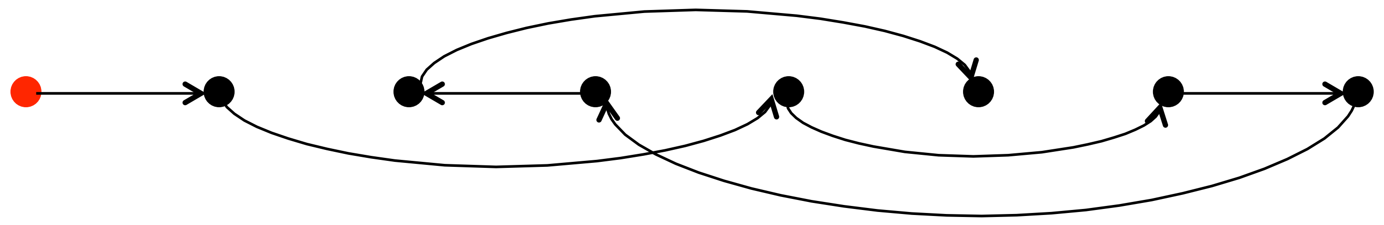
    S = ATGCGTGGCA
  - Path 2: Gives the string

    S =

SBH: Hamiltonian Path Approach

- **Example**:

  S = {ATG TGG TGC GTG GGC GCA GCG CGT}



- There are actually two Hamiltonian paths in this graph:
  - Path 1: Gives the string

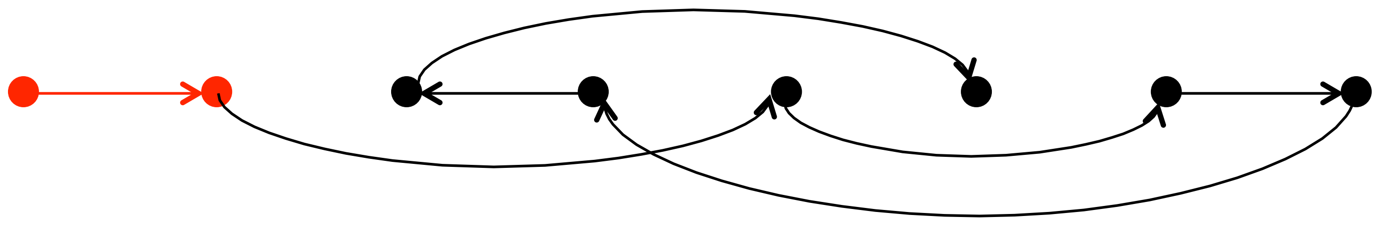
    S = ATGCGTGGCA
  - Path 2: Gives the string

    S = ATG

SBH: Hamiltonian Path Approach

- **Example**:

S = {ATG TGG TGC GTG GGC GCA GCG CGT}

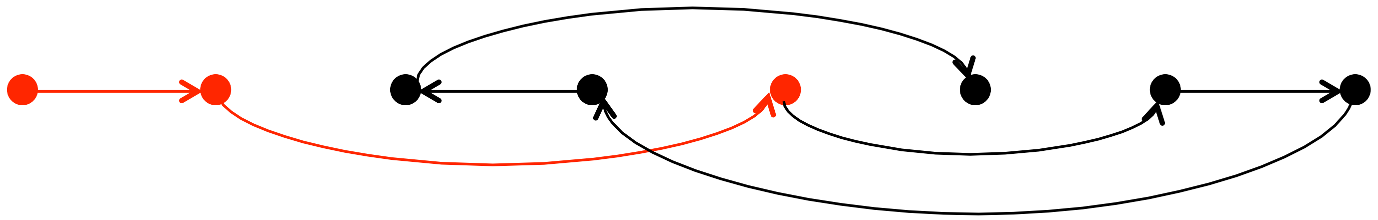


- There are actually two Hamiltonian paths in this graph:
  - Path 1: Gives the string
    S = ATGCGTGGCA
  - Path 2: Gives the string
    S = ATGG

SBH: Hamiltonian Path Approach

- **Example**:

S = {ATG TGG TGC GTG GGC GCA GCG CGT}



- There are actually two Hamiltonian paths in this graph:
  - Path 1: Gives the string

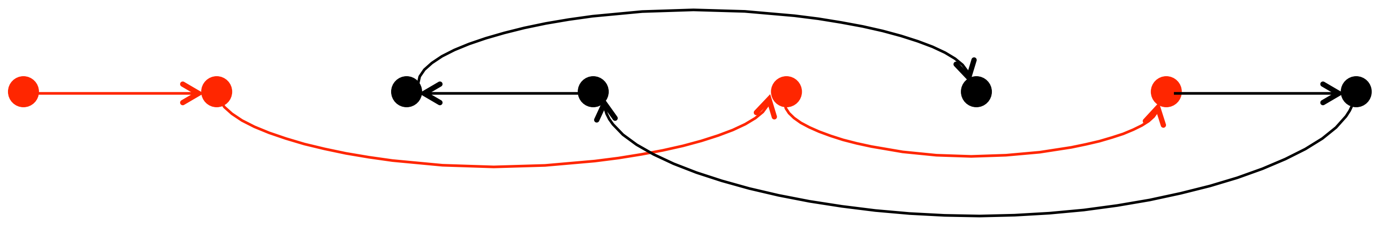
    S = ATGCGTGGCA
  - Path 2: Gives the string

    S = ATGGC

SBH: Hamiltonian Path Approach

- **Example**:

S = {ATG TGG TGC GTG GGC GCA GCG CGT}



- There are actually two Hamiltonian paths in this graph:
  - Path 1: Gives the string

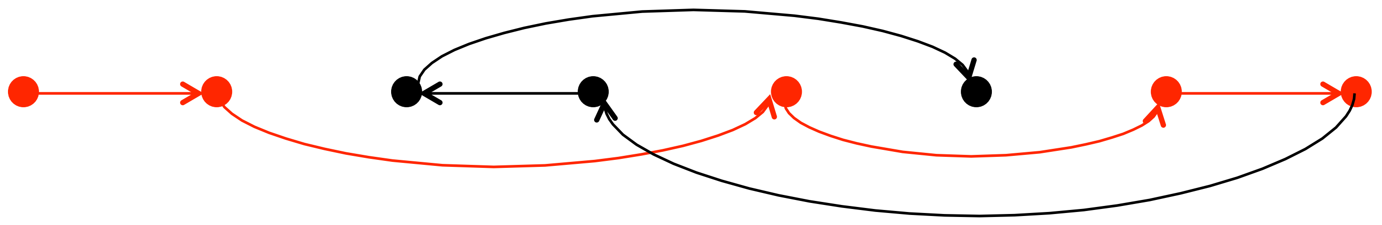
    S = ATGCGTGGCA
  - Path 2: Gives the string

    S = ATGGCG

SBH: Hamiltonian Path Approach

- **Example**:

S = {ATG TGG TGC GTG GGC GCA GCG CGT}



- There are actually two Hamiltonian paths in this graph:
  - Path 1: Gives the string

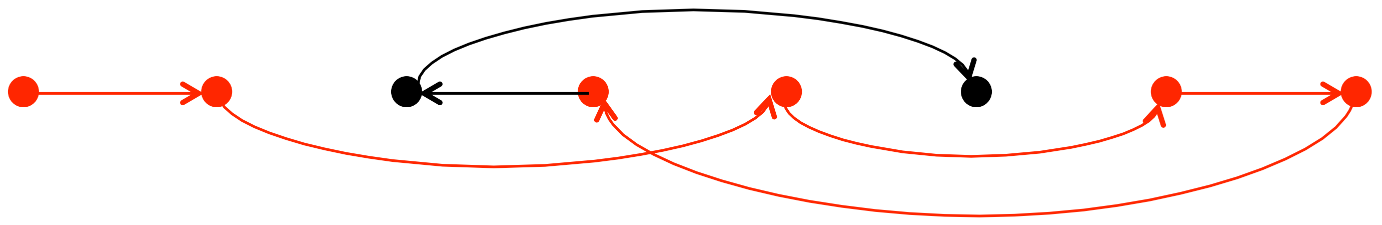
    S = ATGCGTGGCA
  - Path 2: Gives the string

    S = ATGGCGT

SBH: Hamiltonian Path Approach

- **Example**:

  S = {ATG TGG TGC GTG GGC GCA GCG CGT}



- There are actually two Hamiltonian paths in this graph:
  - Path 1: Gives the string

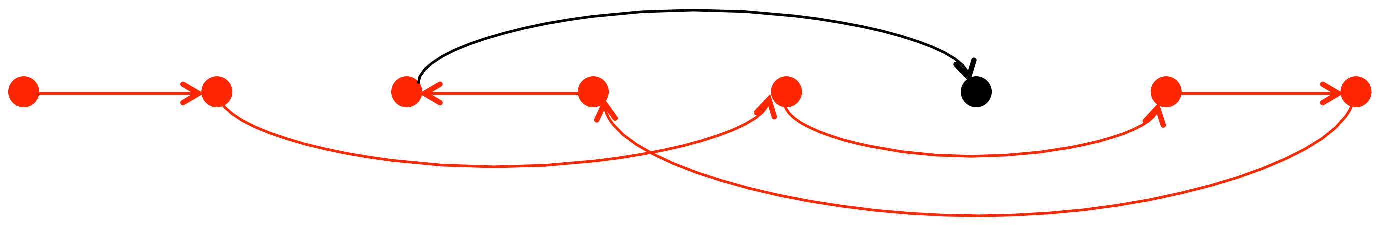
    S = ATGCGTGGCA
  - Path 2: Gives the string

    S = ATGGCGTG

SBH: Hamiltonian Path Approach

- **Example**:

S = {ATG TGG TGC GTG GGC GCA GCG CGT}



- There are actually two Hamiltonian paths in this graph:
  - Path 1: Gives the string

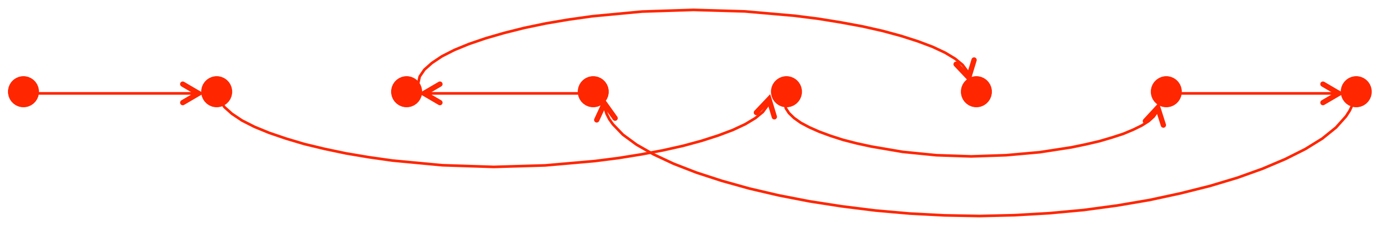
    S = ATGCGTGGCA
  - Path 2: Gives the string

    S = ATGGCGTGC

SBH: Hamiltonian Path Approach

- **Example**:

S = {ATG TGG TGC GTG GGC GCA GCG CGT}



- There are actually two Hamiltonian paths in this graph:
  - Path 1: Gives the string

    S = ATGCGTGGCA
  - Path 2: Gives the string

    S = ATGGCGTGCA

SBH: A Lost Cause?

- At this point, we should be concerned about using a Hamiltonian path to solve SBH.
- After all, recall that SSP was an NP-Complete problem, and we have seen that an instance of SBH is an instance of SSP.
- However, note that SBH is actually a specific case of SSP, so there is still hope for an efficient algorithm for SBH:
  - We are considering a spectrum of only *l*-mers, and not strings of any other length.
  - Also, we only are connecting two *l*-mers with an edge if and only if the overlap between them is $l - 1$, whereas before we connected *l*-mers if there was any overlap at all.

- **Note**: SBH is not NP-Complete since SBH reduces to SSP, but not vice-versa.

SBH: Eulerian Path Approach

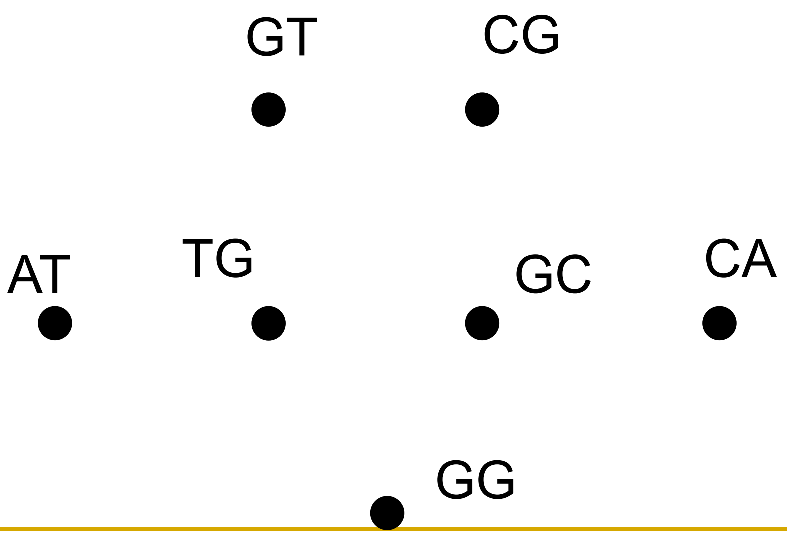
- So instead, let us consider a completely *different* graph G:
    - Vertices = the set of (l – 1)-mers which are substrings of some l-mer from our set *S*.
    - *v* is connected to *w* with a *directed* edge if the final l – 2 elements of *v* agree with the first l – 2 elements of *w*, and the *union* of *v* and *w* is in *S*.

- **Example**: *S* = {ATG, TGG, TGC, GTG, GGC, GCA, GCG, CGT}.

SBH: Eulerian Path Approach

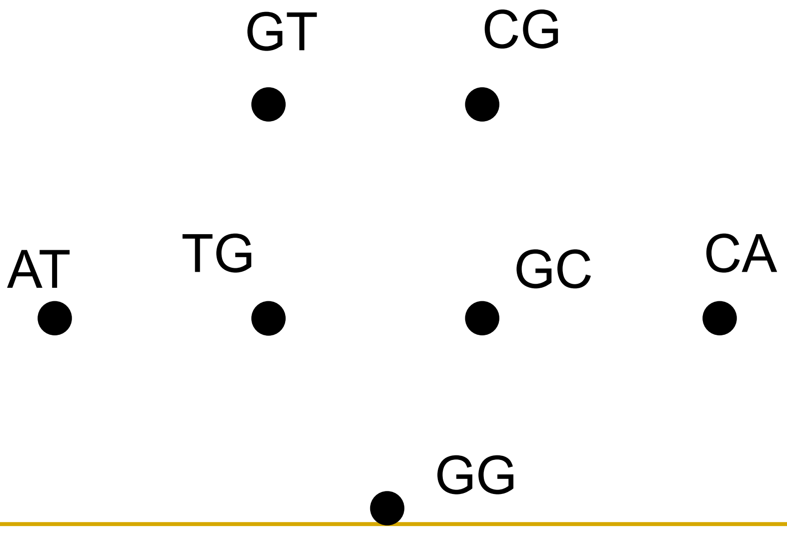
- So instead, let us consider a completely *different* graph G:
    - Vertices = the set of (*l* – 1)-mers which are substrings of some *l*-mer from our set *S*.
    - *v* is connected to *w* with a *directed* edge if the final *l* – 2 elements of *v* agree with the first *l* – 2 elements of *w*, and the *union* of *v* and *w* is in *S*.

- **Example**: *S* = {ATG, TGG, TGC, GTG, GGC, GCA, GCG, CGT}.
    - V = {AT}.



SBH: Eulerian Path Approach

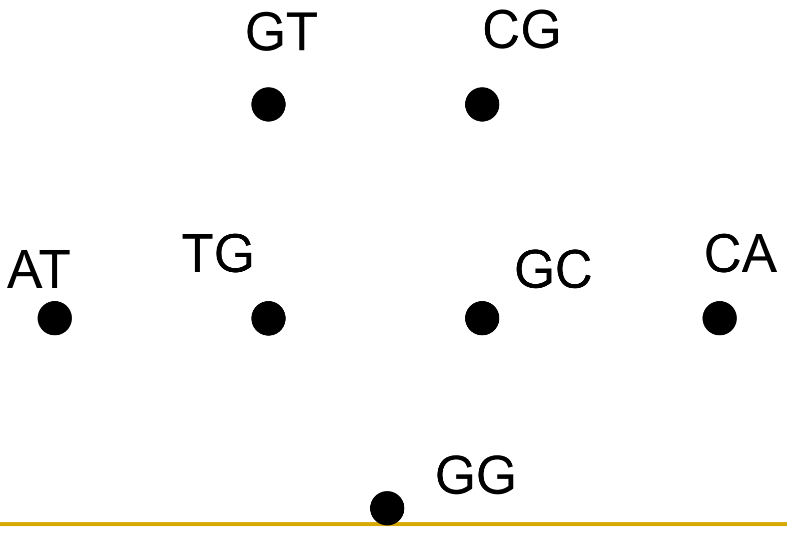
- So instead, let us consider a completely *different* graph G:
  - Vertices = the set of (l – 1)-mers which are substrings of some l-mer from our set *S*.
  - *v* is connected to *w* with a *directed* edge if the final l – 2 elements of *v* agree with the first l – 2 elements of *w*, and the *union* of *v* and *w* is in *S*.
- **Example**: *S* = {ATG, TGG, TGC, GTG, GGC, GCA, GCG, CGT}.
  - V = {AT, TG}.



SBH: Eulerian Path Approach

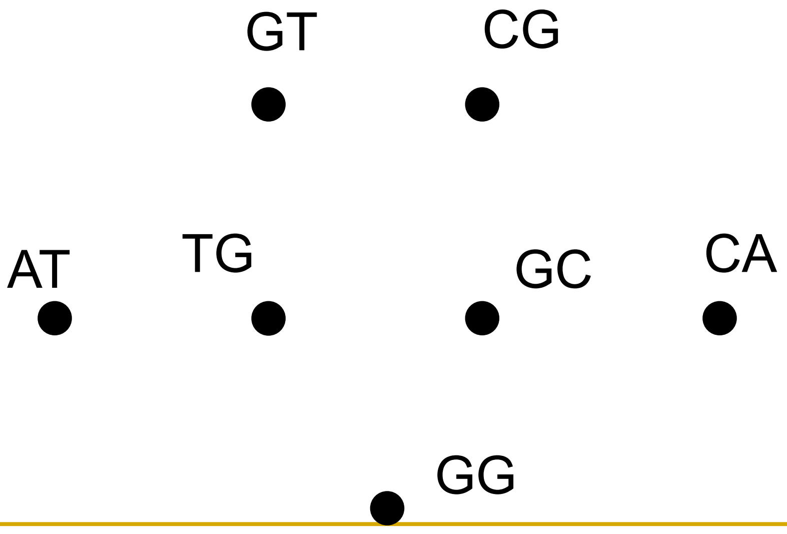
- So instead, let us consider a completely *different* graph G:
  - Vertices = the set of (l – 1)-mers which are substrings of some l-mer from our set *S*.
  - *v* is connected to *w* with a *directed* edge if the final l – 2 elements of *v* agree with the first l – 2 elements of *w*, and the *union* of *v* and *w* is in *S*.

- **Example**: *S* = {ATG, TGG, TGC, GTG, GGC, GCA, GCG, CGT}.
  - V = {AT, TG, GG}.



SBH: Eulerian Path Approach

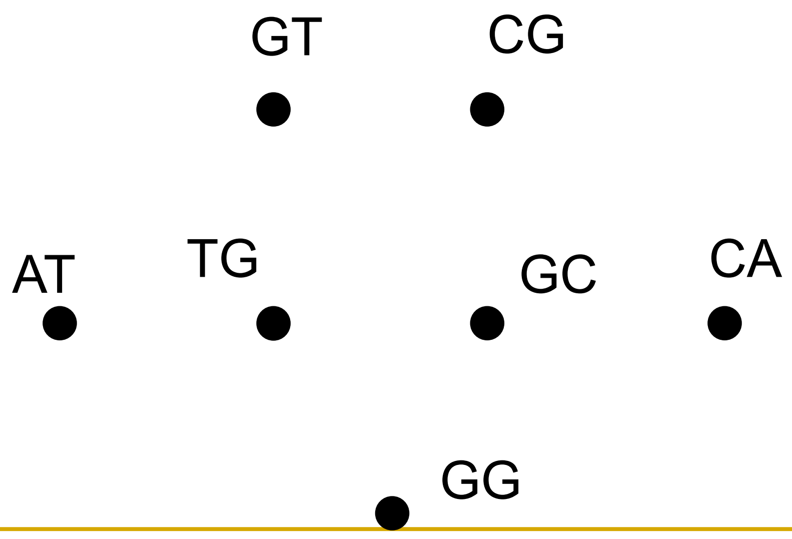
- So instead, let us consider a completely *different* graph G:
  - Vertices = the set of (l – 1)-mers which are substrings of some l-mer from our set *S*.
  - *v* is connected to *w* with a *directed* edge if the final l – 2 elements of *v* agree with the first l – 2 elements of *w*, and the *union* of *v* and *w* is in *S*.
- **Example**: *S* = {ATG, TGG, TGC, GTG, GGC, GCA, GCG, CGT}.
  - V = {AT, TG, GG, GC}.



SBH: Eulerian Path Approach

- So instead, let us consider a completely *different* graph G:
  - Vertices = the set of (*l* – 1)-mers which are substrings of some *l*-mer from our set *S*.
  - *v* is connected to *w* with a *directed* edge if the final *l* – 2 elements of *v* agree with the first *l* – 2 elements of *w*, and the *union* of *v* and *w* is in *S*.

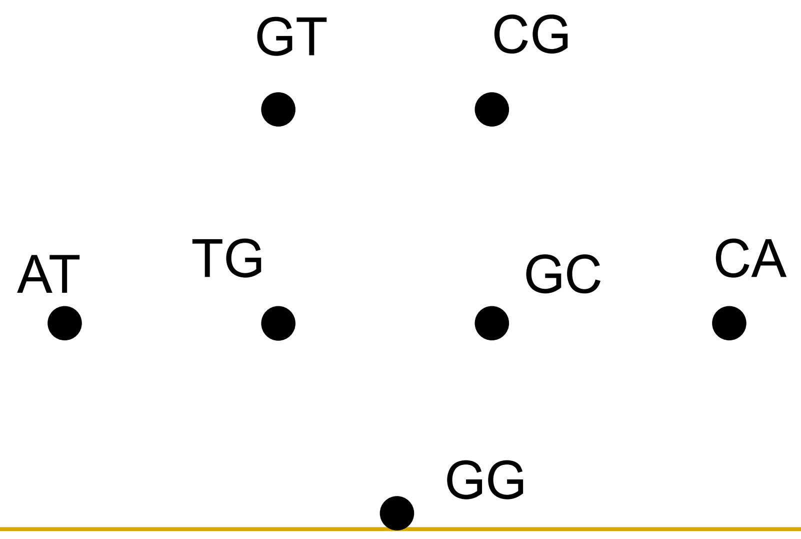
- **Example**: *S* = {ATG, TGG, TGC, GTG, GGC, GCA, GCG, CGT}.
  - V = {AT, TG, GG, GC, GT}.



SBH: Eulerian Path Approach

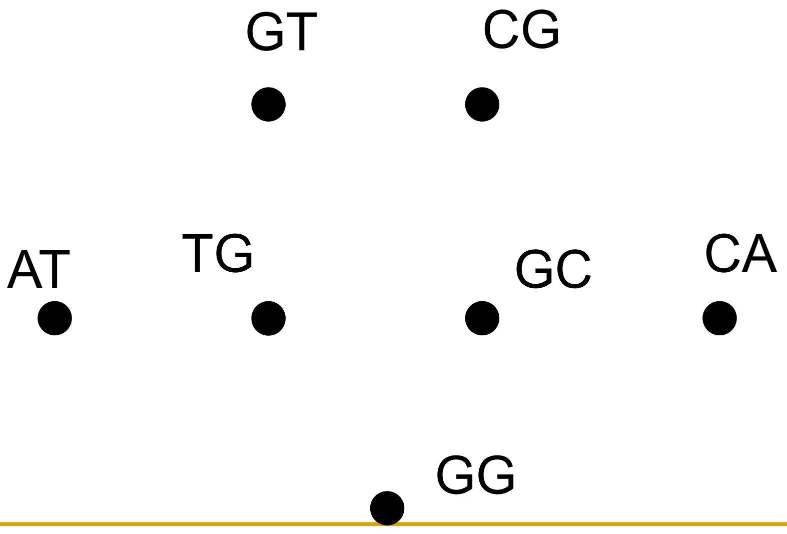
- So instead, let us consider a completely *different* graph G:
    - Vertices = the set of (l – 1)-mers which are substrings of some l-mer from our set S.
    - v is connected to w with a *directed* edge if the final l – 2 elements of v agree with the first l – 2 elements of w, and the *union* of v and w is in S.

- **Example**: S = {ATG, TGG, TGC, GTG, GGC, GCA, GCG, CGT}.
    - V = {AT, TG, GG, GC, GT, CA}.



SBH: Eulerian Path Approach

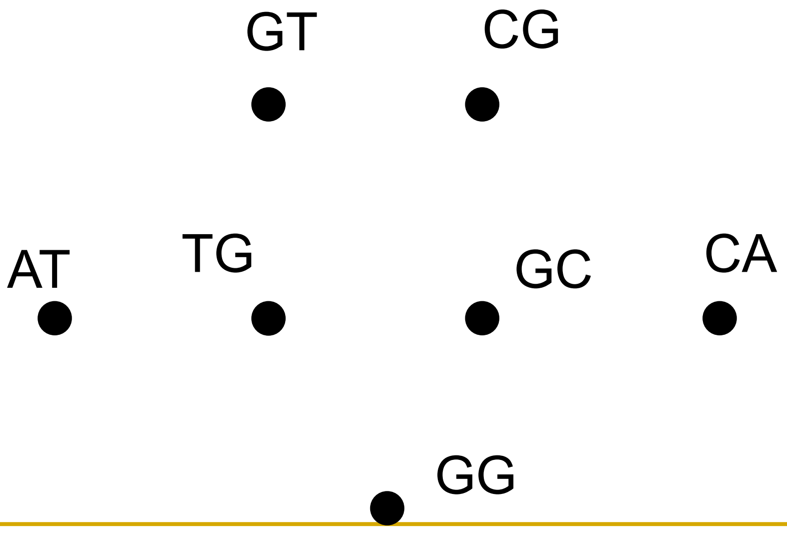
- So instead, let us consider a completely *different* graph G:
    - Vertices = the set of (*l* – 1)-mers which are substrings of some *l*-mer from our set *S*.
    - *v* is connected to *w* with a *directed* edge if the final *l* – 2 elements of *v* agree with the first *l* – 2 elements of *w*, and the *union* of *v* and *w* is in *S*.
- **Example**: *S* = {ATG, TGG, TGC, GTG, GGC, GCA, GCG, CGT}.
    - V = {AT, TG, GG, GC, GT, CA, CG}.



SBH: Eulerian Path Approach

- So instead, let us consider a completely *different* graph G:
  - Vertices = the set of (*l* – 1)-mers which are substrings of some *l*-mer from our set *S*.
  - *v* is connected to *w* with a *directed* edge if the final *l* – 2 elements of *v* agree with the first *l* – 2 elements of *w*, and the *union* of *v* and *w* is in *S*.

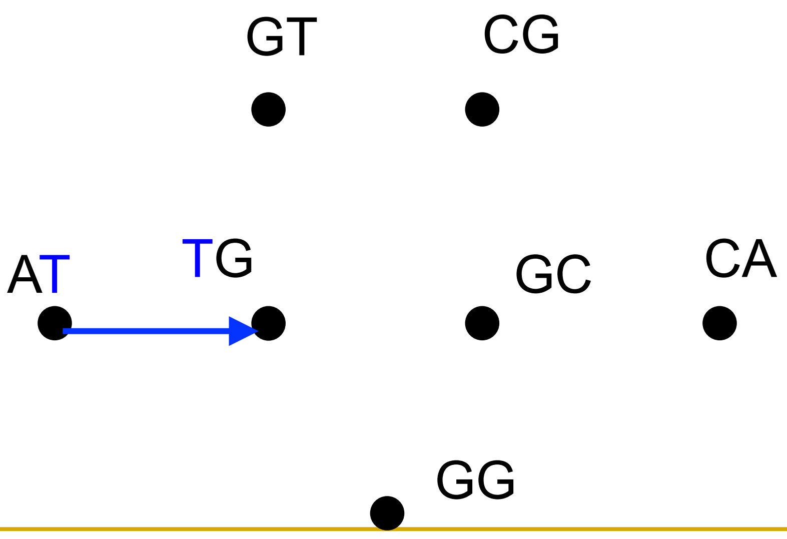
- **Example**: *S* = {ATG, TGG, TGC, GTG, GGC, GCA, GCG, CGT}.
  - V = {AT, TG, GG, GC, GT, CA, CG}.
  - E = shown at right.



SBH: Eulerian Path Approach

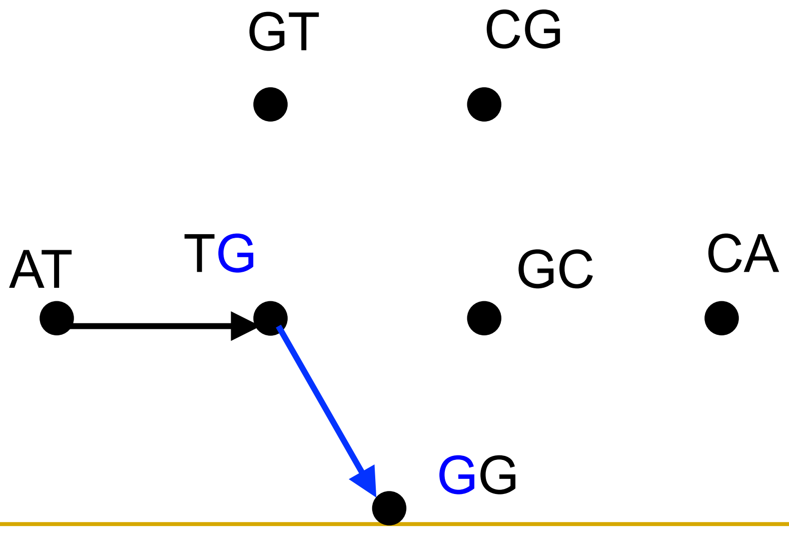
- So instead, let us consider a completely *different* graph G:
  - Vertices = the set of (*l* – 1)-mers which are substrings of some *l*-mer from our set *S*.
  - *v* is connected to *w* with a *directed* edge if the final *l* – 2 elements of *v* agree with the first *l* – 2 elements of *w*, and the *union* of *v* and *w* is in *S*.

- **Example**: *S* = {ATG, TGG, TGC, GTG, GGC, GCA, GCG, CGT}.
  - V = {AT, TG, GG, GC, GT, CA, CG}.
  - E = shown at right.



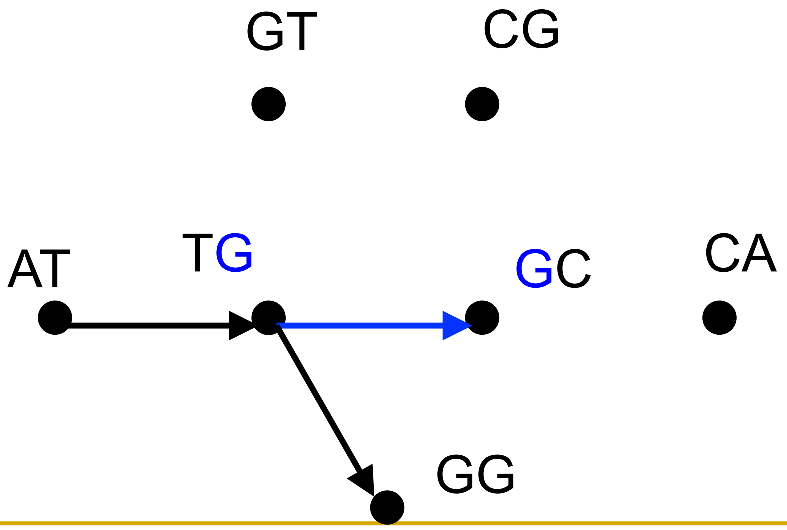
SBH: Eulerian Path Approach

- So instead, let us consider a completely *different* graph G:
  - Vertices = the set of (*l* – 1)-mers which are substrings of some *l*-mer from our set *S*.
  - *v* is connected to *w* with a *directed* edge if the final *l* – 2 elements of *v* agree with the first *l* – 2 elements of *w*, and the *union* of *v* and *w* is in *S*.
- **Example**: *S* = {ATG, TGG, TGC, GTG, GGC, GCA, GCG, CGT}.
  - V = {AT, TG, GG, GC, GT, CA, CG}.
  - E = shown at right.



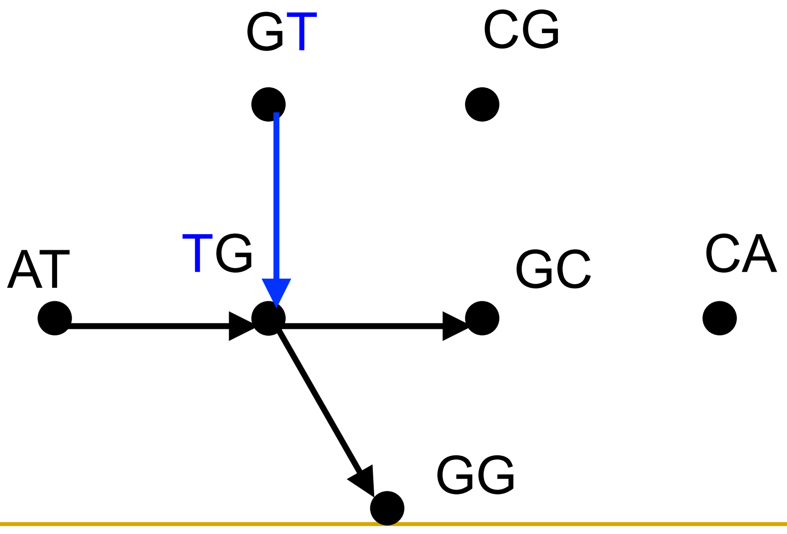
SBH: Eulerian Path Approach

- So instead, let us consider a completely *different* graph G:
  - Vertices = the set of (*l* – 1)-mers which are substrings of some *l*-mer from our set *S*.
  - *v* is connected to *w* with a *directed* edge if the final *l* – 2 elements of *v* agree with the first *l* – 2 elements of *w*, and the *union* of *v* and *w* is in *S*.
- **Example**: *S* = {ATG, TGG, TGC, GTG, GGC, GCA, GCG, CGT}.
  - V = {AT, TG, GG, GC, GT, CA, CG}.
  - E = shown at right.



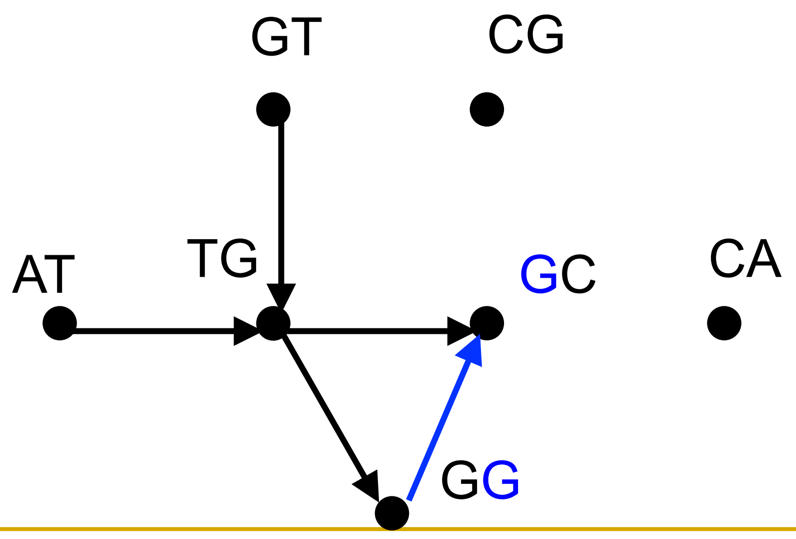
SBH: Eulerian Path Approach

- So instead, let us consider a completely *different* graph G:
  - Vertices = the set of (l – 1)-mers which are substrings of some l-mer from our set S.
  - v is connected to w with a *directed* edge if the final l – 2 elements of v agree with the first l – 2 elements of w, and the *union* of v and w is in S.
- **Example**: S = {ATG, TGG, TGC, GTG, GGC, GCA, GCG, CGT}.
  - V = {AT, TG, GG, GC, GT, CA, CG}.
  - E = shown at right.



SBH: Eulerian Path Approach

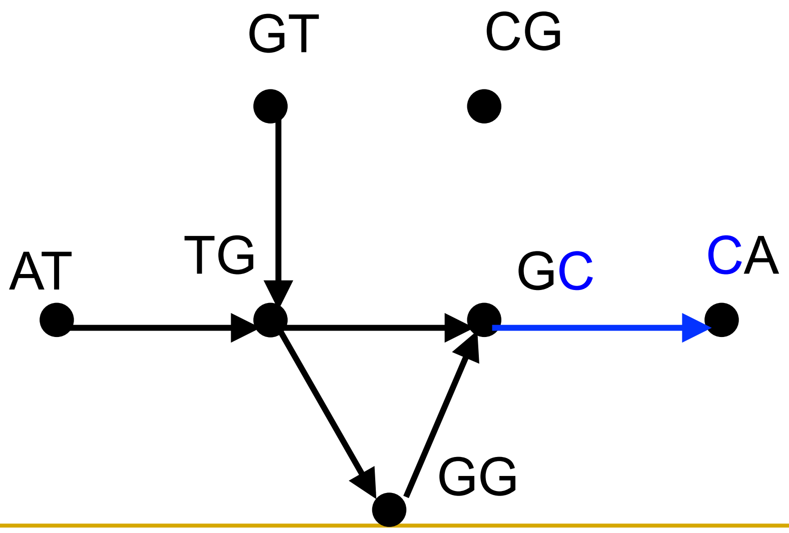
- So instead, let us consider a completely *different* graph G:
  - Vertices = the set of (l – 1)-mers which are substrings of some l-mer from our set S.
  - v is connected to w with a *directed* edge if the final l – 2 elements of v agree with the first l – 2 elements of w, and the *union* of v and w is in S.
- **Example**: S = {ATG, TGG, TGC, GTG, GGC, GCA, GCG, CGT}.
  - V = {AT, TG, GG, GC, GT, CA, CG}.
  - E = shown at right.



SBH: Eulerian Path Approach

- So instead, let us consider a completely *different* graph G:
  - Vertices = the set of (l – 1)-mers which are substrings of some l-mer from our set *S*.
  - *v* is connected to *w* with a *directed* edge if the final l – 2 elements of *v* agree with the first l – 2 elements of *w*, and the *union* of *v* and *w* is in *S*.

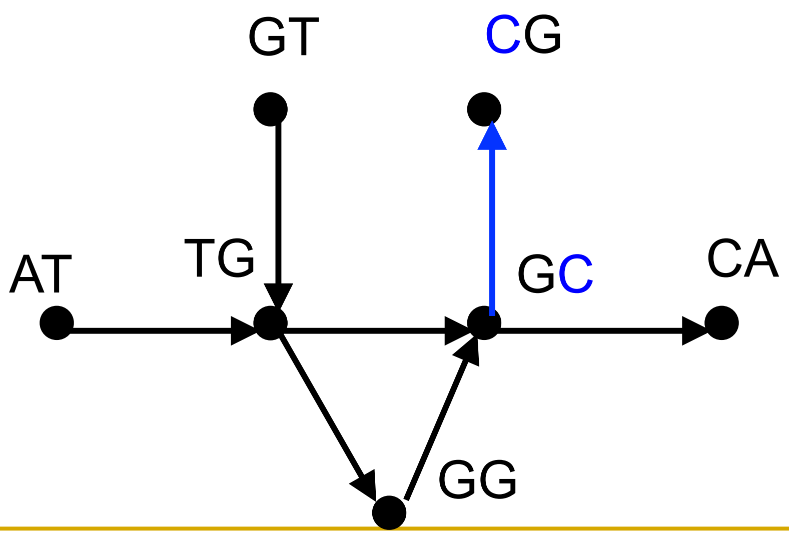
- **Example**: *S* = {ATG, TGG, TGC, GTG, GGC, GCA, GCG, CGT}.
  - V = {AT, TG, GG, GC, GT, CA, CG}.
  - E = shown at right.



SBH: Eulerian Path Approach

- So instead, let us consider a completely *different* graph G:
  - Vertices = the set of (l – 1)-mers which are substrings of some l-mer from our set S.
  - v is connected to w with a *directed* edge if the final l – 2 elements of v agree with the first l – 2 elements of w, and the *union* of v and w is in S.

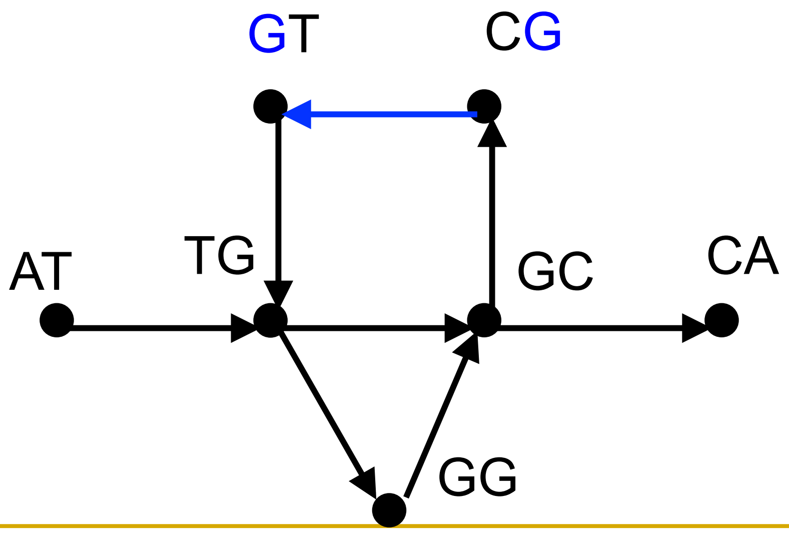
- **Example**: S = {ATG, TGG, TGC, GTG, GGC, GCA, GCG, CGT}.
  - V = {AT, TG, GG, GC, GT, CA, CG}.
  - E = shown at right.



SBH: Eulerian Path Approach

- So instead, let us consider a completely *different* graph G:
  - Vertices = the set of (*l* – 1)-mers which are substrings of some *l*-mer from our set *S*.
  - *v* is connected to *w* with a *directed* edge if the final *l* – 2 elements of *v* agree with the first *l* – 2 elements of *w*, and the *union* of *v* and *w* is in *S*.

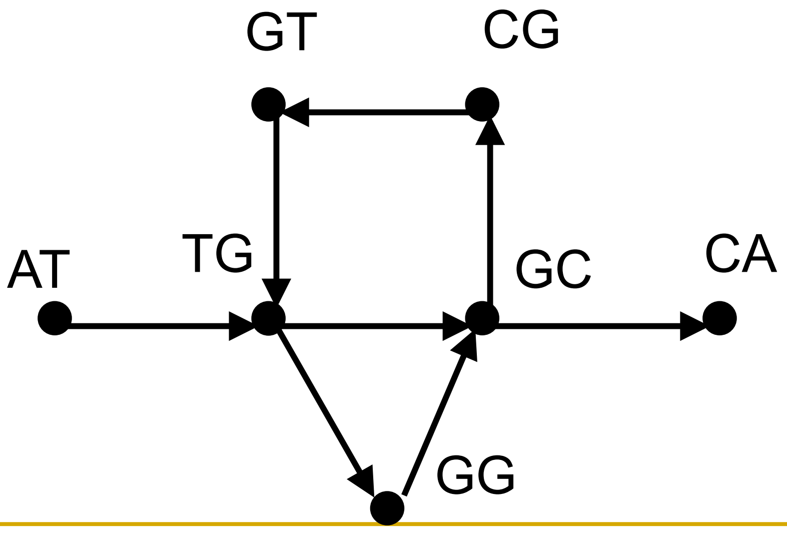
- **Example**: *S* = {ATG, TGG, TGC, GTG, GGC, GCA, GCG, CGT}.
  - V = {AT, TG, GG, GC, GT, CA, CG}.
  - E = shown at right.



SBH: Eulerian Path Approach

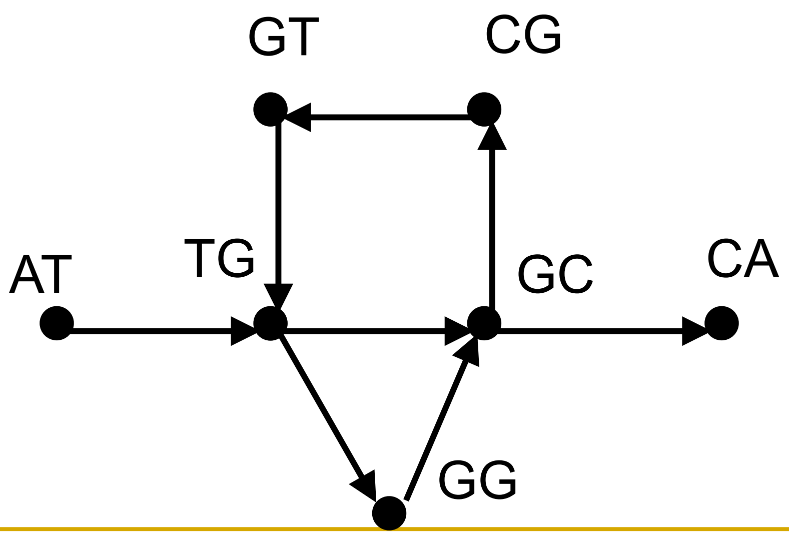
- So instead, let us consider a completely *different* graph G:
  - Vertices = the set of (l – 1)-mers which are substrings of some l-mer from our set S.
  - v is connected to w with a *directed* edge if the final l – 2 elements of v agree with the first l – 2 elements of w, and the *union* of v and w is in S.

- **Example**: S = {ATG, TGG, TGC, GTG, GGC, GCA, GCG, CGT}.
  - V = {AT, TG, GG, GC, GT, CA, CG}.
  - E = shown at right.



SBH: Eulerian Path Approach

- **Key Point**: A sequence reconstruction will actually correspond to an *Eulerian* path in this graph.
- Recall that an Eulerian path is "easy" to find (one can always be found in linear time)…so we have found a simple solution to SBH!
- In our example, two solutions:

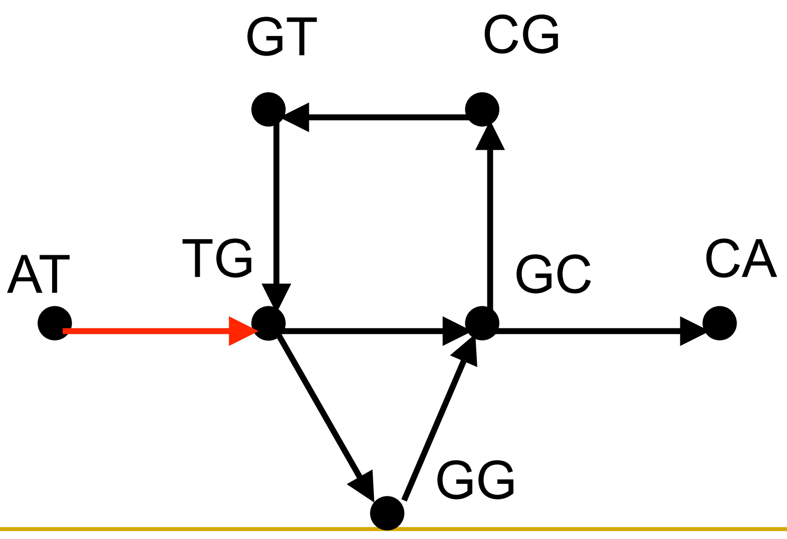


SBH: Eulerian Path Approach

- **Key Point**: A sequence reconstruction will actually correspond to an *Eulerian* path in this graph.

- Recall that an Eulerian path is "easy" to find (one can always be found in linear time)…so we have found a simple solution to SBH!

- In our example, two solutions:
  1. ATG

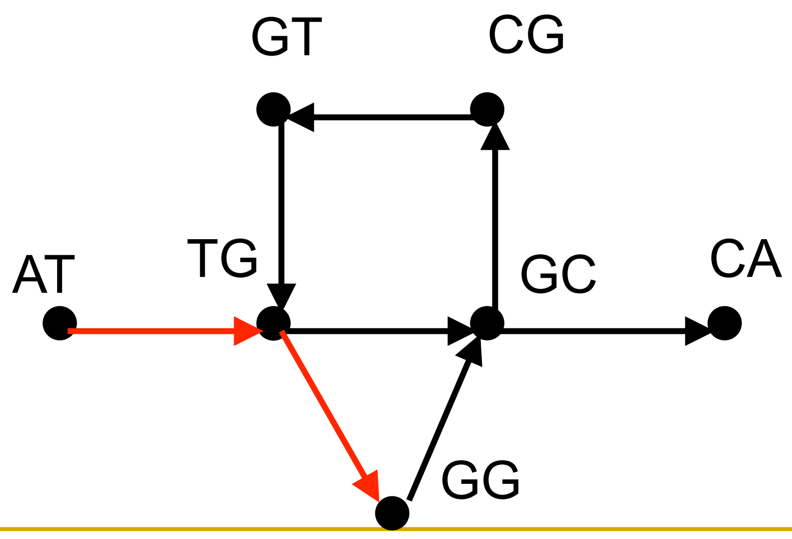


SBH: Eulerian Path Approach

- **Key Point**: A sequence reconstruction will actually correspond to an *Eulerian* path in this graph.

- Recall that an Eulerian path is "easy" to find (one can always be found in linear time)…so we have found a simple solution to SBH!

- In our example, two solutions:
  1. ATGG

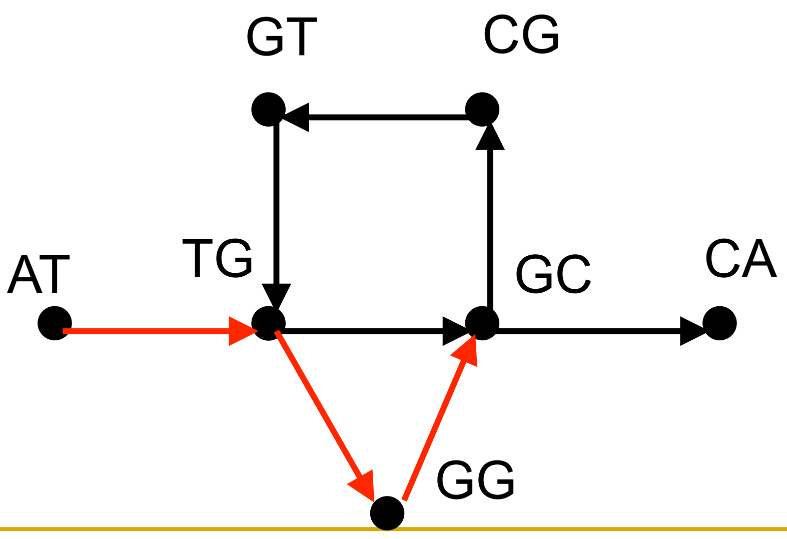


SBH: Eulerian Path Approach

- **Key Point**: A sequence reconstruction will actually correspond to an *Eulerian* path in this graph.

- Recall that an Eulerian path is "easy" to find (one can always be found in linear time)…so we have found a simple solution to SBH!

- In our example, two solutions:
  1. ATGGC

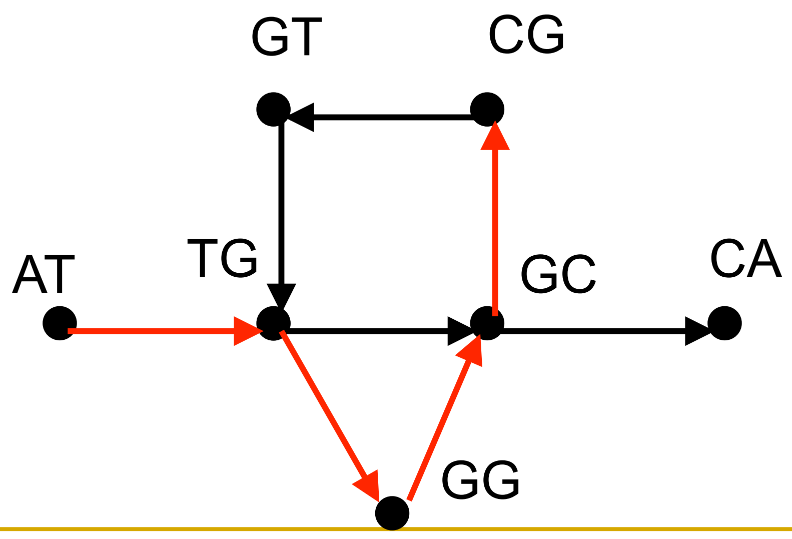


SBH: Eulerian Path Approach

- **Key Point**: A sequence reconstruction will actually correspond to an *Eulerian* path in this graph.

- Recall that an Eulerian path is “easy” to find (one can always be found in linear time)…so we have found a simple solution to SBH!

- In our example, two solutions:
  1. ATGGCG

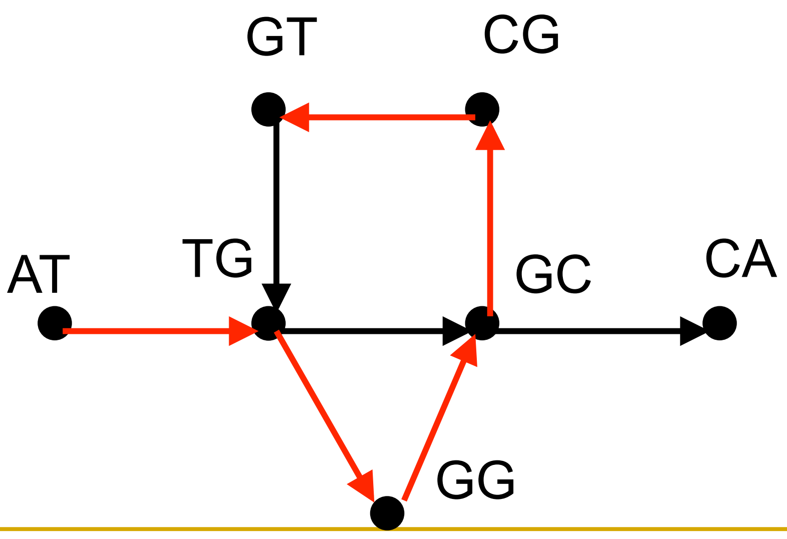


SBH: Eulerian Path Approach

- **Key Point**: A sequence reconstruction will actually correspond to an *Eulerian* path in this graph.

- Recall that an Eulerian path is "easy" to find (one can always be found in linear time)…so we have found a simple solution to SBH!

- In our example, two solutions:
  1. ATGGCGT

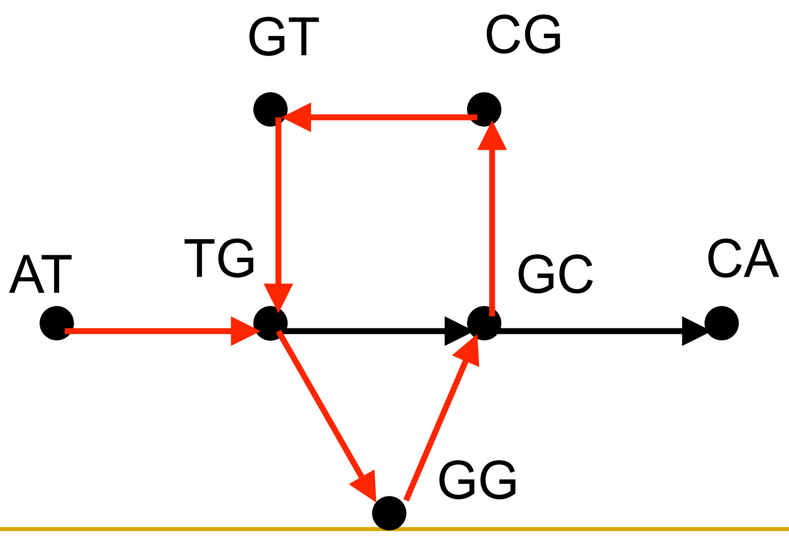


SBH: Eulerian Path Approach

- **Key Point**: A sequence reconstruction will actually correspond to an *Eulerian* path in this graph.

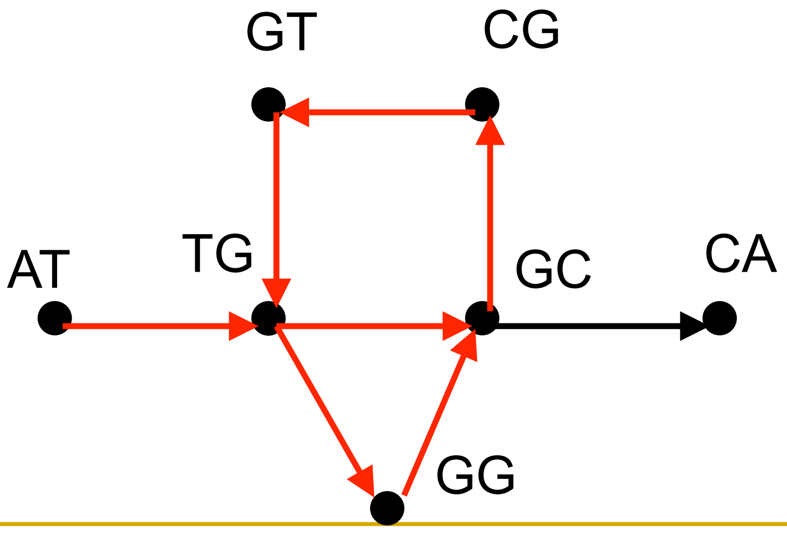
- Recall that an Eulerian path is "easy" to find (one can always be found in linear time)…so we have found a simple solution to SBH!

- In our example, two solutions:
  1. ATGGCGTG



SBH: Eulerian Path Approach

- **Key Point**: A sequence reconstruction will actually correspond to an *Eulerian* path in this graph.
- Recall that an Eulerian path is "easy" to find (one can always be found in linear time)…so we have found a simple solution to SBH!
- In our example, two solutions:
  1. ATGGCGTGC

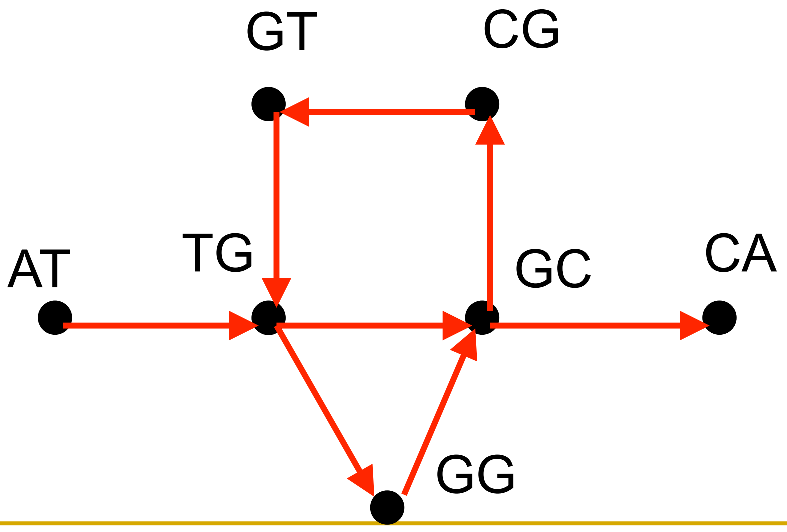


SBH: Eulerian Path Approach

- **Key Point**: A sequence reconstruction will actually correspond to an *Eulerian* path in this graph.

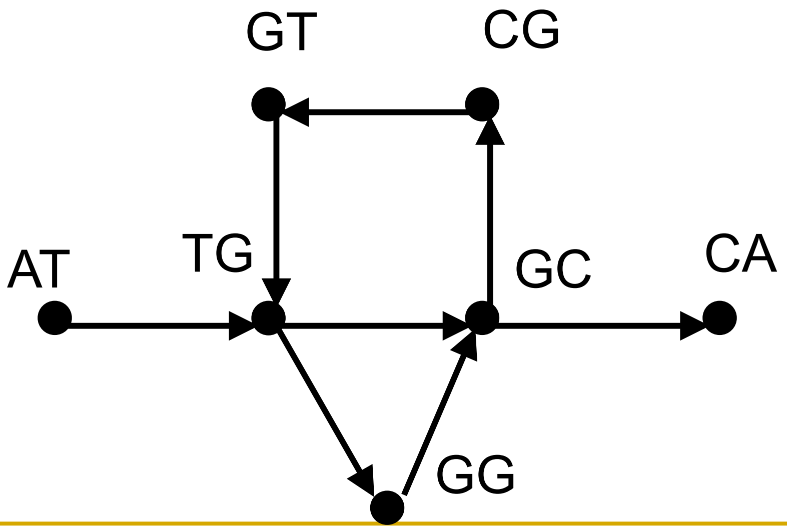
- Recall that an Eulerian path is “easy” to find (one can always be found in linear time)…so we have found a simple solution to SBH!

- In our example, two solutions:
  1. ATGGCGTGCA



SBH: Eulerian Path Approach

- **Key Point**: A sequence reconstruction will actually correspond to an *Eulerian* path in this graph.
- Recall that an Eulerian path is “easy” to find (one can always be found in linear time)…so we have found a simple solution to SBH!
- In our example, two solutions:
  1. ATGGCGTGCA

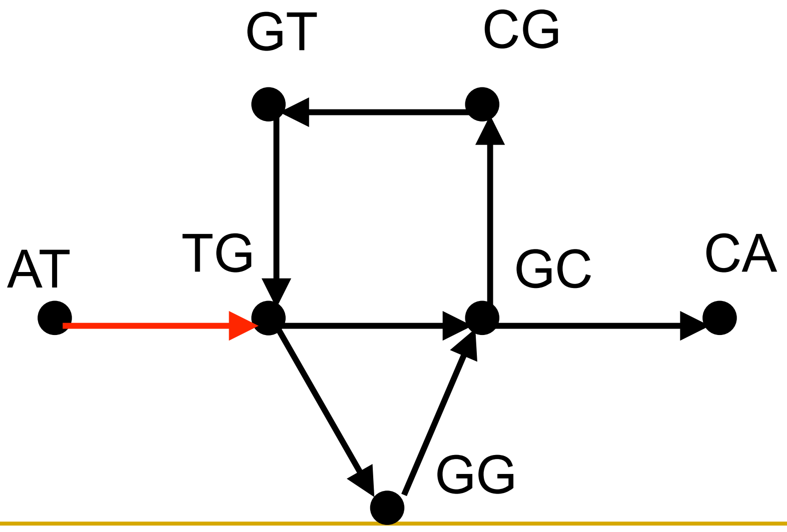


SBH: Eulerian Path Approach

- **Key Point**: A sequence reconstruction will actually correspond to an *Eulerian* path in this graph.

- Recall that an Eulerian path is “easy” to find (one can always be found in linear time)…so we have found a simple solution to SBH!

- In our example, two solutions:
  1. ATGGCGTGCA
  2. ATG

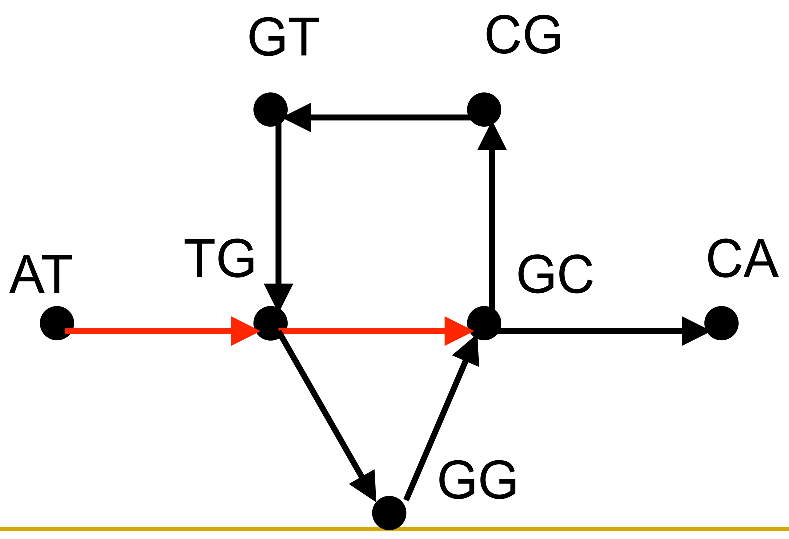


SBH: Eulerian Path Approach

- **Key Point**: A sequence reconstruction will actually correspond to an *Eulerian* path in this graph.

- Recall that an Eulerian path is "easy" to find (one can always be found in linear time)…so we have found a simple solution to SBH!

- In our example, two solutions:
  1. ATGGCGTGCA
  2. ATGC

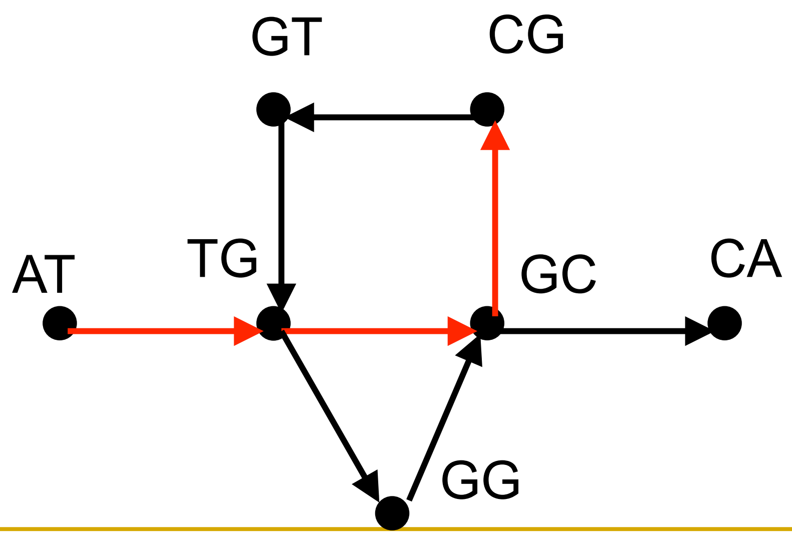


SBH: Eulerian Path Approach

- **Key Point**: A sequence reconstruction will actually correspond to an *Eulerian* path in this graph.

- Recall that an Eulerian path is "easy" to find (one can always be found in linear time)…so we have found a simple solution to SBH!

- In our example, two solutions:
  1. ATGGCGTGCA
  2. ATGCG

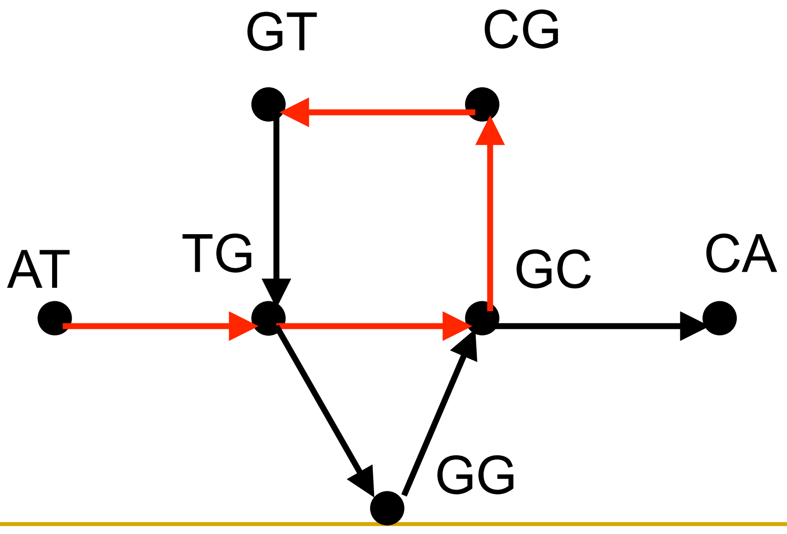


SBH: Eulerian Path Approach

- **Key Point**: A sequence reconstruction will actually correspond to an *Eulerian* path in this graph.

- Recall that an Eulerian path is "easy" to find (one can always be found in linear time)…so we have found a simple solution to SBH!

- In our example, two solutions:
  1. ATGGCGTGCA
  2. ATGCGT

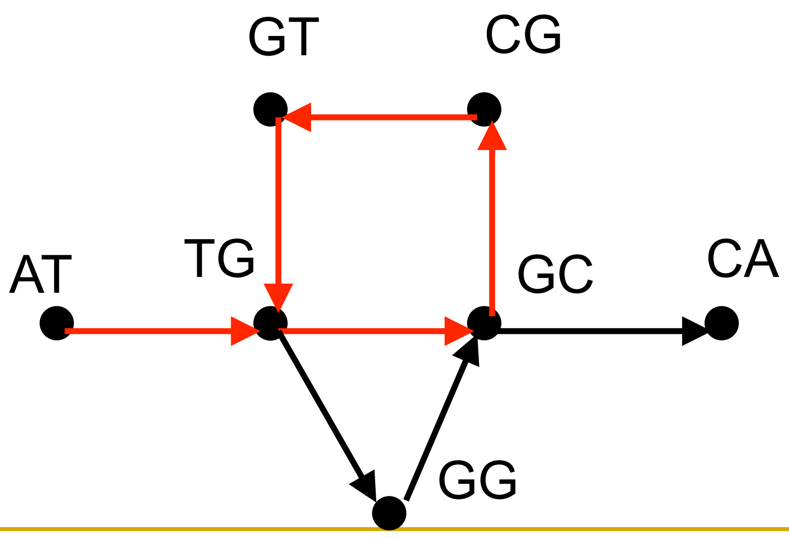


SBH: Eulerian Path Approach

- **Key Point**: A sequence reconstruction will actually correspond to an *Eulerian* path in this graph.

- Recall that an Eulerian path is "easy" to find (one can always be found in linear time)…so we have found a simple solution to SBH!

- In our example, two solutions:
  1. ATGGCGTGCA
  2. ATGCGTG

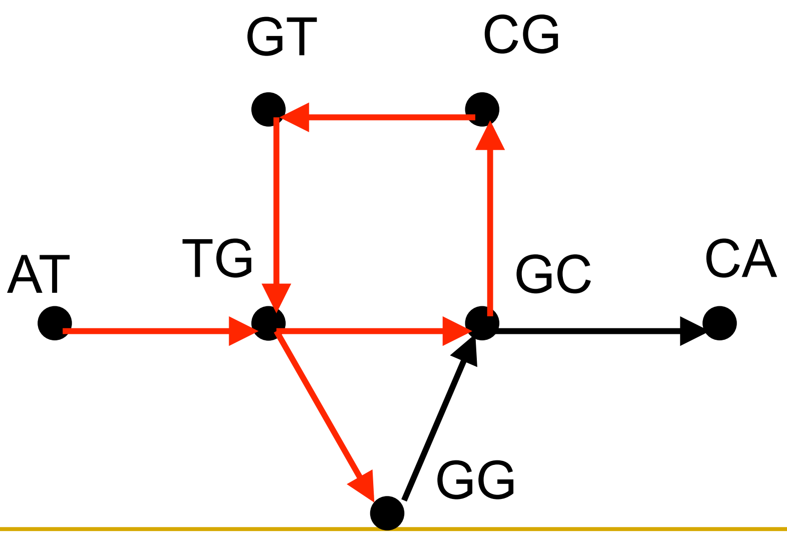


SBH: Eulerian Path Approach

- **Key Point**: A sequence reconstruction will actually correspond to an *Eulerian* path in this graph.

- Recall that an Eulerian path is “easy” to find (one can always be found in linear time)…so we have found a simple solution to SBH!

- In our example, two solutions:
  1. ATGGCGTGCA
  2. ATGCGTGG

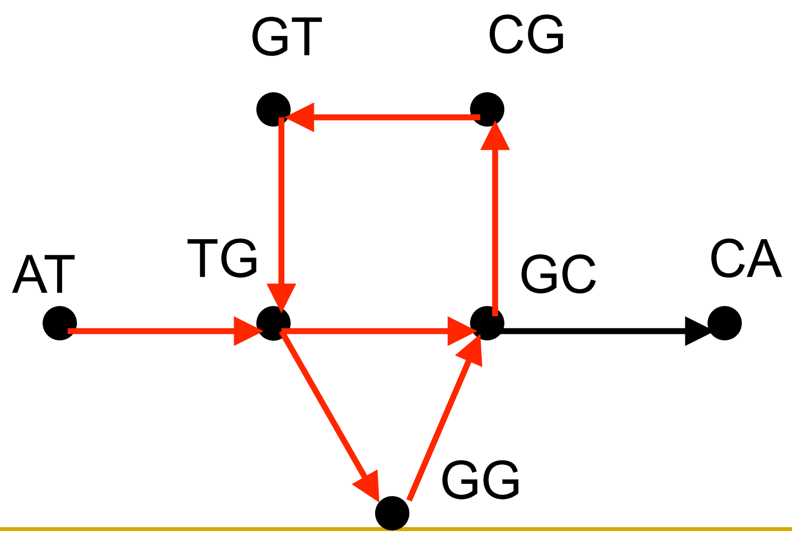


SBH: Eulerian Path Approach

- **Key Point**: A sequence reconstruction will actually correspond to an *Eulerian* path in this graph.

- Recall that an Eulerian path is "easy" to find (one can always be found in linear time)…so we have found a simple solution to SBH!

- In our example, two solutions:
  1. ATGGCGTGCA
  2. ATGCGTGGC

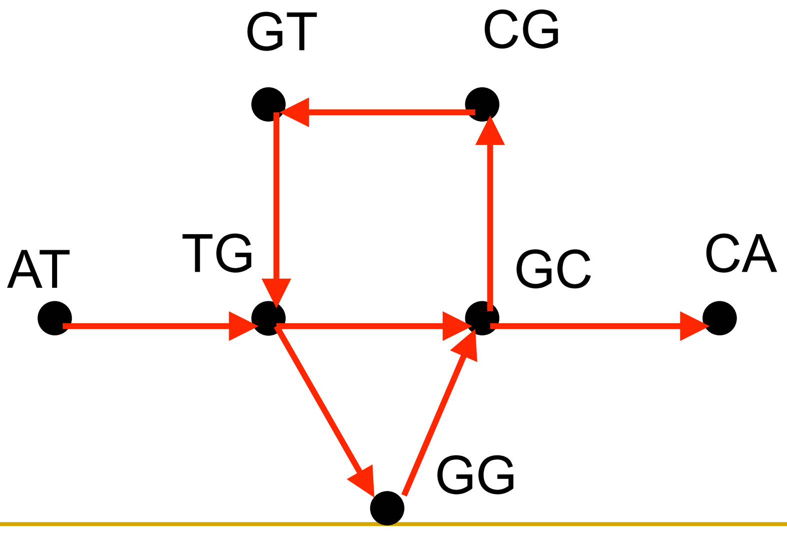


SBH: Eulerian Path Approach

- **Key Point**: A sequence reconstruction will actually correspond to an *Eulerian* path in this graph.

- Recall that an Eulerian path is "easy" to find (one can always be found in linear time)…so we have found a simple solution to SBH!

- In our example, two solutions:
  1. ATGGCGTGCA
  2. ATGCGTGGCA



But…How Do We Know an Eulerian Path Exists?

- A graph is **balanced** if for every vertex the number of incoming edges equals to the number of outgoing edges. We write this for vertex *v* as:

$$in(v)=out(v)$$

- **Theorem**: A connected graph is *Eulerian* (i.e. contains an Eulerian cycle) if and only if each of its vertices is balanced.

- We will prove this by demonstrating the following:
  1. Every Eulerian graph is balanced.
  2. Every balanced graph is Eulerian.

Every Eulerian Graph is Balanced

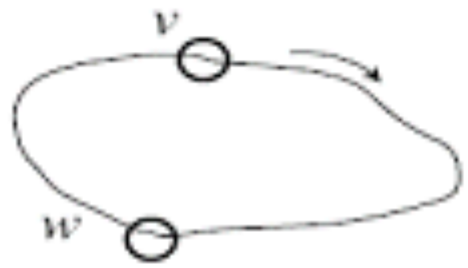
- Suppose we have an Eulerian graph G. Call *C* the Eulerian cycle of G, and let *v* be any vertex of G.

- For every edge *e* entering *v*, we can pair *e* with an edge leaving *v*, which is simply the edge in our cycle *C* that follows *e*.

- Therefore it directly follows that *in(v)=out(v)* as needed, and since our choice of *v* was arbitrary, this relation must hold for all vertices in G, so we are finished with the first part.

Every Balanced Graph is Eulerian

- Next, suppose that we have a balanced graph G.
- We will actually *construct* an Eulerian cycle in G.
- Start with an arbitrary vertex *v* and form a path in G without repeated edges until we reach a "dead end," meaning a vertex with no unused edges leaving it.
- G is balanced, so every time we enter a vertex *w* that isn't *v* during the course of our path, we can find an edge leaving *w*. So our dead end is *v* and we have a *cycle*.

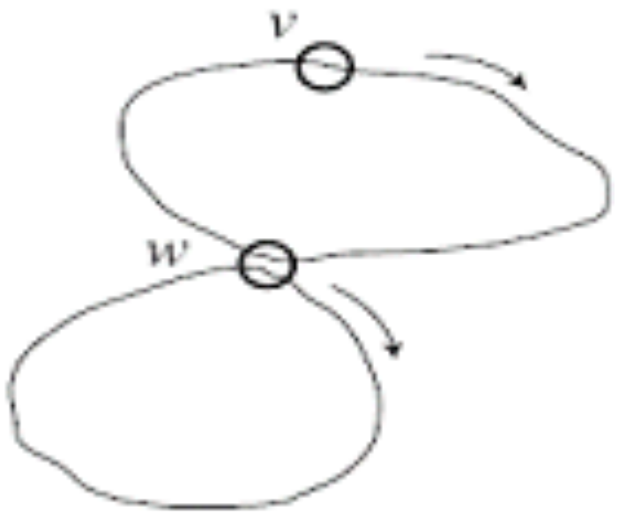


Every Balanced Graph is Eulerian

- We have two simple cases for our cycle, which we call C:
  1. C is an Eulerian cycle → G is Eulerian → DONE.
  2. C is not an Eulerian cycle.

- So we can assume that C is not an Eulerian cycle, which means that C contains vertices which have untraversed edges.

- Let *w* be such a vertex, and start a new path from *w*. Once again, we must obtain a cycle, say C'.

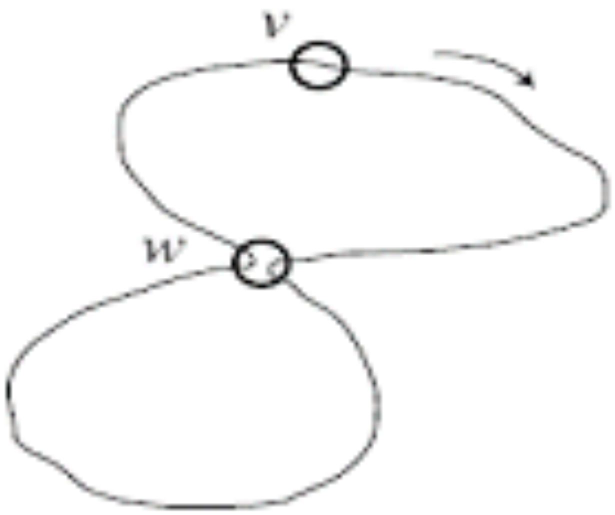


Every Balanced Graph is Eulerian

- Combine our cycles C and C' into a bigger cycle C* by swapping edges at *w* (see figure).

- Once again, we test C*:
  1. C* is an Eulerian cycle → G is Eulerian → DONE.
  2. C* is not an Eulerian cycle.

- If C* is not Eulerian, we iterate our procedure. Because G has a finite number of edges, we must eventually reach a point where our current cycle is Eulerian (Case 1 above). DONE.



Euler's Theorem: Extension

- A vertex v is **semi-balanced** if either $in(v) = out(v) + 1$ or $in(v) = out(v) - 1$.

- **Theorem**: A connected graph has an Eulerian path if and only if it contains at most two semi-balanced vertices and all other vertices are balanced.
  - If G has no semi-balanced vertices, DONE.
  - If G has two semi-balanced vertices, connect them with a new edge *e*, so that the graph G + *e* is balanced and must be Eulerian. Remove *e* from the Eulerian cycle in G + *e* to obtain an Eulerian path in G.
  - **Think**: Why can G not have just one semi-balanced vertex?

Some Difficulties with SBH

- **Fidelity of Hybridization:** It is difficult to detect differences between probes hybridized with perfect matches and those with one mismatch.

- **Array Size:** The effect of low fidelity can be decreased with longer *l*-mers, but array size increases exponentially in *l*. Array size is limited with current technology.

- **Practicality:** SBH is still impractical. As DNA microarray technology improves, SBH may become practical in the future.

Some Difficulties with SBH

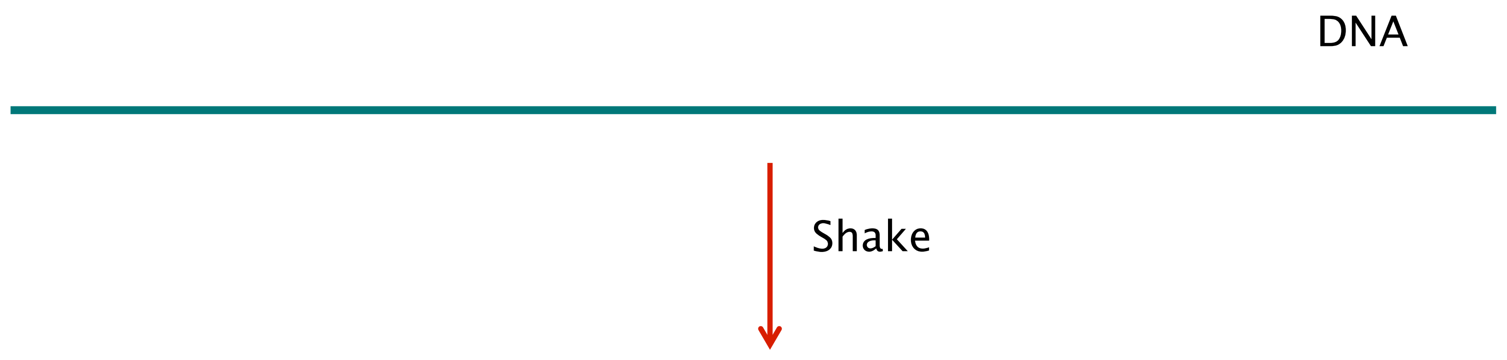
- **Practicality Again**: Although SBH is still impractical, it spearheaded expression analysis and SNP analysis techniques.

- **Practicality Again and Again**: In 2007 Solexa (now Illumina) developed a new DNA sequencing approach that generates so many short *l*-mers that they essentially mimic a universal DNA array.

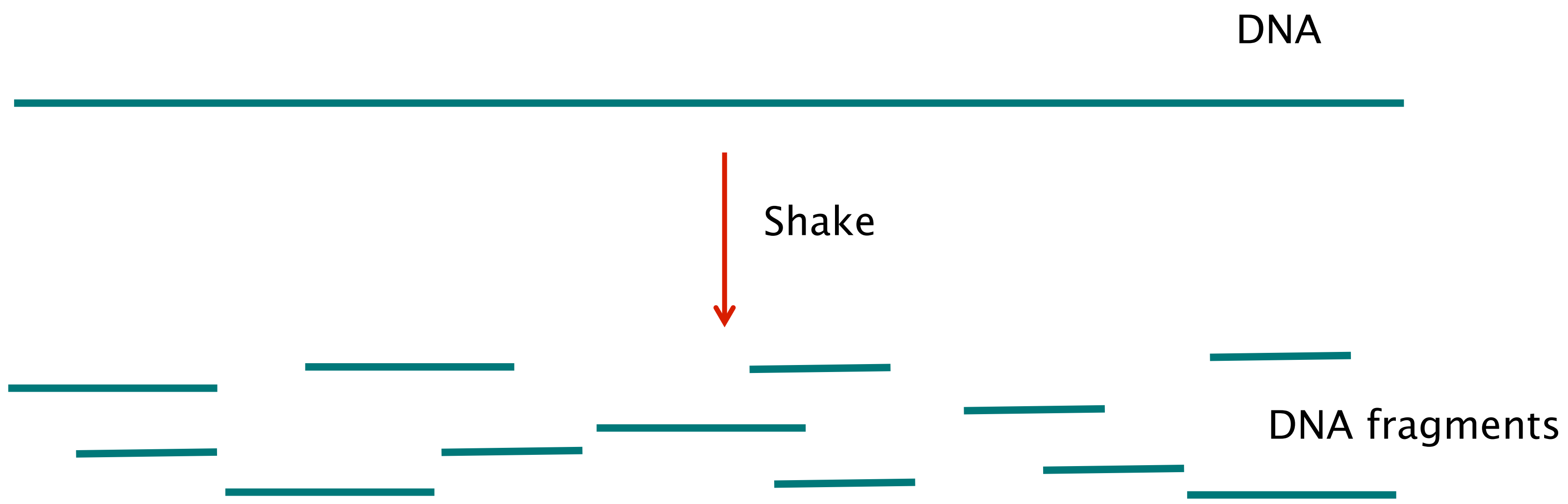
Section 7: Fragment Assembly & Repeats in DNA

Traditional DNA Sequencing

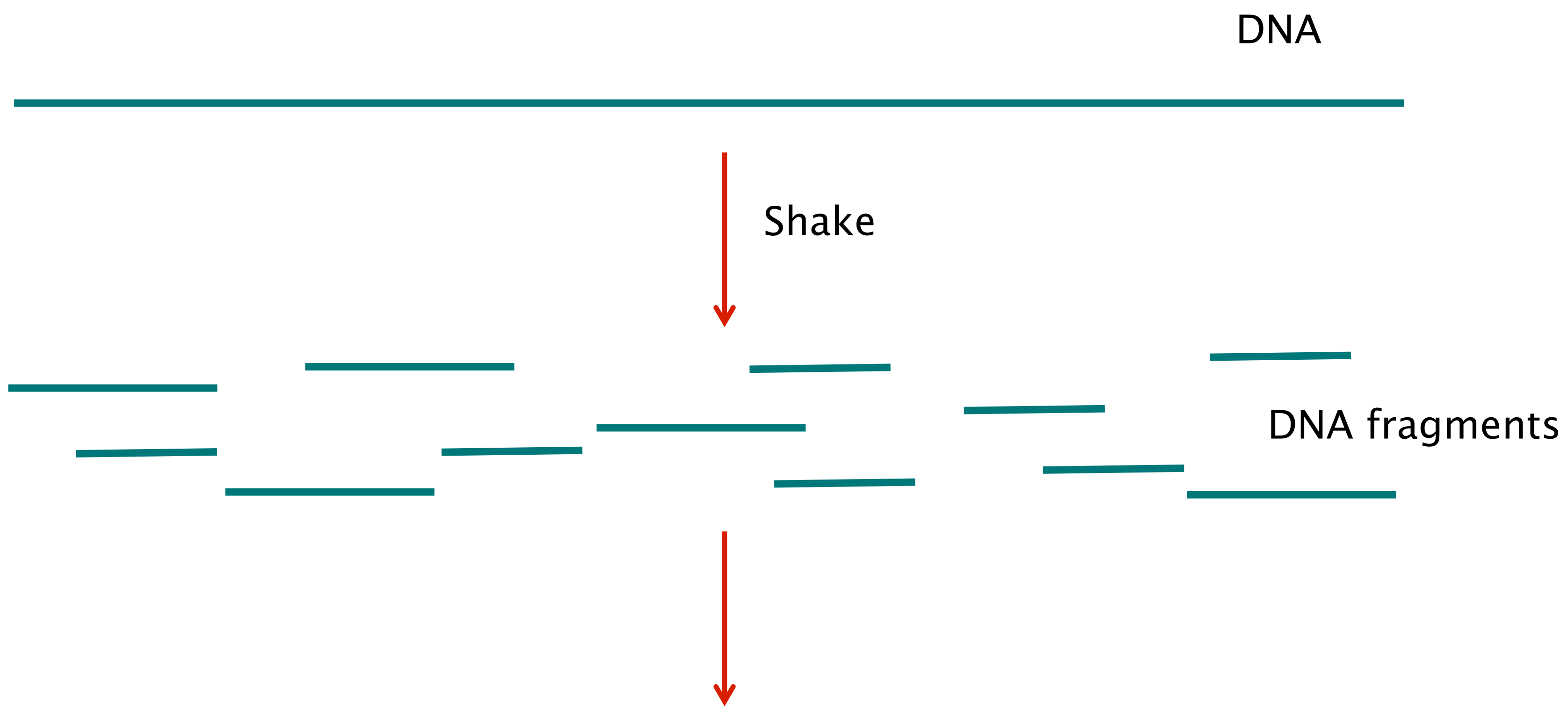
Traditional DNA Sequencing



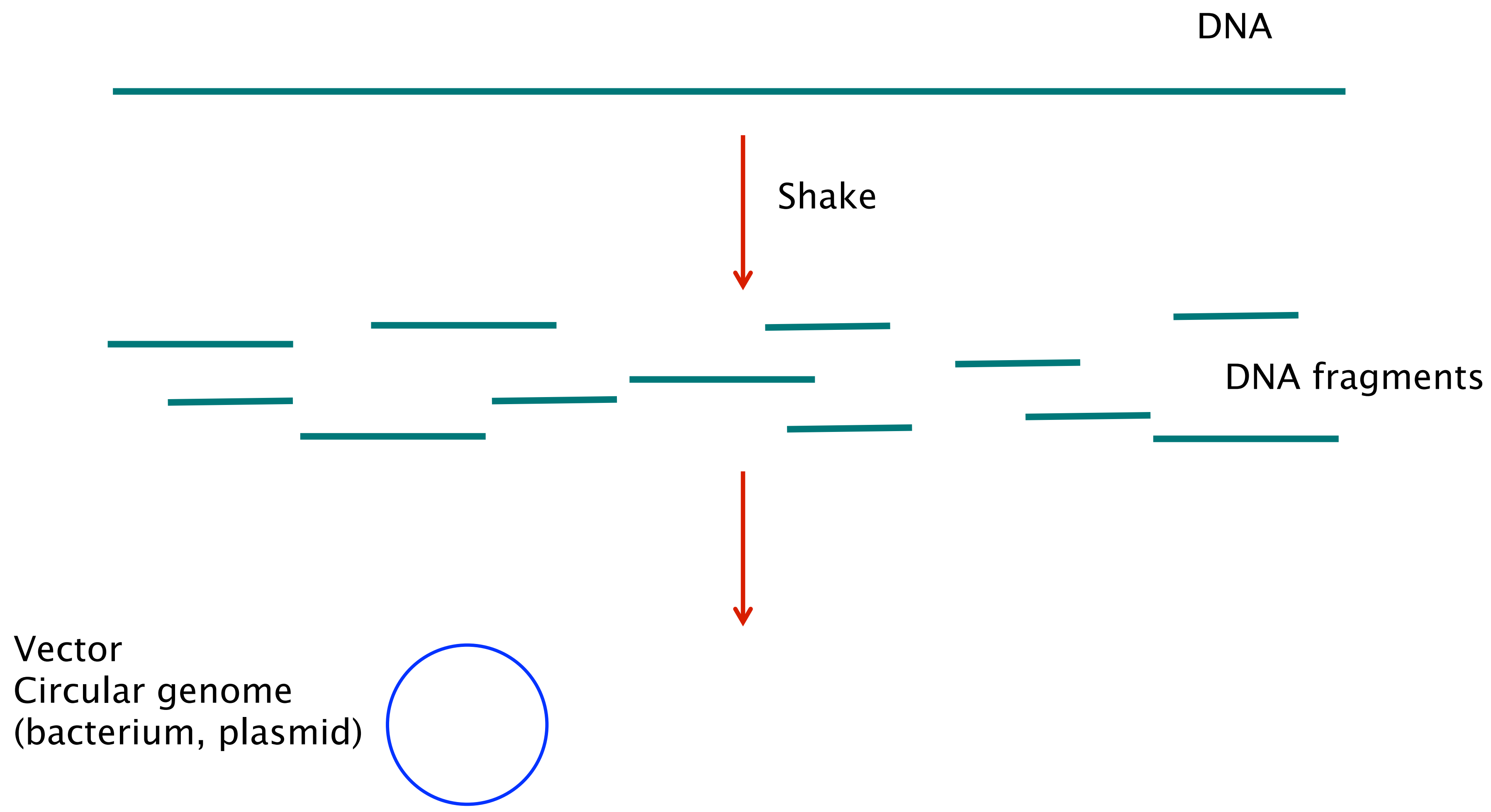
Traditional DNA Sequencing



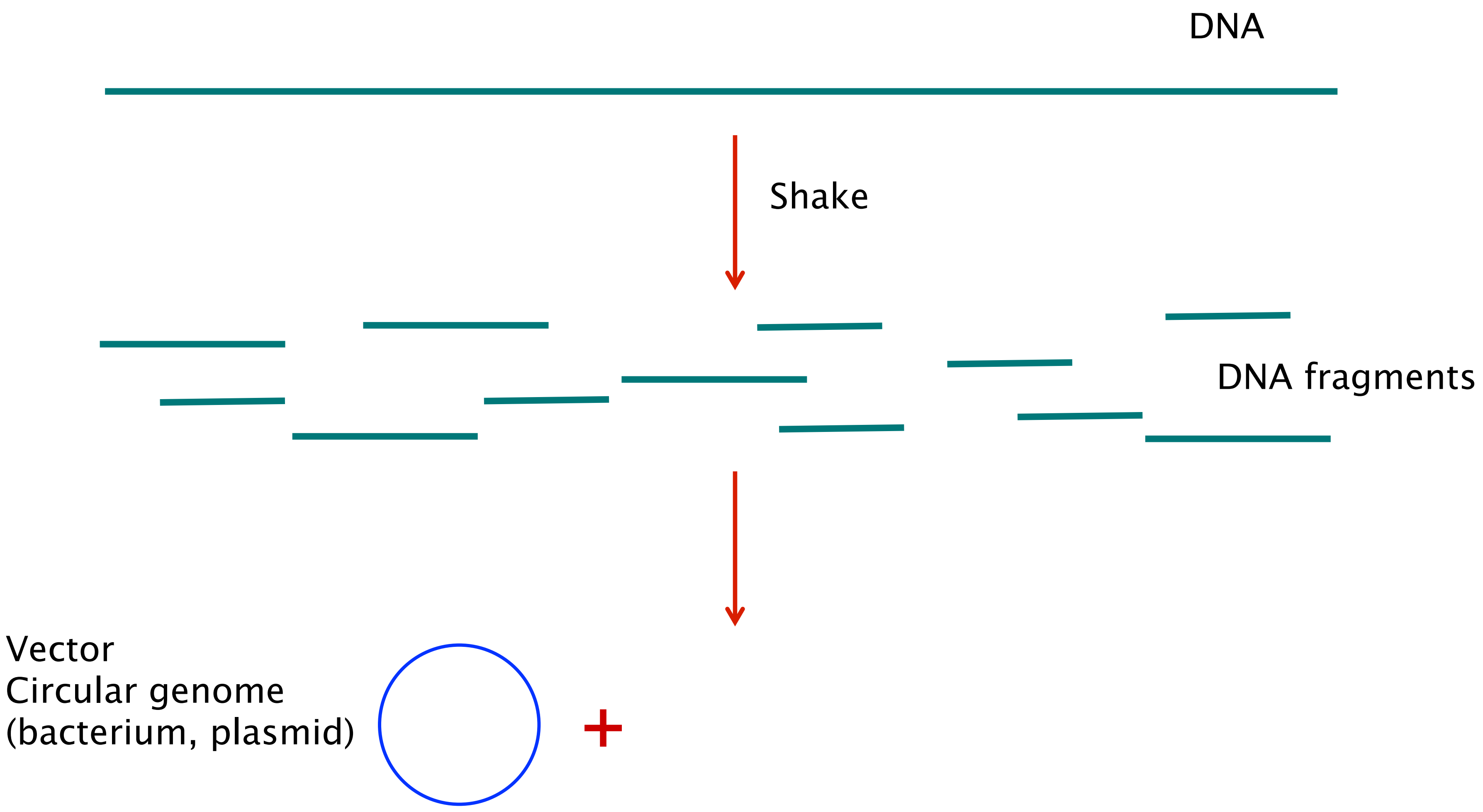
Traditional DNA Sequencing



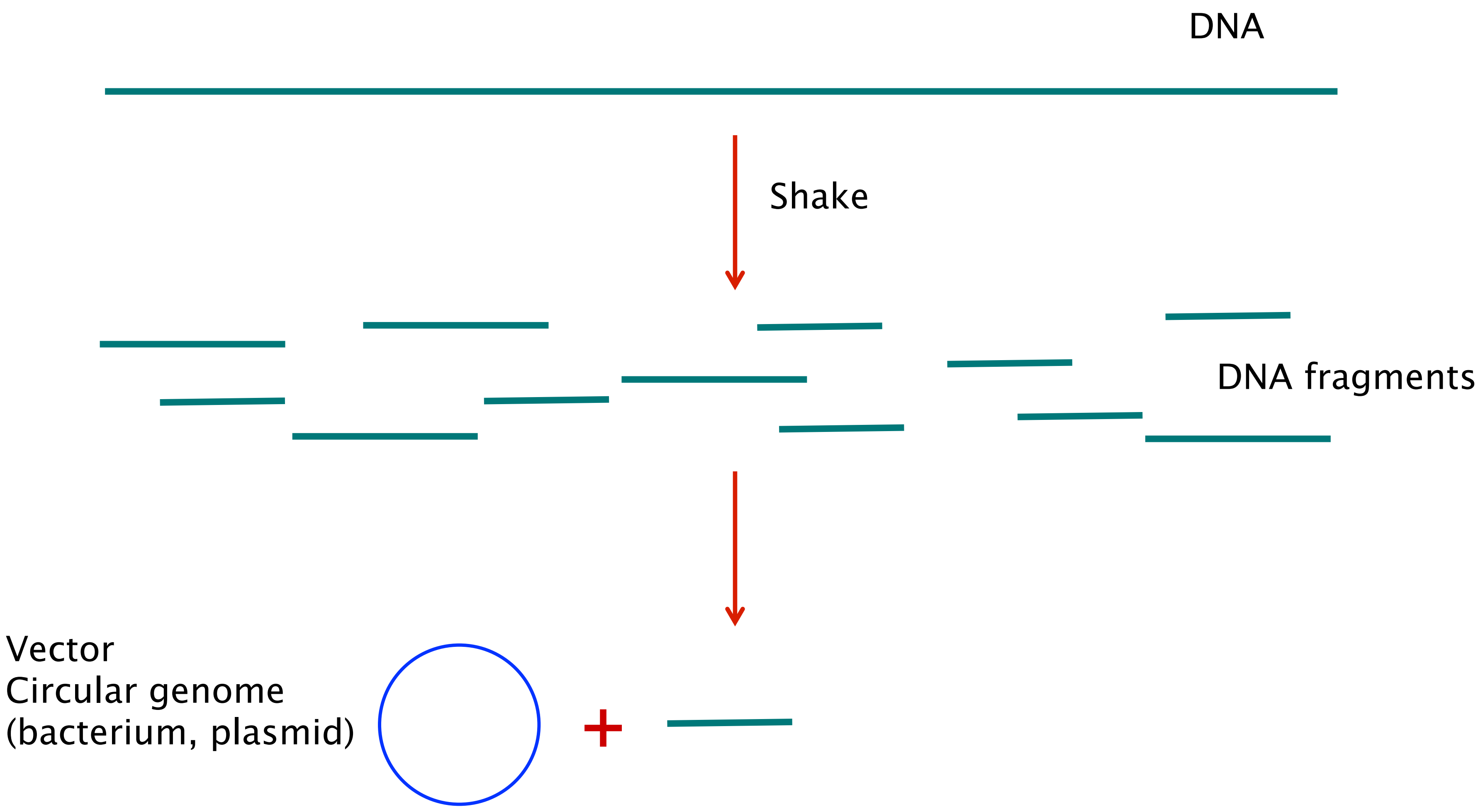
Traditional DNA Sequencing



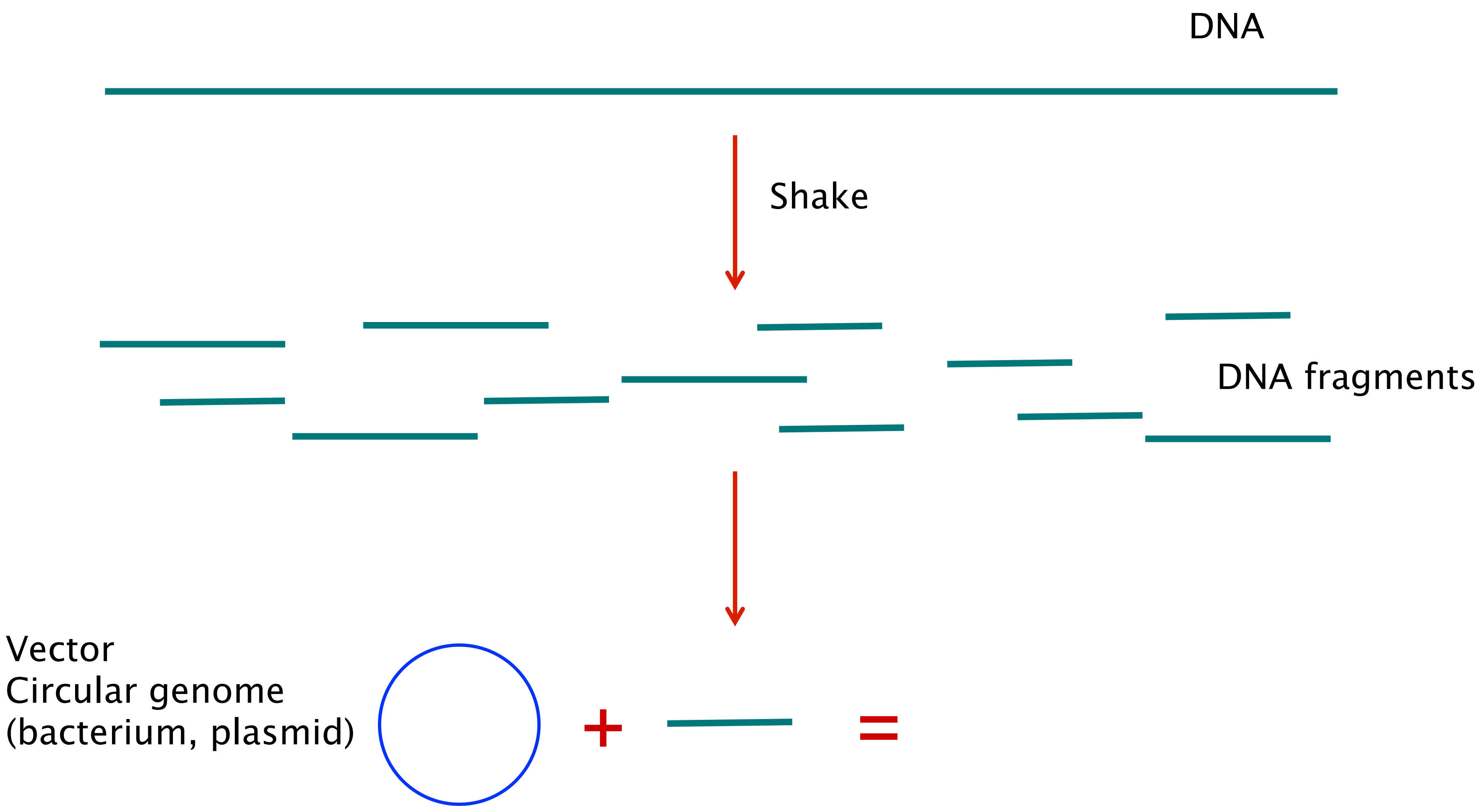
Traditional DNA Sequencing



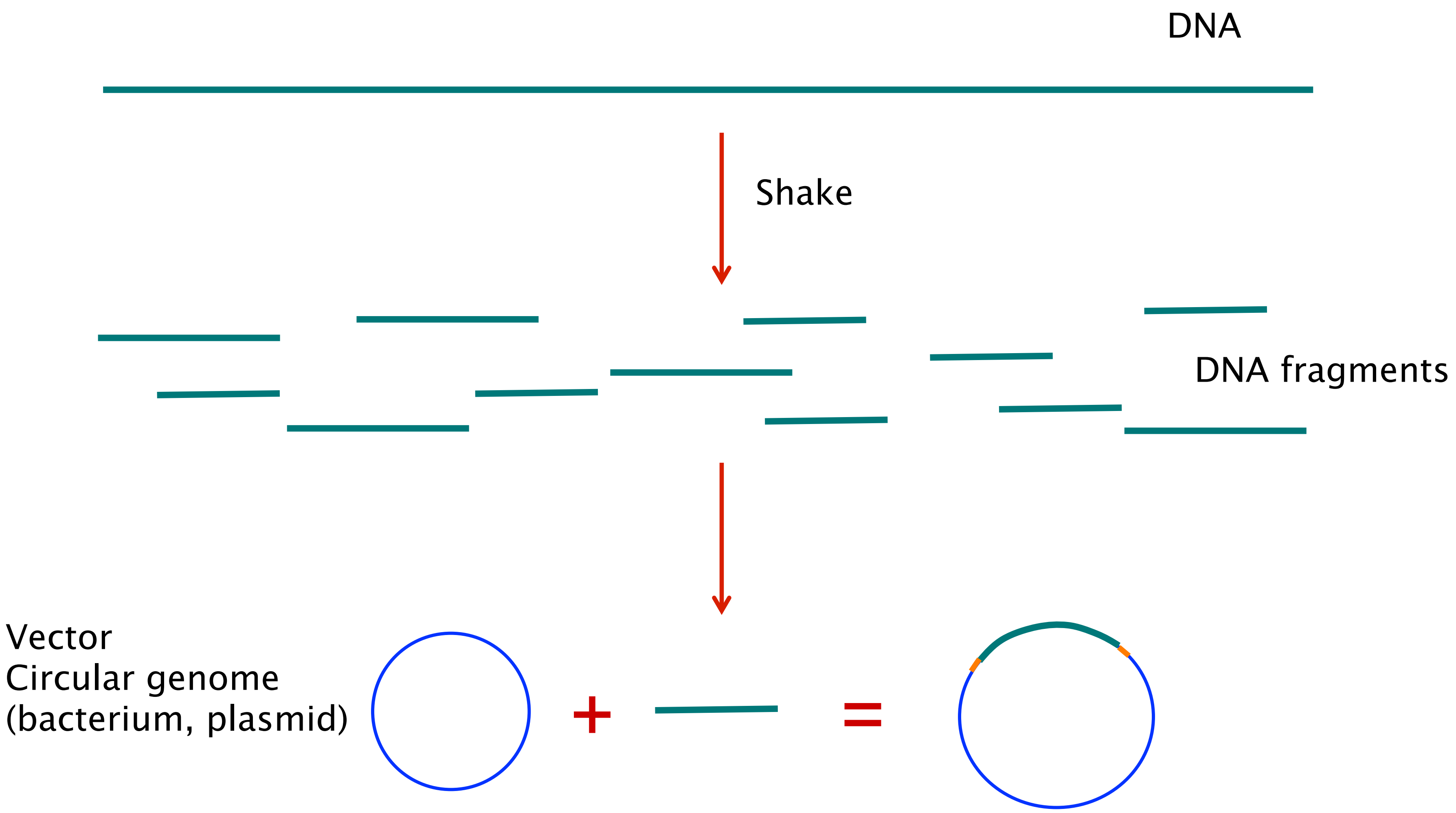
Traditional DNA Sequencing



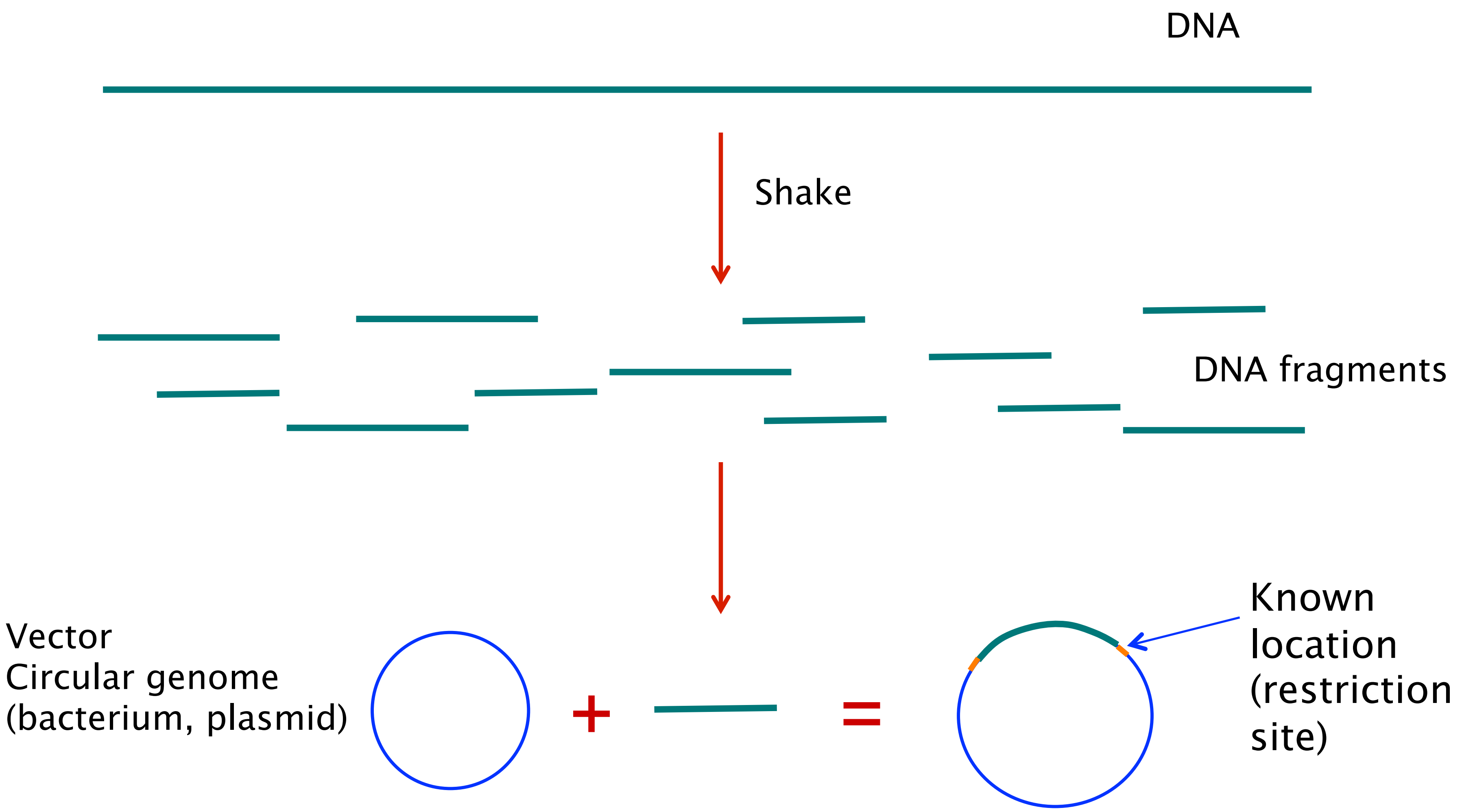
Traditional DNA Sequencing



Traditional DNA Sequencing

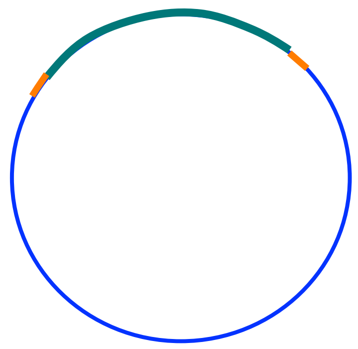


Traditional DNA Sequencing

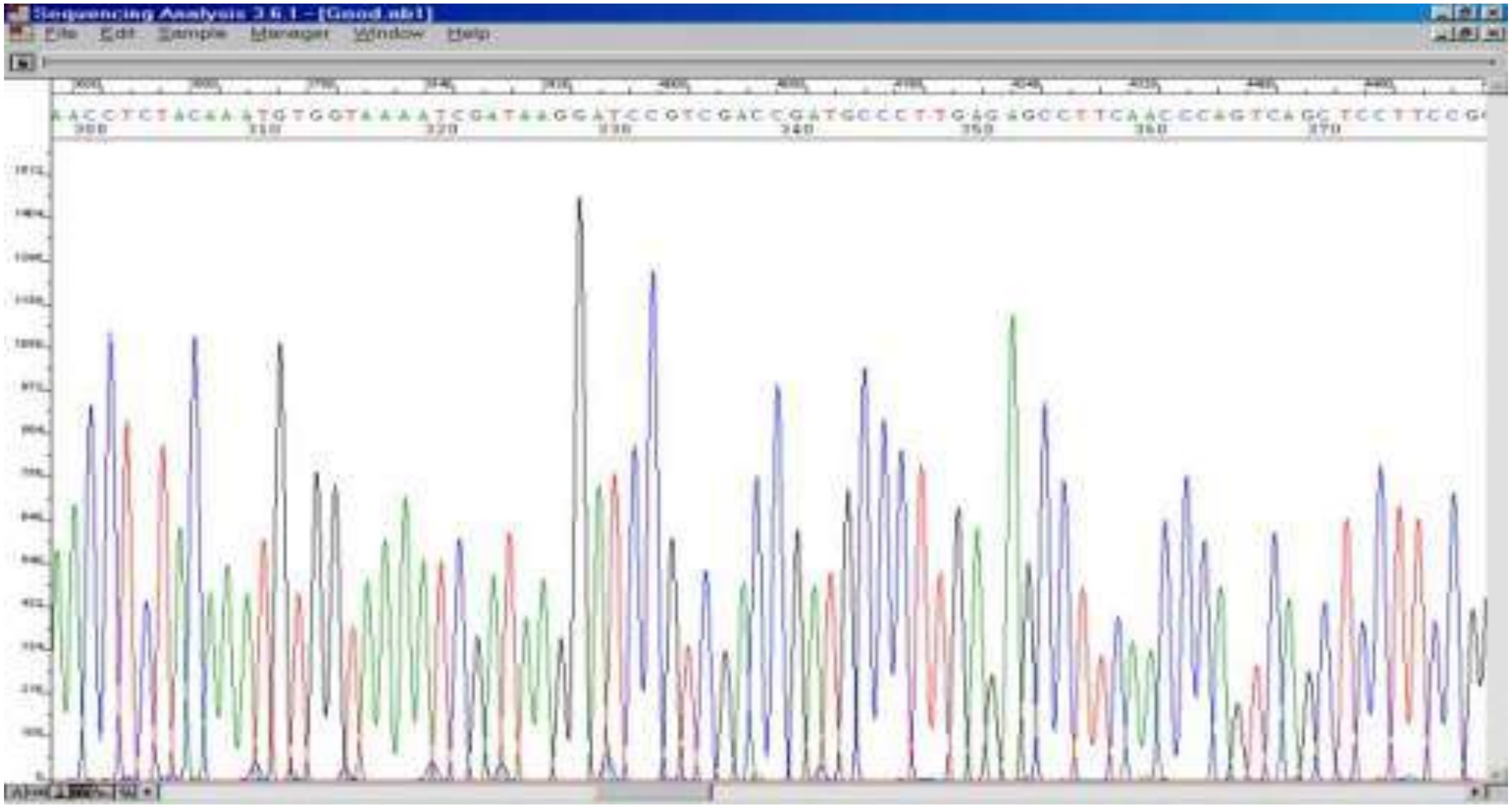


Different Types of Vectors

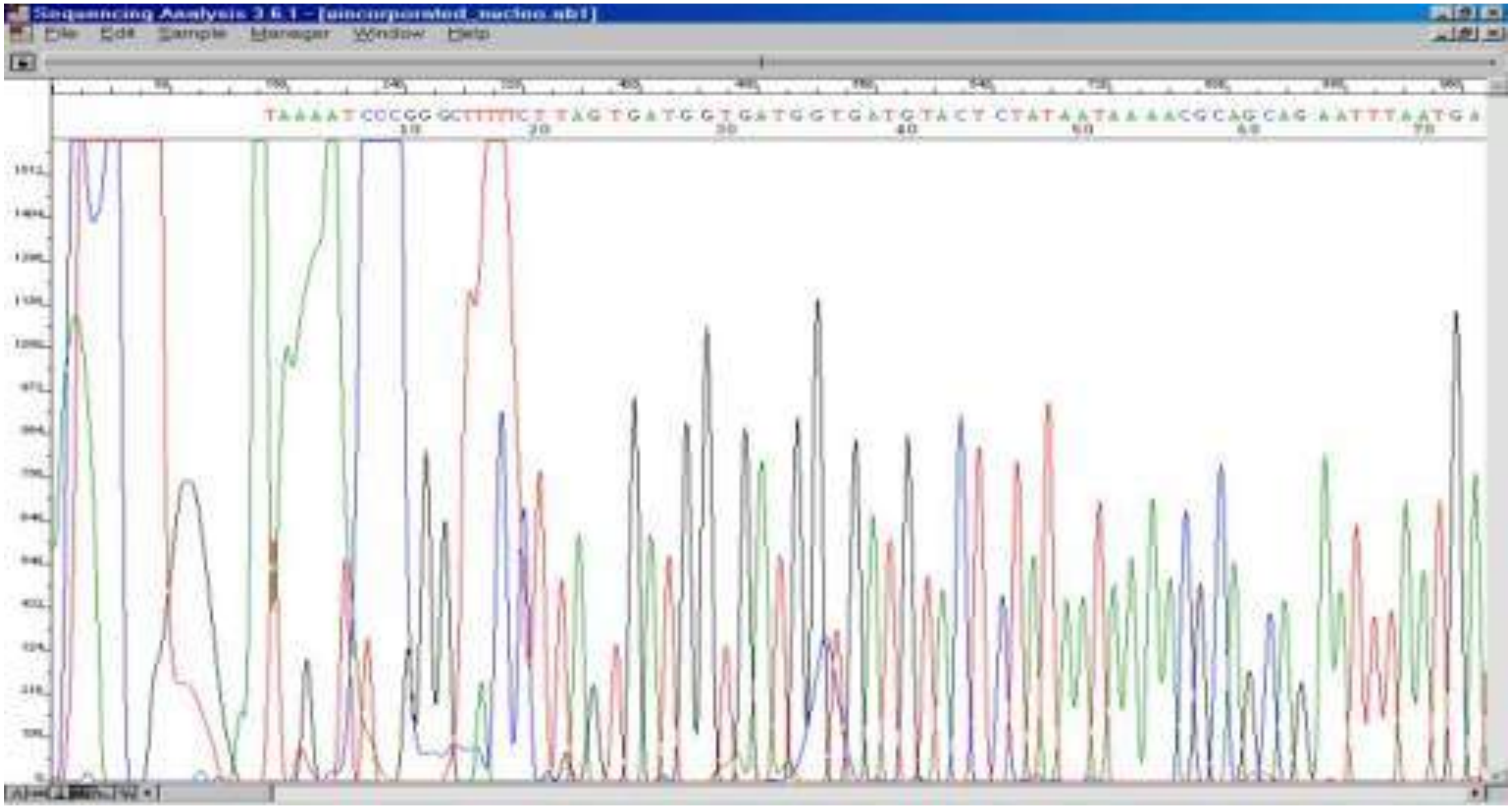
Vector	**Size of Insert (bp)**
Plasmid	2,000 - 10,000
Cosmid	40,000
BAC (Bacterial Artificial Chromosome)	70,000 - 300,000
YAC (Yeast Artificial Chromosome)	> 300,000 Not used much recently



Electrophoresis Diagrams



Electrophoresis Diagrams: Hard to Read

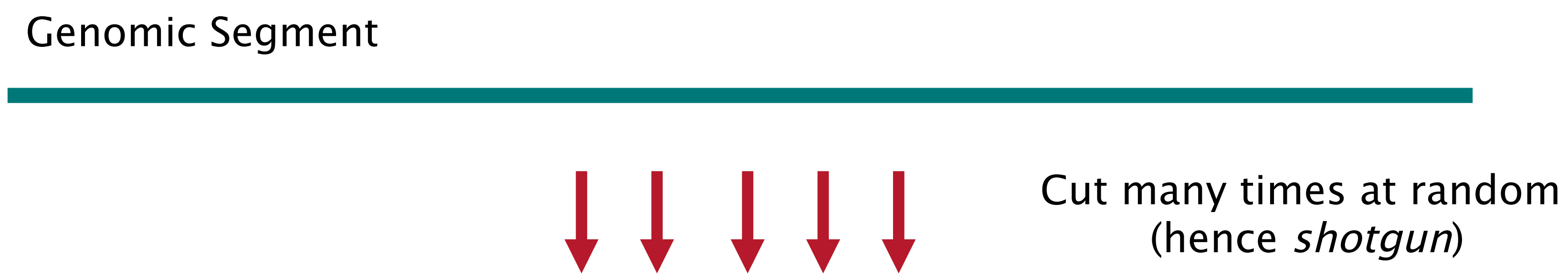

Sequencing Analysis 3.6.1 - [unincorporated_nucleo.ab1]
File Edit Sample Manager Window Help
TAAAATCCCGG GCTTTTTCT TAGTGATGGTGATGGTGATGTACT CTATAATAAAACGCAGCAG AATTTAATGA
10
20
30
40
50
60
70

Reading an Electropherogram

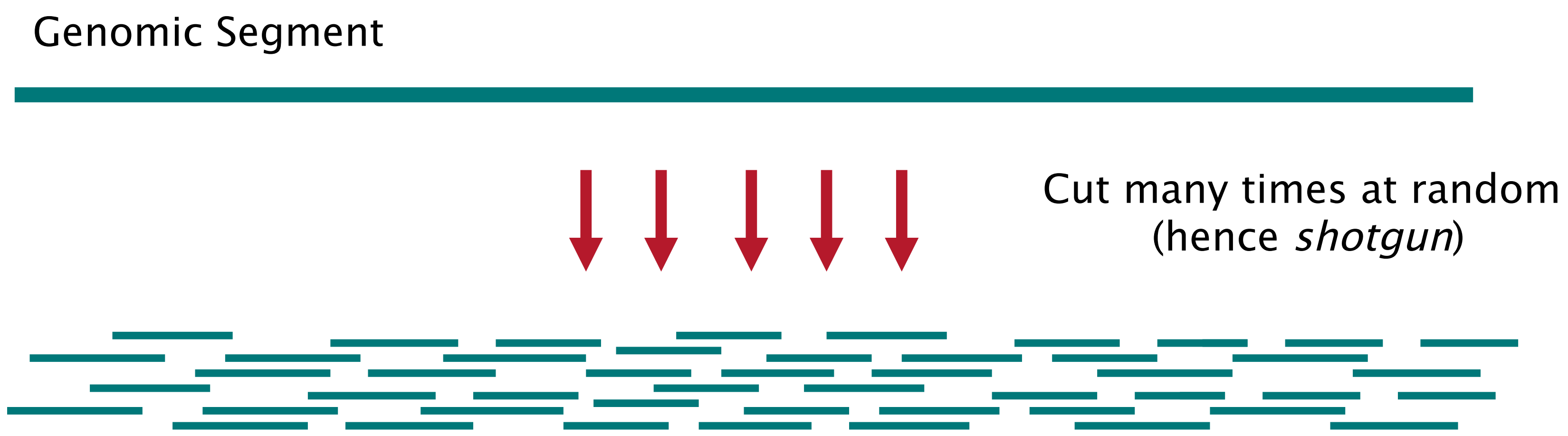
- Reading an Electropherogram requires four processes:
  1. Filtering
  2. Smoothening
  3. Correction for length compressions
  4. A method for calling the nucleotides – **PHRED**

Shotgun Sequencing

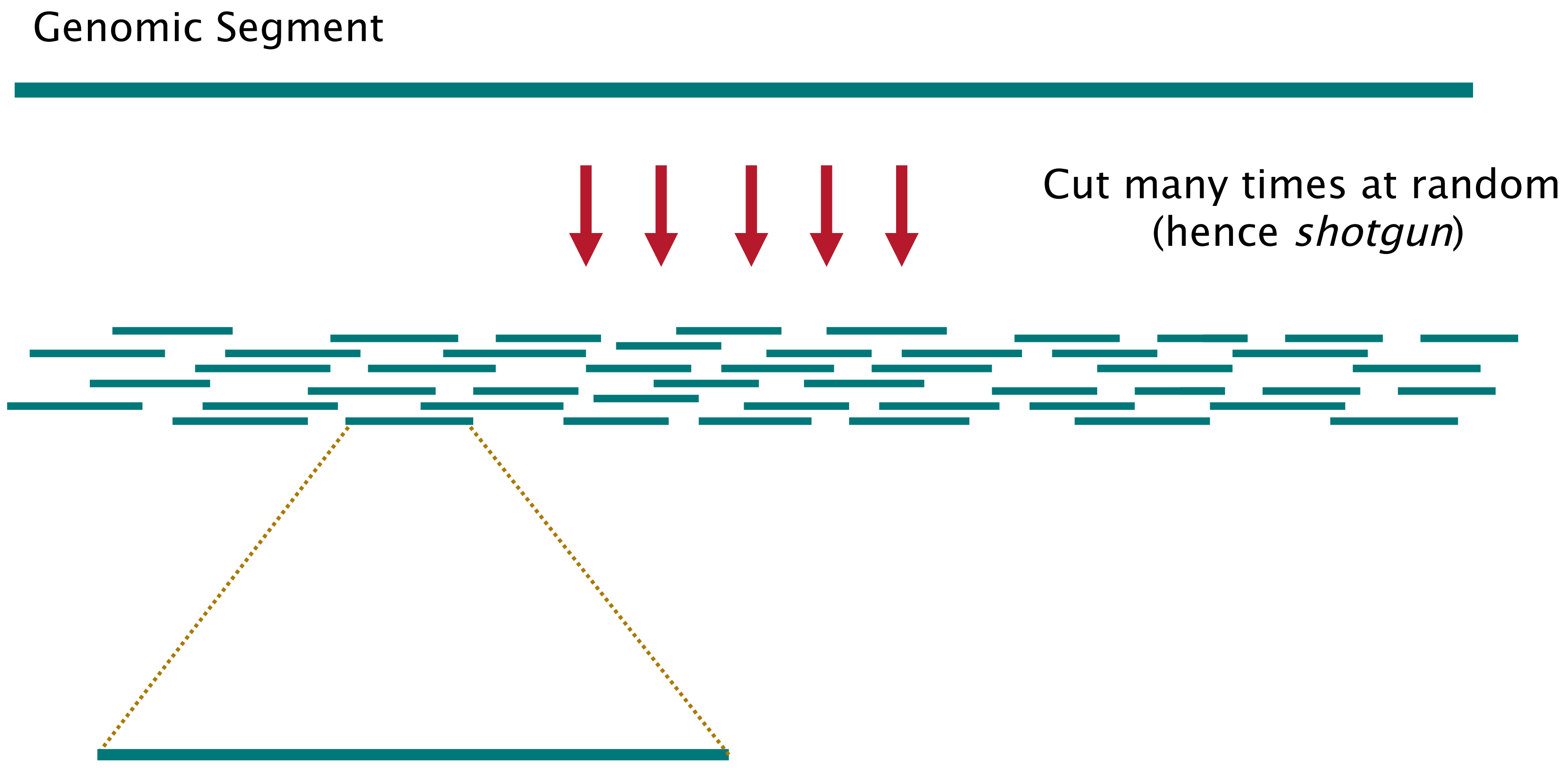
Shotgun Sequencing



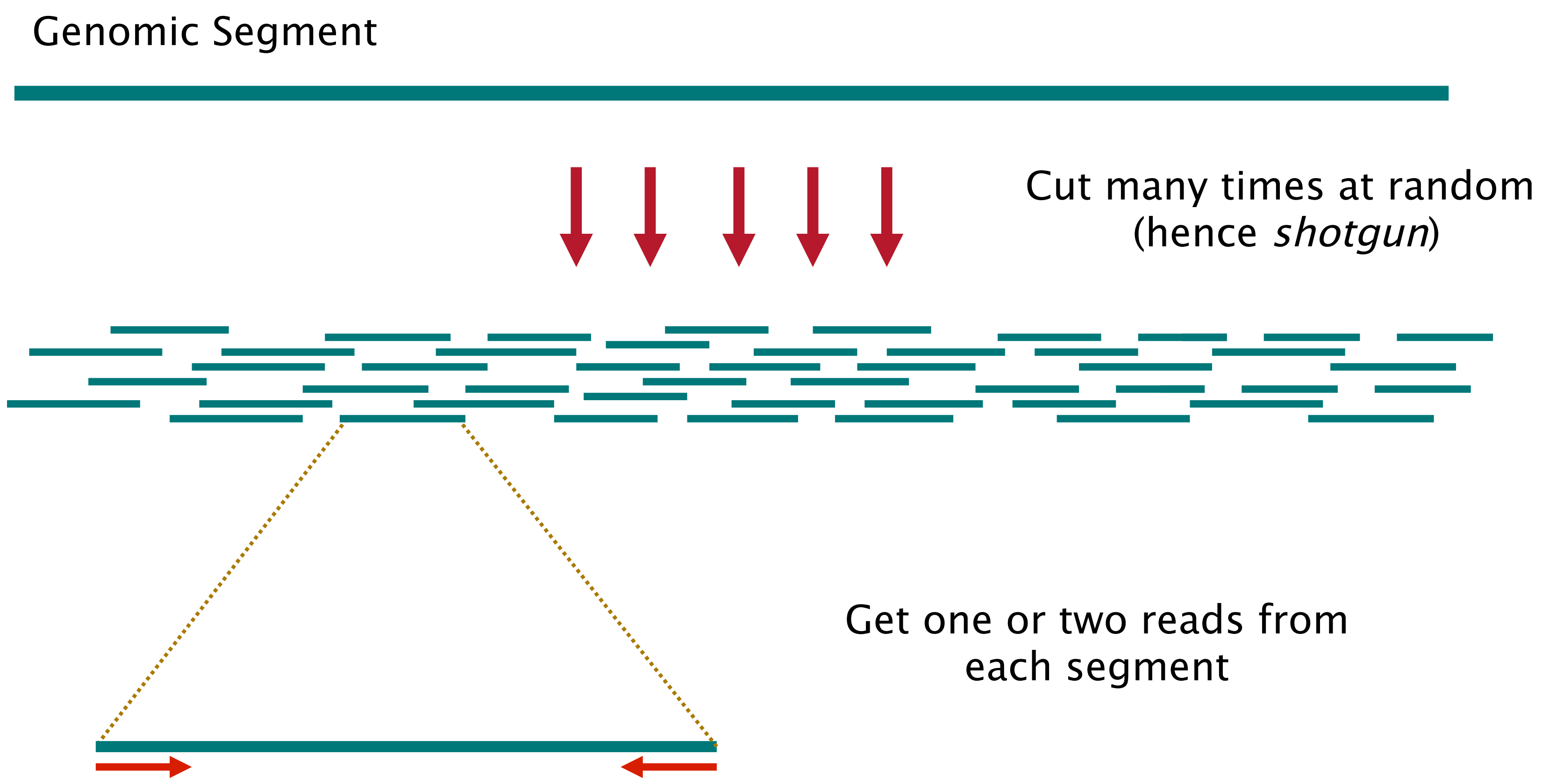
Shotgun Sequencing



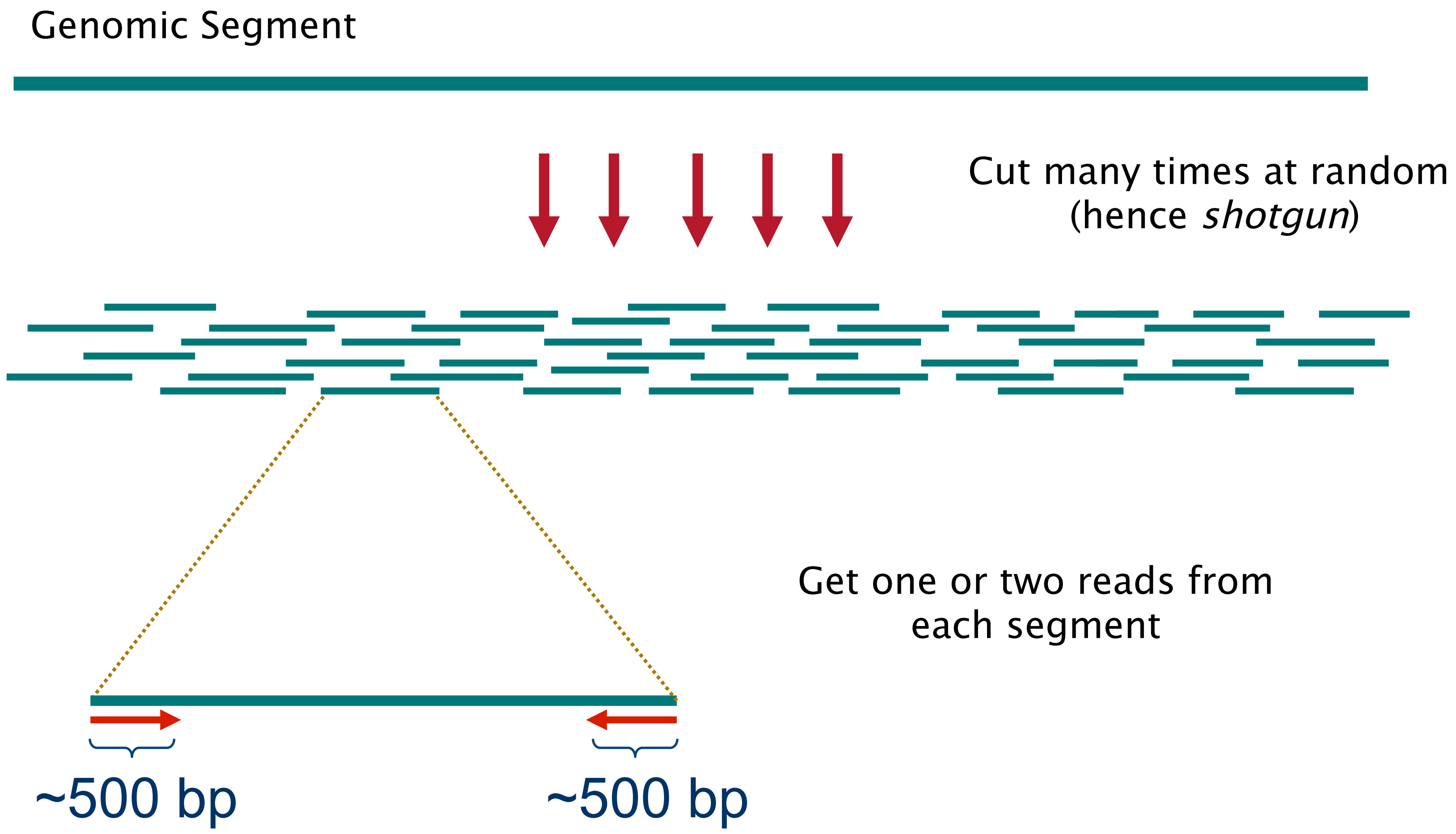
Shotgun Sequencing



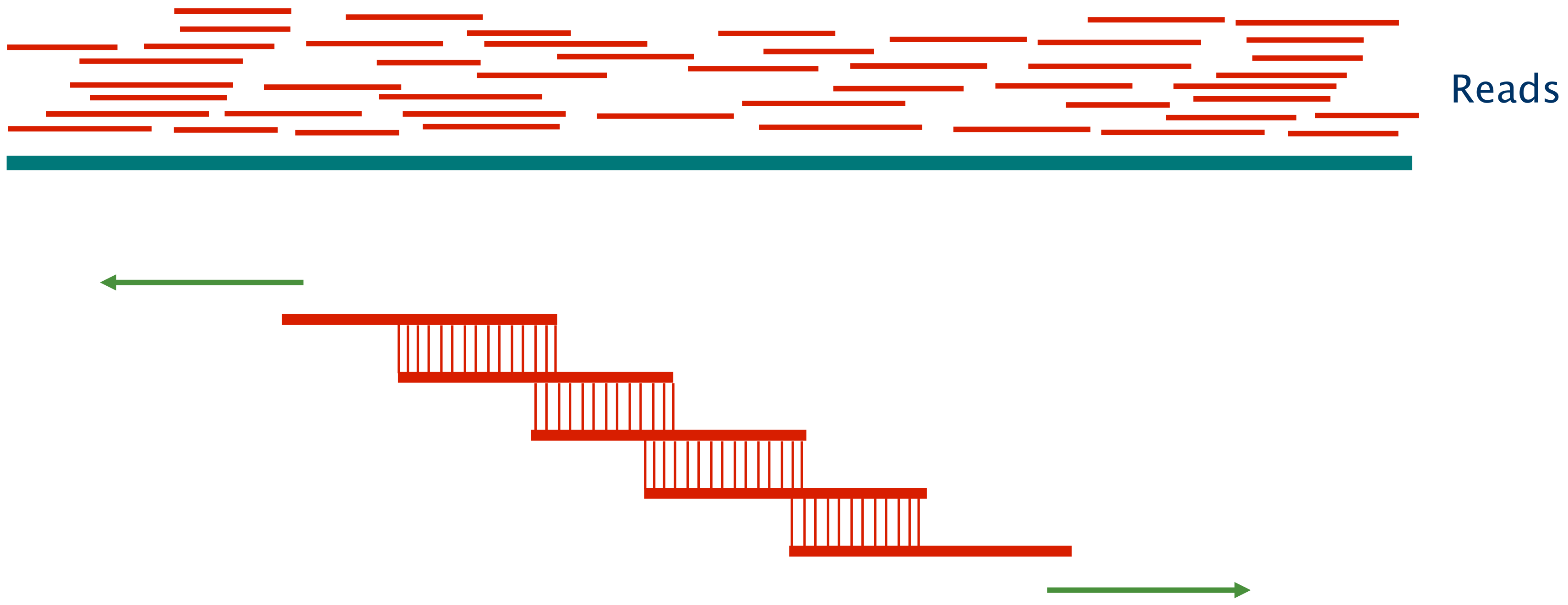
Shotgun Sequencing


Genomic Segment
Cut many times at random (hence *shotgun*)
Get one or two reads from each segment

Shotgun Sequencing

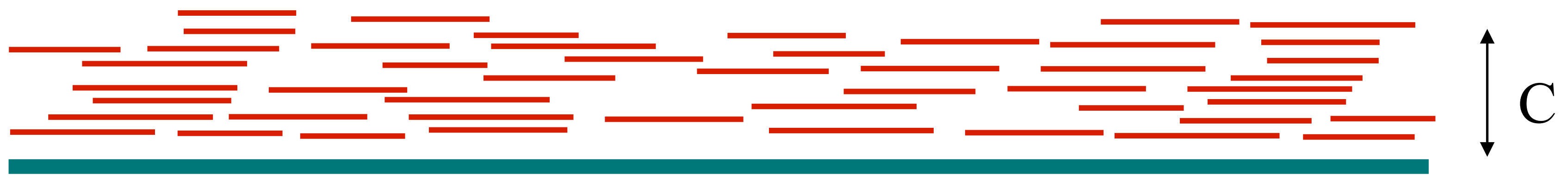

Genomic Segment
Cut many times at random (hence *shotgun*)
Get one or two reads from each segment
~500 bp
~500 bp

Fragment Assembly



- Cover region with ~7-fold redundancy.
- Overlap reads and extend to reconstruct the original genomic region.

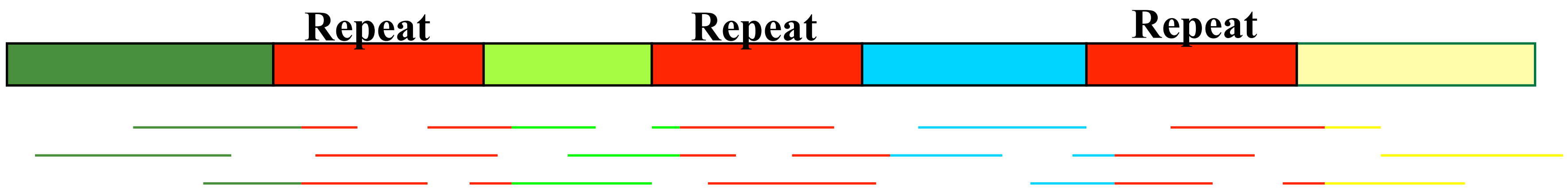
Read Coverage



- Length of genomic segment: ***L***
- Number of reads: ***n***
- Length of each read: ***l***
- Define the **coverage** as: $C = n\, l / L$
- **Question**: How much coverage is enough?
  - Lander-Waterman Model: Assuming uniform distribution of reads, $C = 10$ results in 1 gap in coverage per million nucleotides.

Challenges in Fragment Assembly

- Repeats: A **major** problem for fragment assembly.
- More than 50% of human genome are repeats:
  - Over 1 million *Alu* repeats (about 300 bp).
  - About 200,000 LINE repeats (1000 bp and longer).



DNA Assembly Analogy: Triazzle

- A Triazzle ® puzzle has only 16 pieces and looks simple.
- BUT… there are many repeats!
- The repeats make it very difficult to solve.
- This repetition is what makes fragment assembly is so difficult.

http://www.triazzle.com/

Repeat Classification

Repeat Type	**Explanation**
• Low-Complexity DNA	(e.g. ATATATATACATA…)
• Microsatellite repeats	$(a_1 \ldots a_k)^N$ where k ~ 3-6 (e.g. CAGCAGTAGCAGCACCAG)
• Gene Families	genes duplicate & then diverge
• Segmental duplications	~very long, very similar copies

Repeat Classification

Repeat Type	**Explanation**
• SINE Transposon	Short Interspersed Nuclear Elements (e.g., Alu: ~300 bp long, 10^6 copies)
• LINE Transposon	Long Interspersed Nuclear Elements ~500 - 5,000 bp long, 200,000 copies
• LTR retroposons	Long Terminal Repeats (~700 bp) at each end

Section 8: Fragment Assembly Algorithms

Assembly Method: Overlap-Layout-Consensus

- **Assemblers:** ARACHNE, PHRAP, CAP, TIGR, CELERA

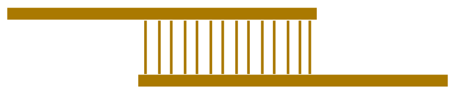
Assembly Method: Overlap-Layout-Consensus

- **Assemblers:** ARACHNE, PHRAP, CAP, TIGR, CELERA
- Three steps:

Assembly Method: Overlap-Layout-Consensus

- **Assemblers:** ARACHNE, PHRAP, CAP, TIGR, CELERA

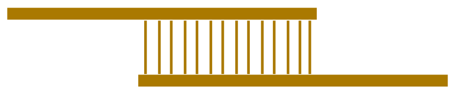
- Three steps:
  1. **Overlap**: Find potentially overlapping reads.

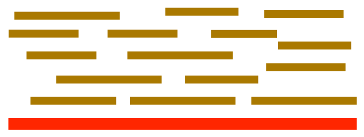


Assembly Method: Overlap-Layout-Consensus

- **Assemblers:** ARACHNE, PHRAP, CAP, TIGR, CELERA

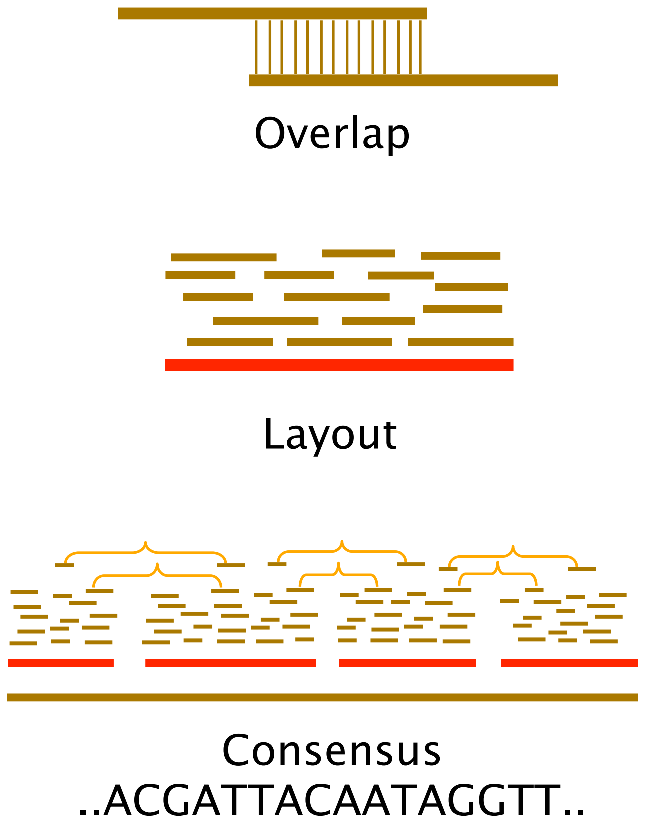
- Three steps:
  1. **Overlap**: Find potentially overlapping reads.
  2. **Layout**: Merge reads into contigs and contigs into supercontigs.


Overlap


Layout

Assembly Method: Overlap-Layout-Consensus

- **Assemblers:** ARACHNE, PHRAP, CAP, TIGR, CELERA
- Three steps:
  1. **Overlap**: Find potentially overlapping reads.
  2. **Layout**: Merge reads into contigs and contigs into supercontigs.
  3. **Consensus**: Derive the DNA sequence and correct any read errors.



Step 1: Overlap

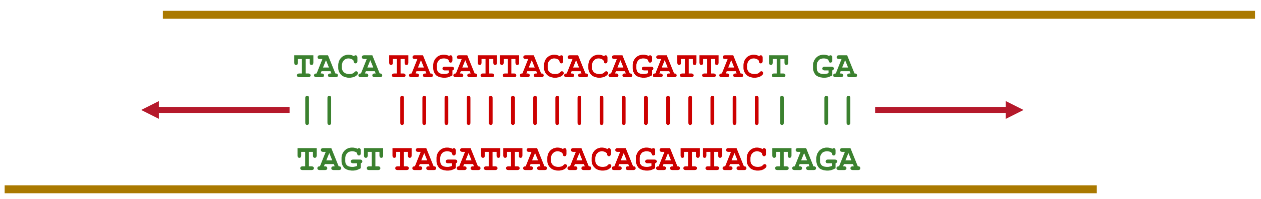
- Find the best match between the suffix of one read and the prefix of another.

- Due to sequencing errors, we need to use dynamic programming to find the optimal *overlap alignment*.

- Apply a filtration method to filter out pairs of fragments that do not share a significantly long common substring.

Step 1: Overlap

- Sort all *k*-mers in reads (*k* ~ 24).
- Find pairs of reads sharing a *k*-mer.
- Extend to full alignment—throw away if not >95% similar.

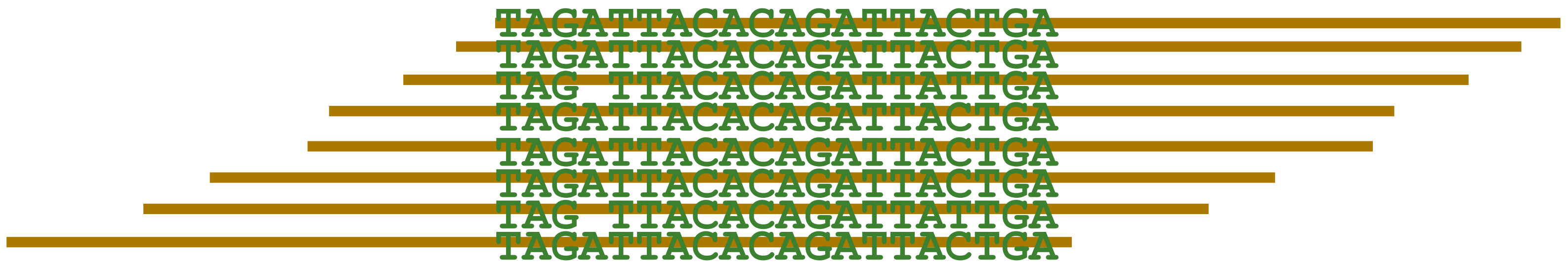


Step 1: Overlap

- A k-mer that appears N times initiates N^2 comparisons.
- For an *Alu* that appears 10^6 times, we will have 10^{12} comparisons – this is too many.
- **Solution:** Discard all k-mers that appear more than $t \times$ Coverage, ($t \sim 10$)

Step 2: Layout

- We next create local multiple alignments from the overlapping reads.



Step 2: Layout

- Repeats are a major challenge.
- Do two aligned fragments really overlap, or are they from two copies of a repeat?
- Solution: *repeat masking* – hide the repeats!

Step 2: Layout

- Repeats are a major challenge.
- Do two aligned fragments really overlap, or are they from two copies of a repeat?
- Solution: *repeat masking* – hide the repeats!
- Masking results in a high rate of misassembly (~20 %).

Step 2: Layout

- Repeats are a major challenge.
- Do two aligned fragments really overlap, or are they from two copies of a repeat?
- Solution: *repeat masking* – hide the repeats!
- Masking results in a high rate of misassembly (~20 %).
- Misassembly means a lot more work at the finishing step.

Step 2: Layout

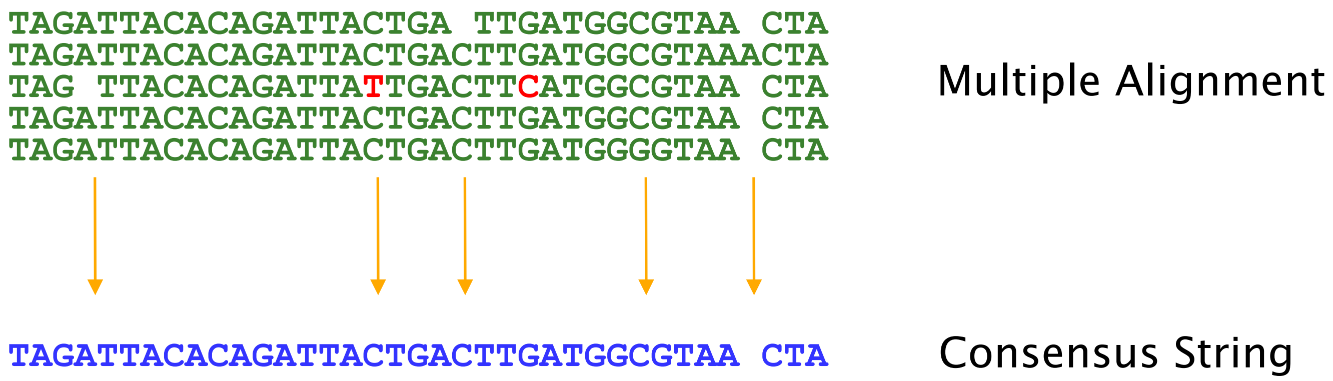
- Repeats shorter than read length are OK.
- Repeats with more base pair differences than the sequencing error rate are OK.
- To make a smaller portion of the genome *appear* repetitive, try to:
  - **Increase read length**
  - **Decrease sequencing error rate**

Step 3: Consensus

- A consensus sequence is derived from a profile of the assembled fragments.
- A sufficient number of reads are required to ensure a statistically significant consensus.
- Reading errors are corrected.

Step 3: Consensus

- Derive multiple alignment from pairwise read alignments.
- Derive each consensus base by weighted voting.



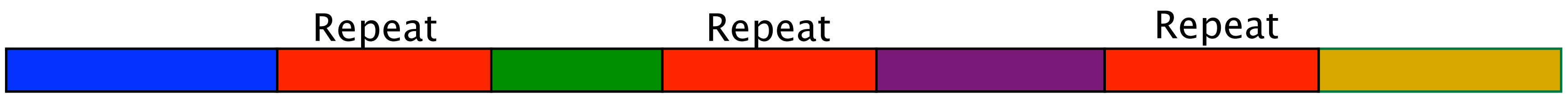
```
TAGATTACACAGATTACTGA TTGATGGCGTAA CTA
TAGATTACACAGATTACTGACTTGATGGCGTAAACTA
TAG TTACACAGATTATTGACTTCATGGCGTAA CTA
TAGATTACACAGATTACTGACTTGATGGCGTAA CTA
TAGATTACACAGATTACTGACTTGATGGGTAA CTA

TAGATTACACAGATTACTGACTTGATGGCGTAA CTA
```

Overlap Graph: Hamiltonian Approach

- Each vertex represents a read from the original sequence.
- Vertices are connected by an edge if they overlap.

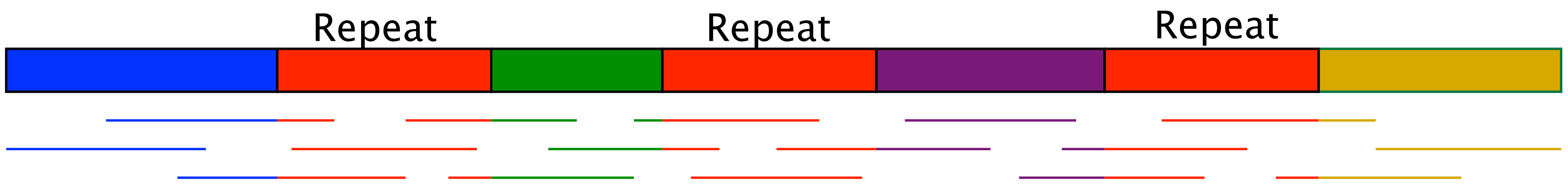
Overlap Graph: Hamiltonian Approach

- Each vertex represents a read from the original sequence.
- Vertices are connected by an edge if they overlap.



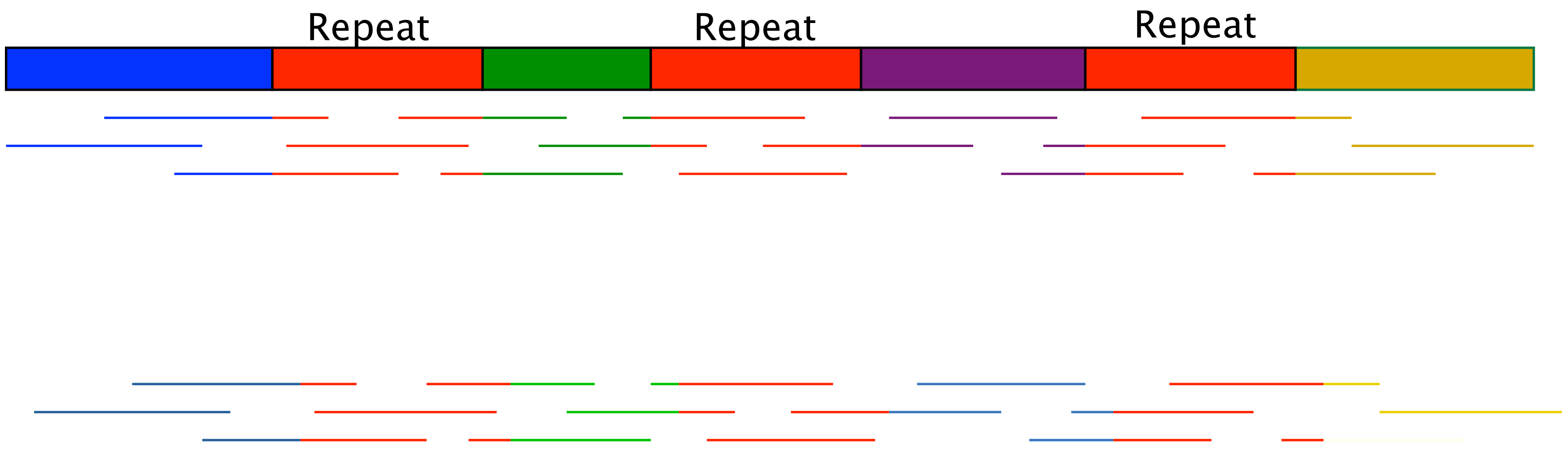
Overlap Graph: Hamiltonian Approach

- Each vertex represents a read from the original sequence.
- Vertices are connected by an edge if they overlap.



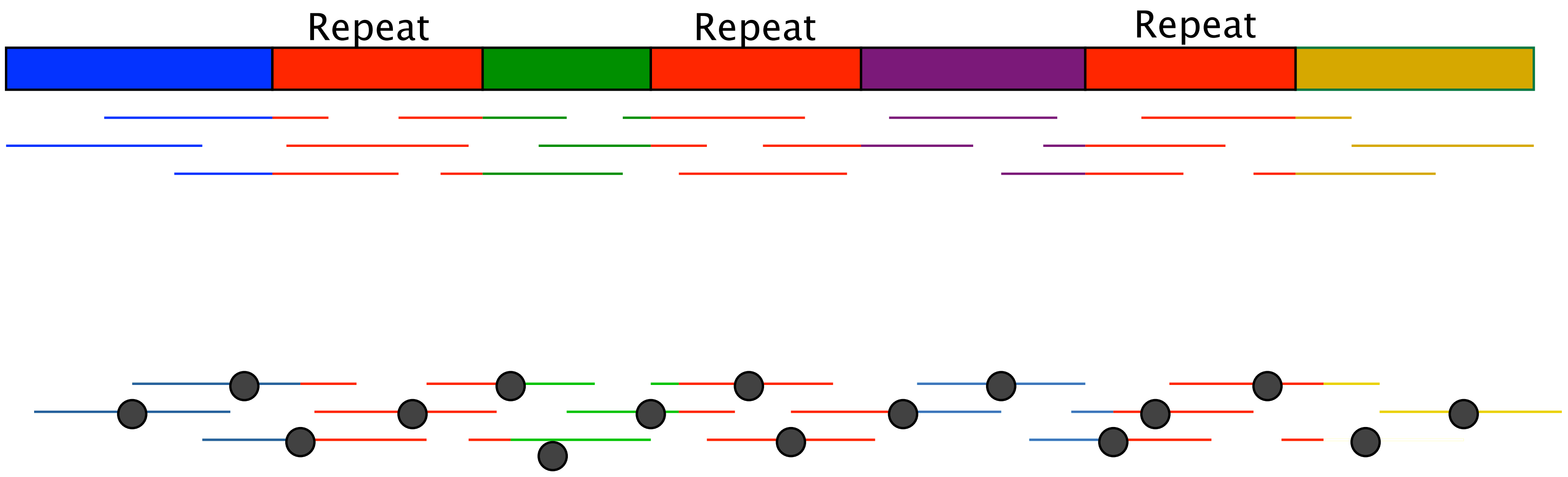
Overlap Graph: Hamiltonian Approach

- Each vertex represents a read from the original sequence.
- Vertices are connected by an edge if they overlap.



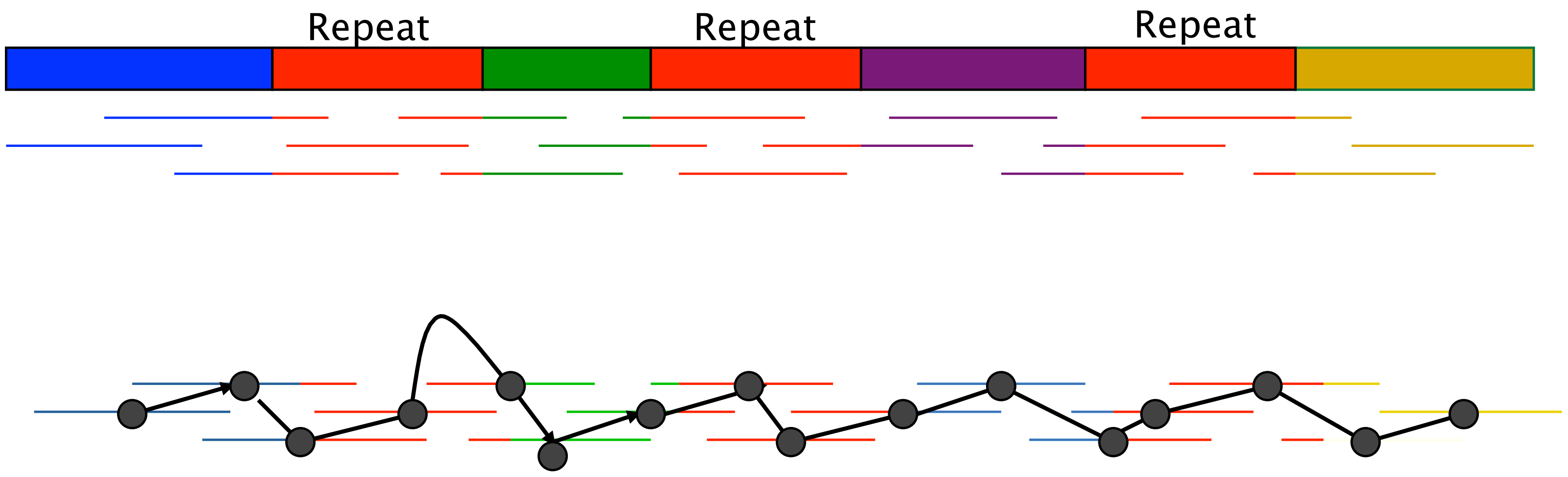
Overlap Graph: Hamiltonian Approach

- Each vertex represents a read from the original sequence.
- Vertices are connected by an edge if they overlap.



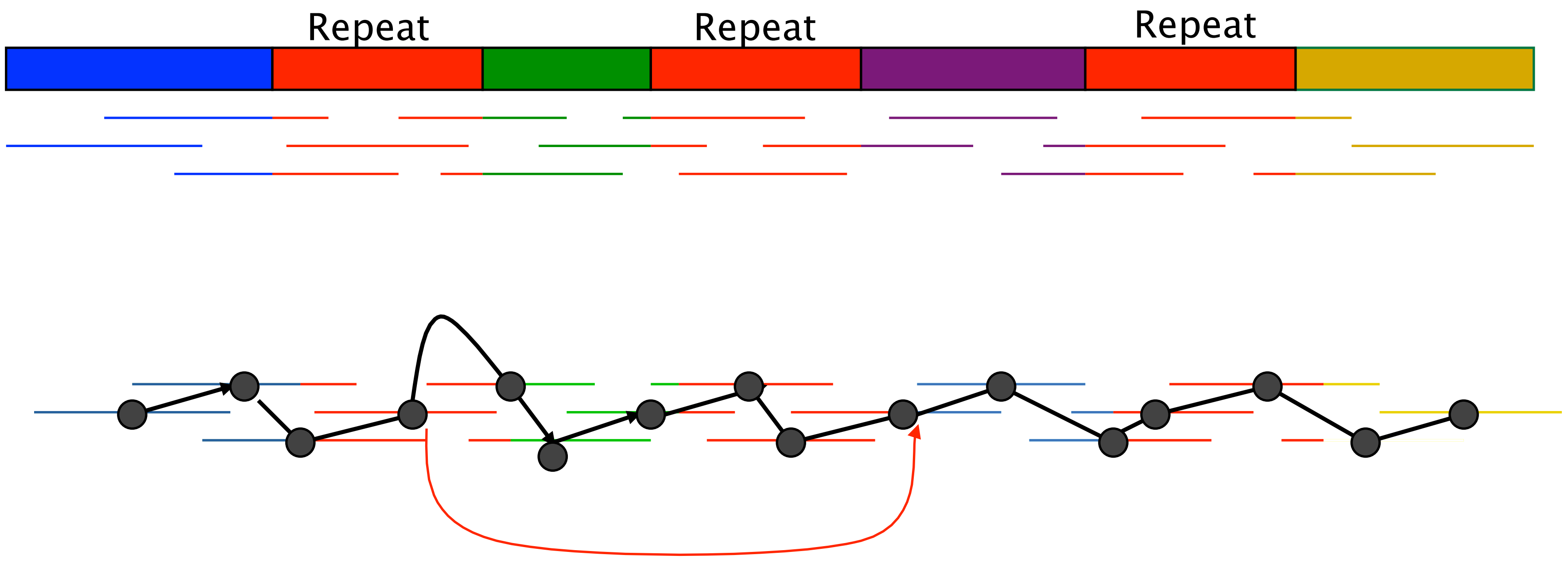
Overlap Graph: Hamiltonian Approach

- Each vertex represents a read from the original sequence.
- Vertices are connected by an edge if they overlap.



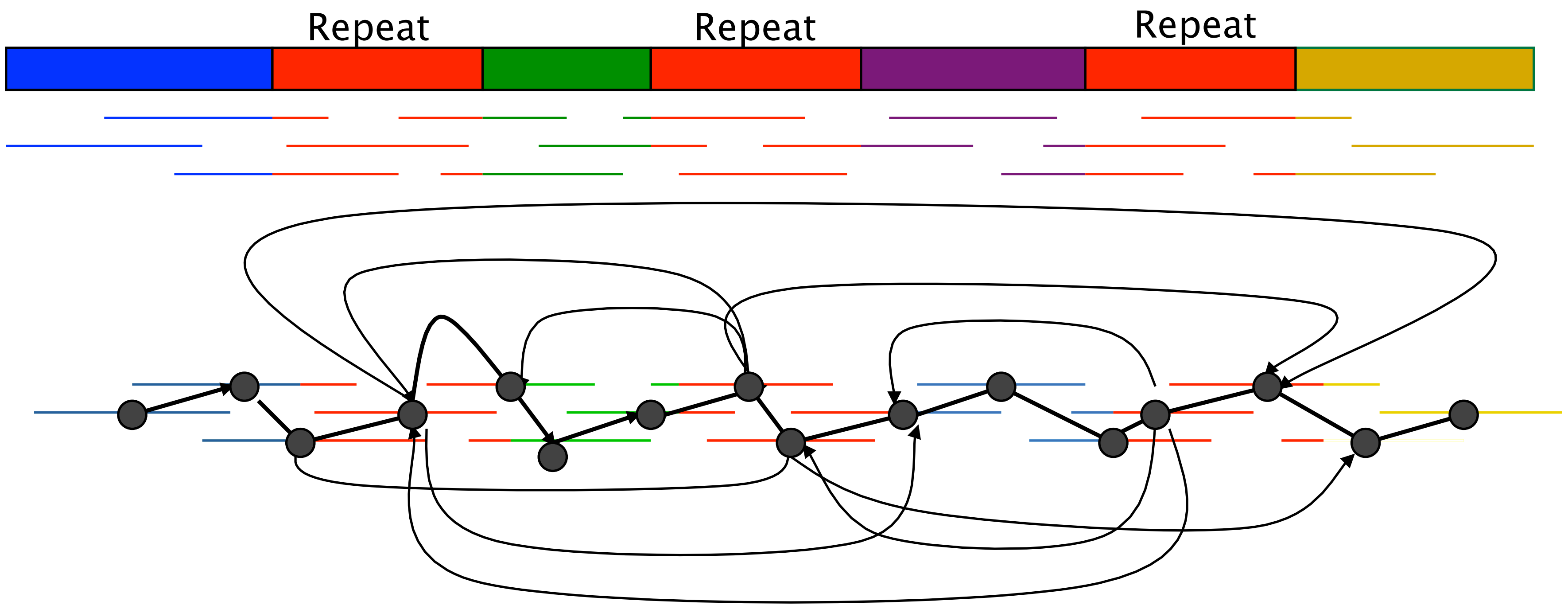
Overlap Graph: Hamiltonian Approach

- Each vertex represents a read from the original sequence.
- Vertices are connected by an edge if they overlap.



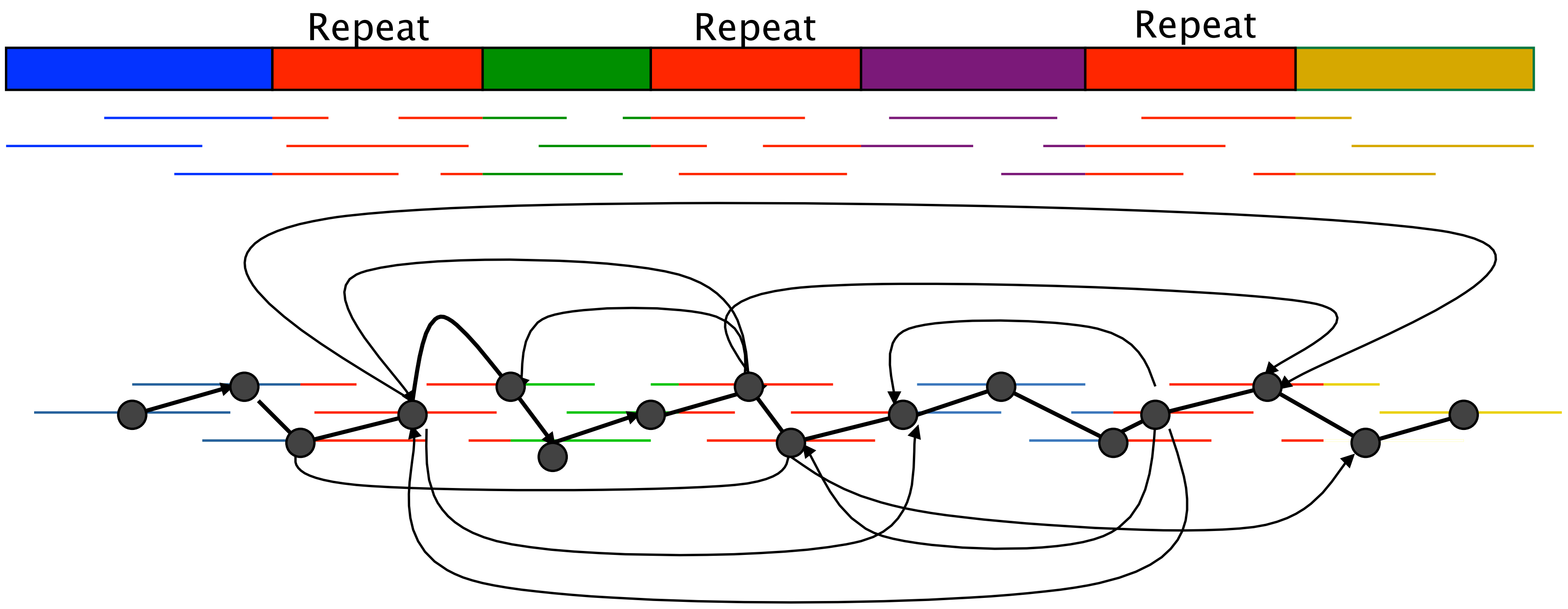
Overlap Graph: Hamiltonian Approach

- Each vertex represents a read from the original sequence.
- Vertices are connected by an edge if they overlap.



Overlap Graph: Hamiltonian Approach

- Each vertex represents a read from the original sequence.
- Vertices are connected by an edge if they overlap.

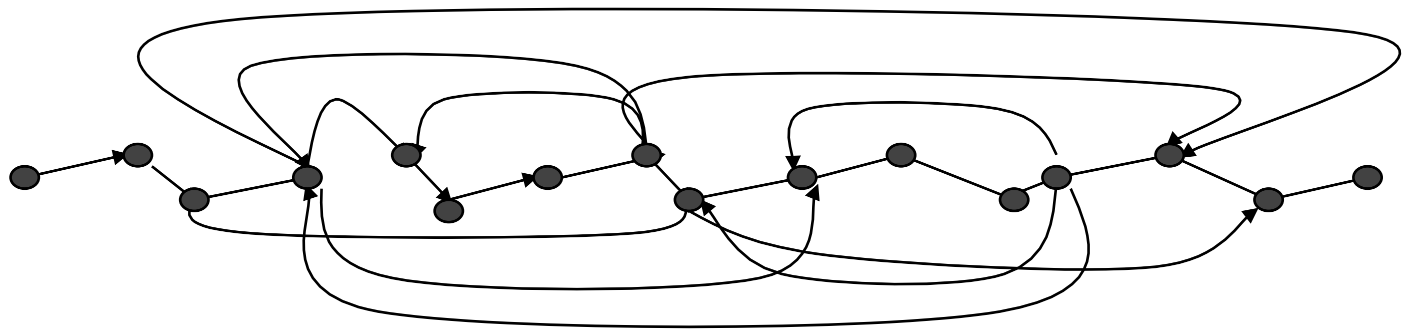


- A *Hamiltonian path* in this graph provides a candidate assembly.

Overlap Graph: Hamiltonian Approach

- So finding an alignment corresponds to finding a Hamiltonian path in the overlap graph.

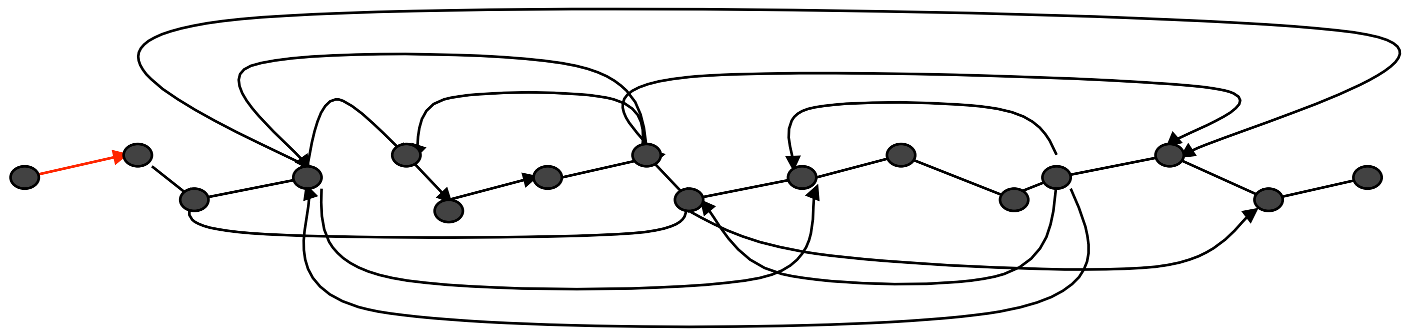
- Recall that the Hamiltonian path/cycle problem is *NP-Complete*: no efficient algorithms are known.



Overlap Graph: Hamiltonian Approach

- So finding an alignment corresponds to finding a Hamiltonian path in the overlap graph.

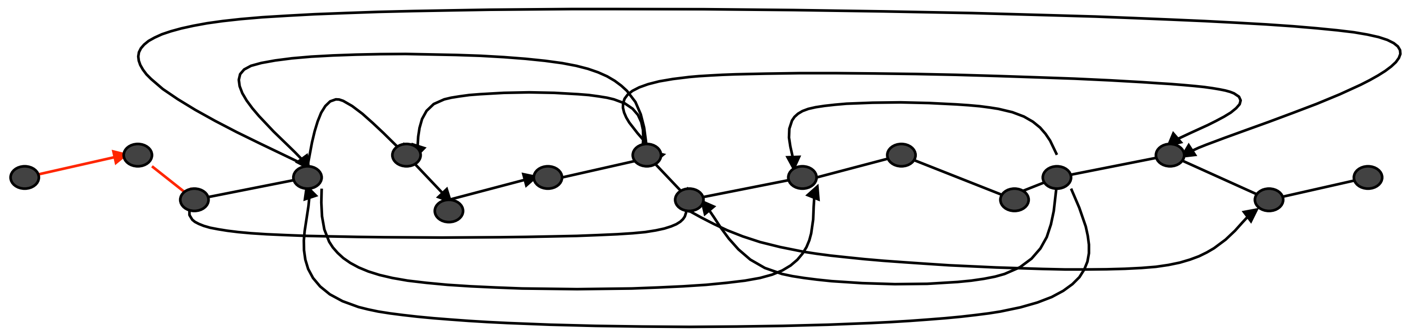
- Recall that the Hamiltonian path/cycle problem is *NP-Complete*: no efficient algorithms are known.



Overlap Graph: Hamiltonian Approach

- So finding an alignment corresponds to finding a Hamiltonian path in the overlap graph.

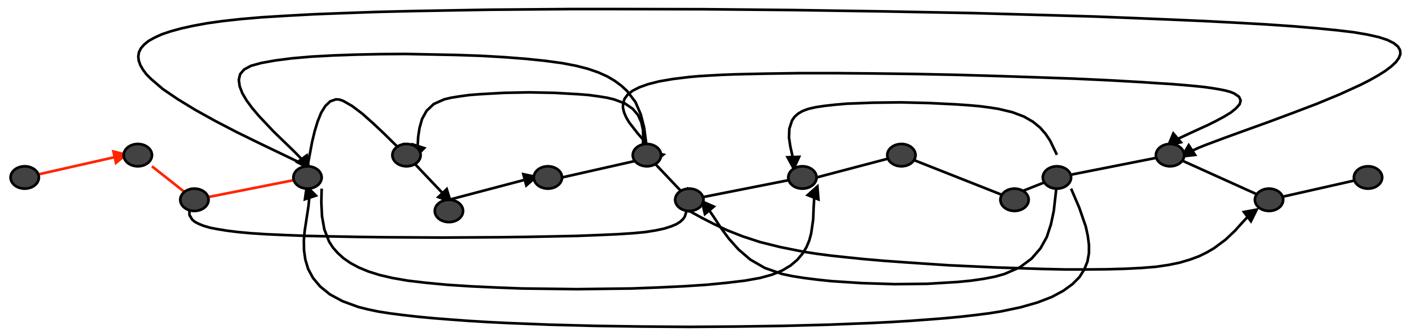
- Recall that the Hamiltonian path/cycle problem is *NP-Complete*: no efficient algorithms are known.



Overlap Graph: Hamiltonian Approach

- So finding an alignment corresponds to finding a Hamiltonian path in the overlap graph.

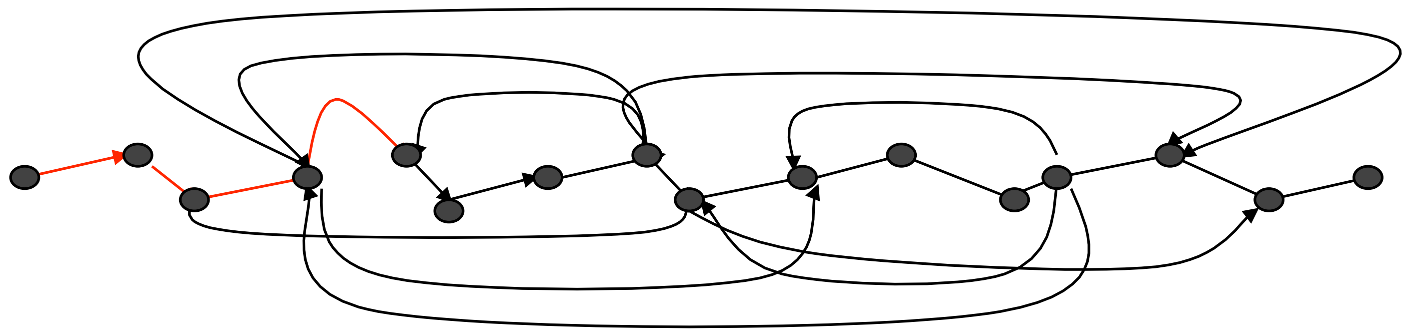
- Recall that the Hamiltonian path/cycle problem is *NP-Complete*: no efficient algorithms are known.



Overlap Graph: Hamiltonian Approach

- So finding an alignment corresponds to finding a Hamiltonian path in the overlap graph.

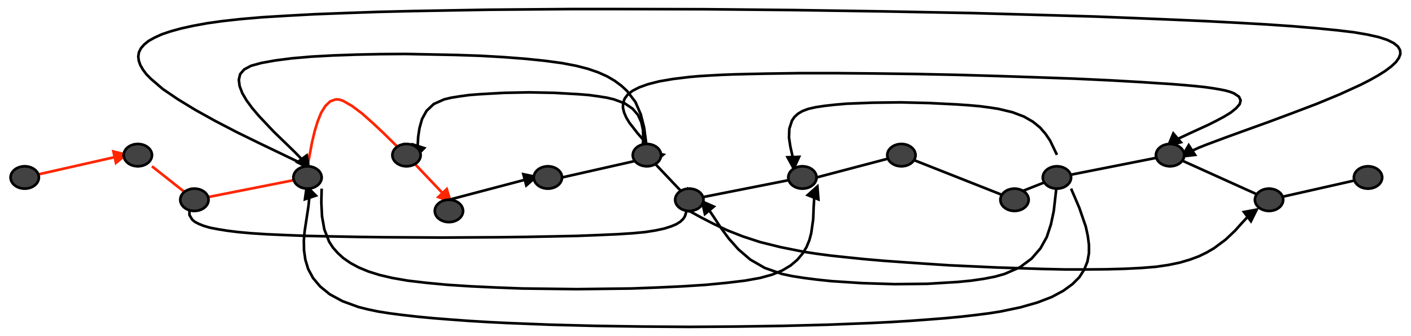
- Recall that the Hamiltonian path/cycle problem is *NP-Complete*: no efficient algorithms are known.



Overlap Graph: Hamiltonian Approach

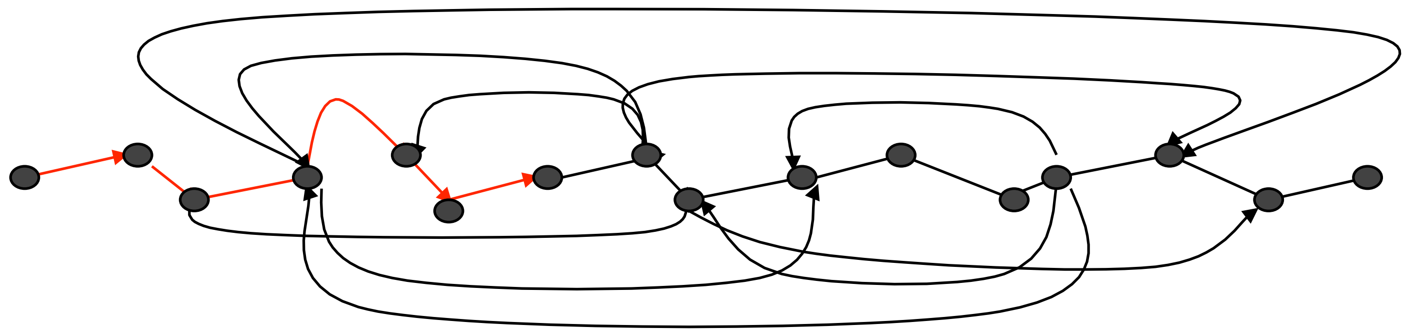
- So finding an alignment corresponds to finding a Hamiltonian path in the overlap graph.

- Recall that the Hamiltonian path/cycle problem is *NP-Complete*: no efficient algorithms are known.



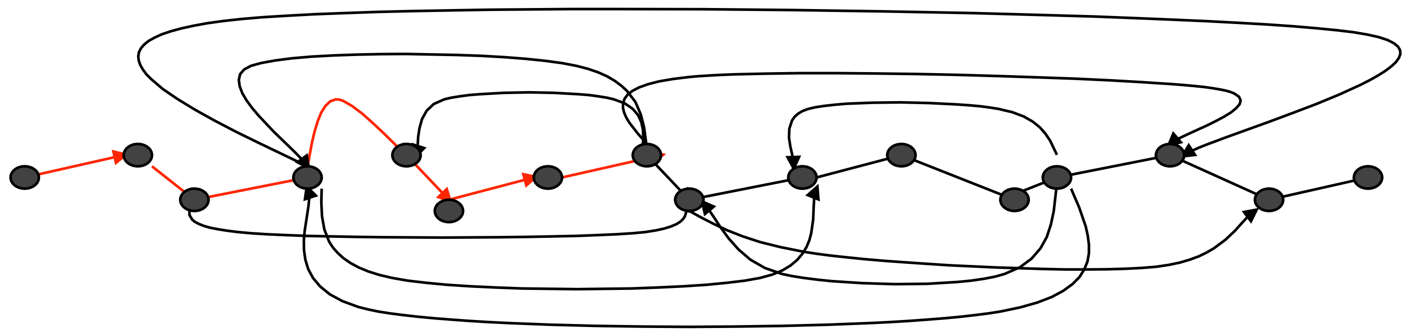
Overlap Graph: Hamiltonian Approach

- So finding an alignment corresponds to finding a Hamiltonian path in the overlap graph.
- Recall that the Hamiltonian path/cycle problem is *NP-Complete*: no efficient algorithms are known.



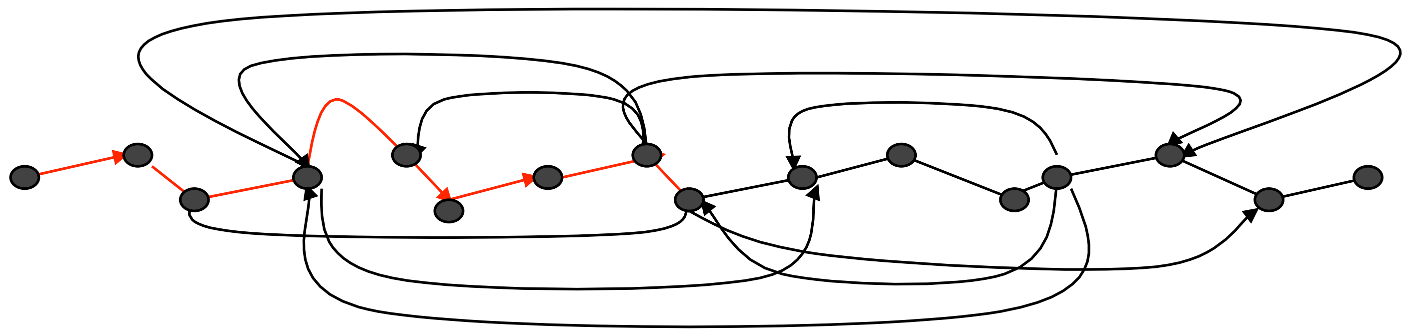
Overlap Graph: Hamiltonian Approach

- So finding an alignment corresponds to finding a Hamiltonian path in the overlap graph.
- Recall that the Hamiltonian path/cycle problem is *NP-Complete*: no efficient algorithms are known.



Overlap Graph: Hamiltonian Approach

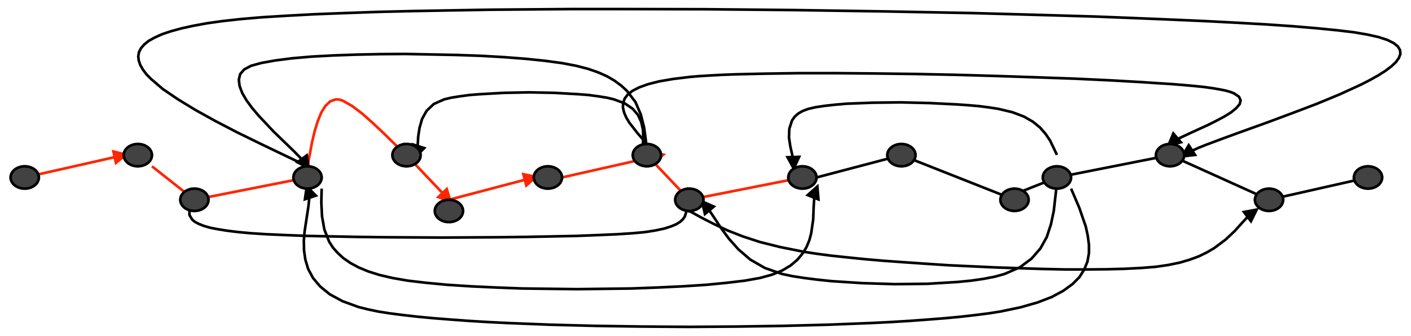
- So finding an alignment corresponds to finding a Hamiltonian path in the overlap graph.
- Recall that the Hamiltonian path/cycle problem is *NP-Complete*: no efficient algorithms are known.



Overlap Graph: Hamiltonian Approach

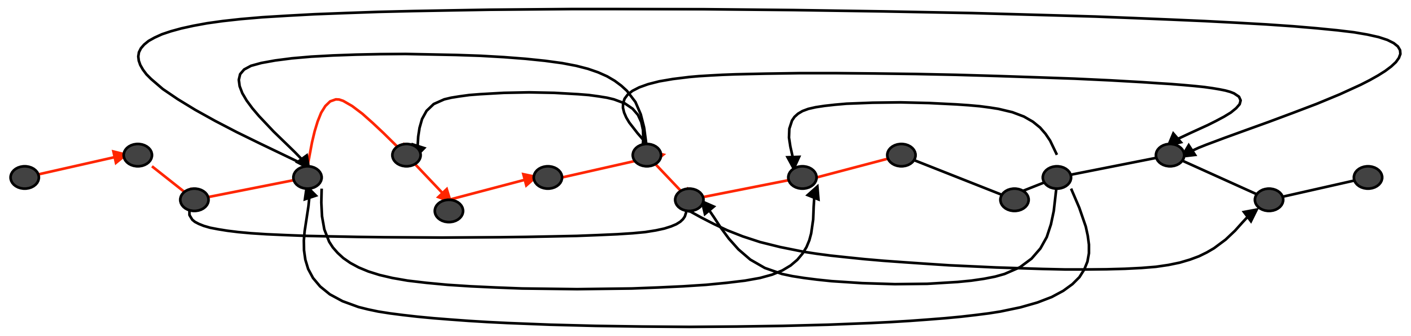
- So finding an alignment corresponds to finding a Hamiltonian path in the overlap graph.

- Recall that the Hamiltonian path/cycle problem is *NP-Complete*: no efficient algorithms are known.



Overlap Graph: Hamiltonian Approach

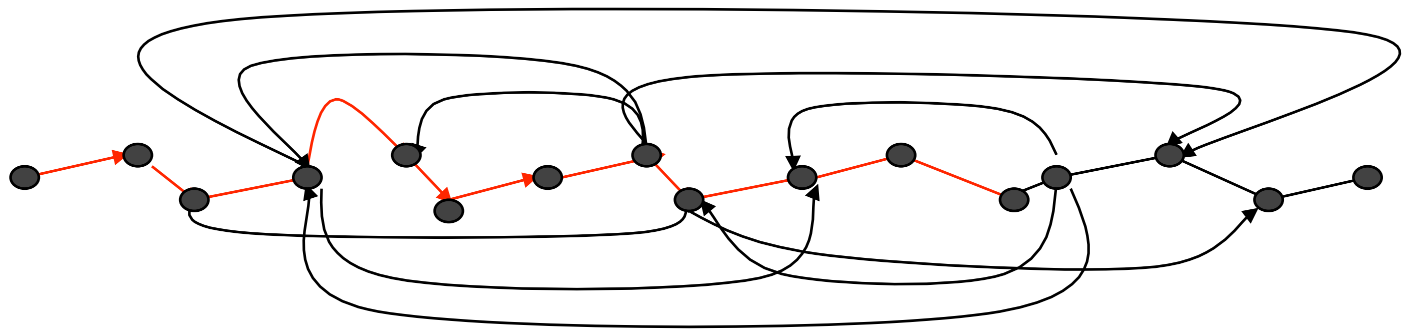
- So finding an alignment corresponds to finding a Hamiltonian path in the overlap graph.
- Recall that the Hamiltonian path/cycle problem is *NP-Complete*: no efficient algorithms are known.



Overlap Graph: Hamiltonian Approach

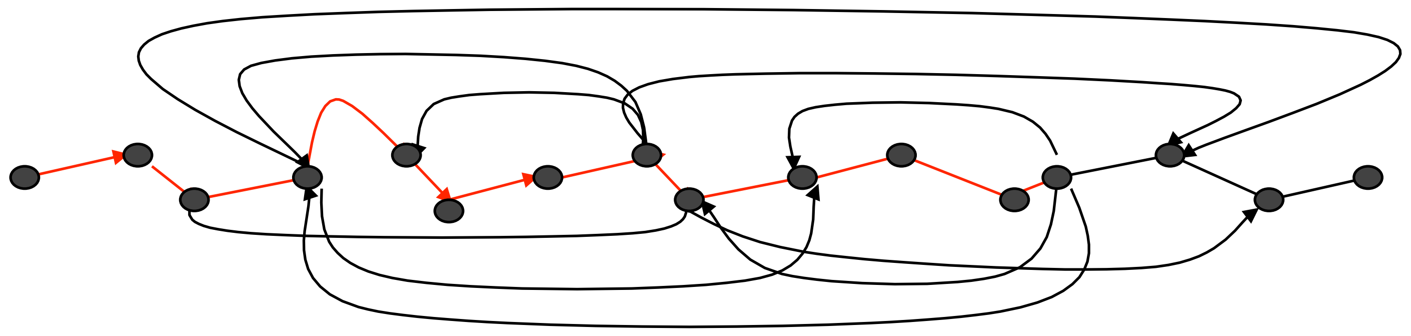
- So finding an alignment corresponds to finding a Hamiltonian path in the overlap graph.

- Recall that the Hamiltonian path/cycle problem is *NP-Complete*: no efficient algorithms are known.



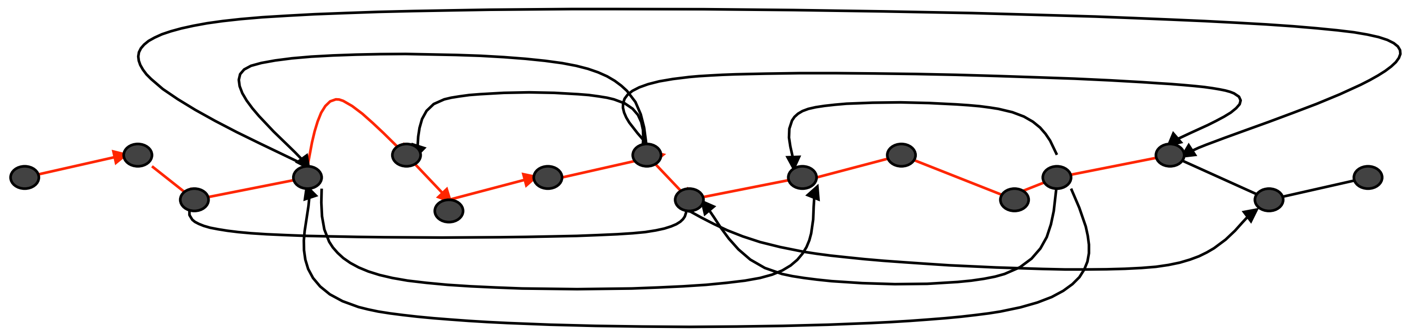
Overlap Graph: Hamiltonian Approach

- So finding an alignment corresponds to finding a Hamiltonian path in the overlap graph.
- Recall that the Hamiltonian path/cycle problem is *NP-Complete*: no efficient algorithms are known.



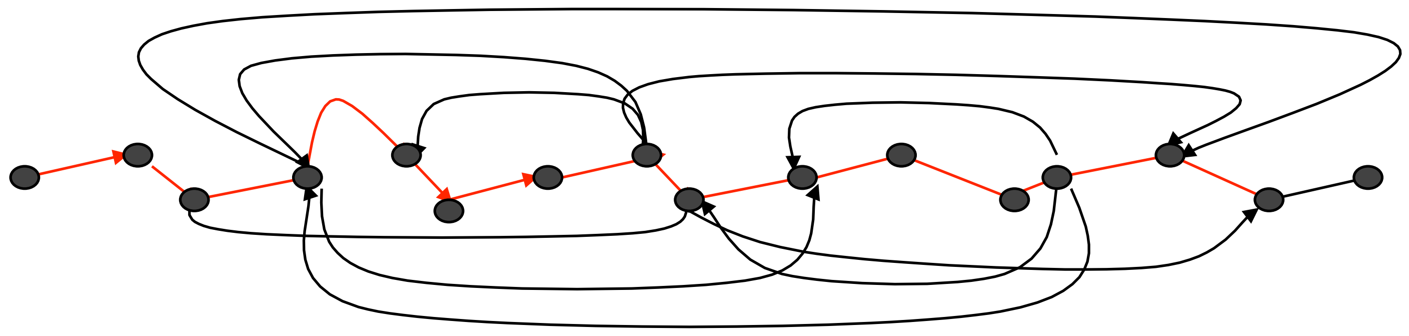
Overlap Graph: Hamiltonian Approach

- So finding an alignment corresponds to finding a Hamiltonian path in the overlap graph.
- Recall that the Hamiltonian path/cycle problem is *NP-Complete*: no efficient algorithms are known.



Overlap Graph: Hamiltonian Approach

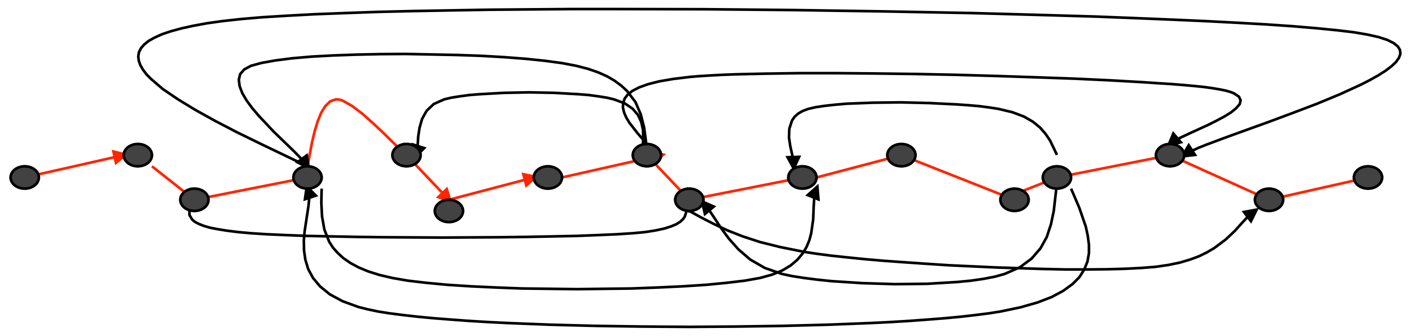
- So finding an alignment corresponds to finding a Hamiltonian path in the overlap graph.
- Recall that the Hamiltonian path/cycle problem is *NP-Complete*: no efficient algorithms are known.



Overlap Graph: Hamiltonian Approach

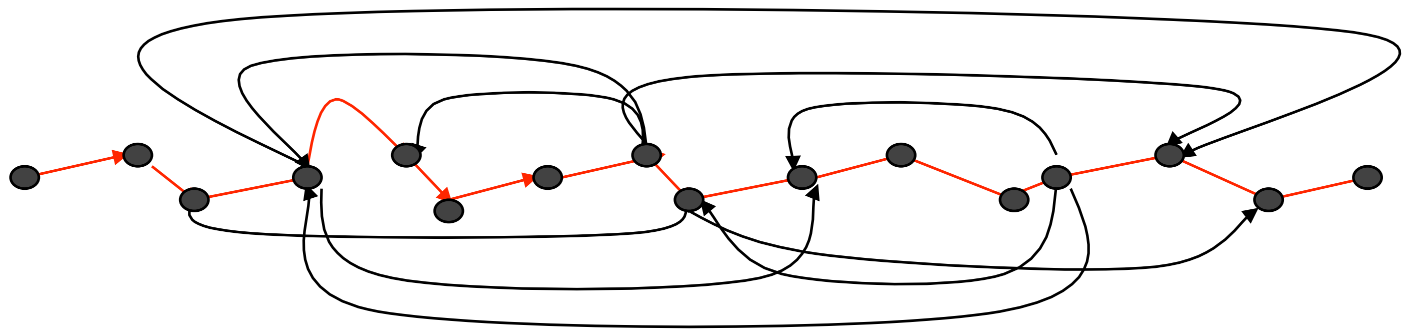
- So finding an alignment corresponds to finding a Hamiltonian path in the overlap graph.

- Recall that the Hamiltonian path/cycle problem is *NP-Complete*: no efficient algorithms are known.



Overlap Graph: Hamiltonian Approach

- So finding an alignment corresponds to finding a Hamiltonian path in the overlap graph.
- Recall that the Hamiltonian path/cycle problem is *NP-Complete*: no efficient algorithms are known.

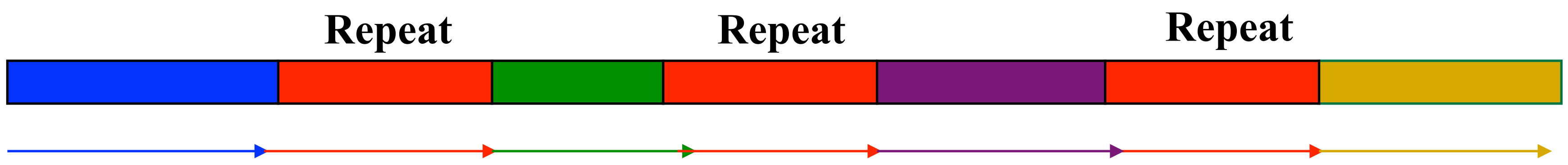


- **Note**: Finding a Hamiltonian path only looks easy because we know the optimal alignment before constructing overlap graph.

EULER Approach to Fragment Assembly

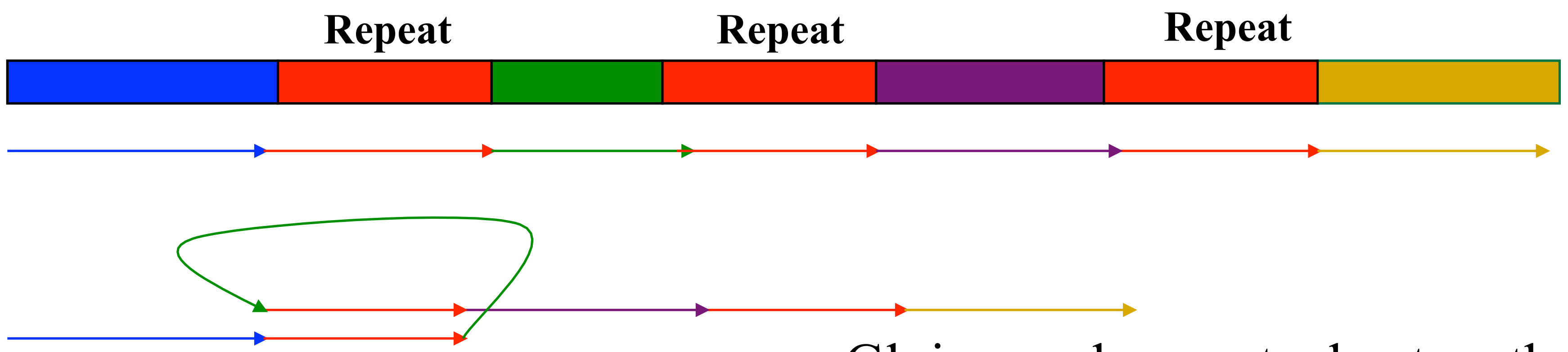
- The “overlap-layout-consensus” technique implicitly solves the Hamiltonian path problem and has a high rate of mis-assembly.
- Can we adapt the Eulerian Path approach borrowed from the SBH problem?
- Fragment assembly without repeat masking can be done in linear time with greater accuracy.

Repeat Graph: Eulerian Approach



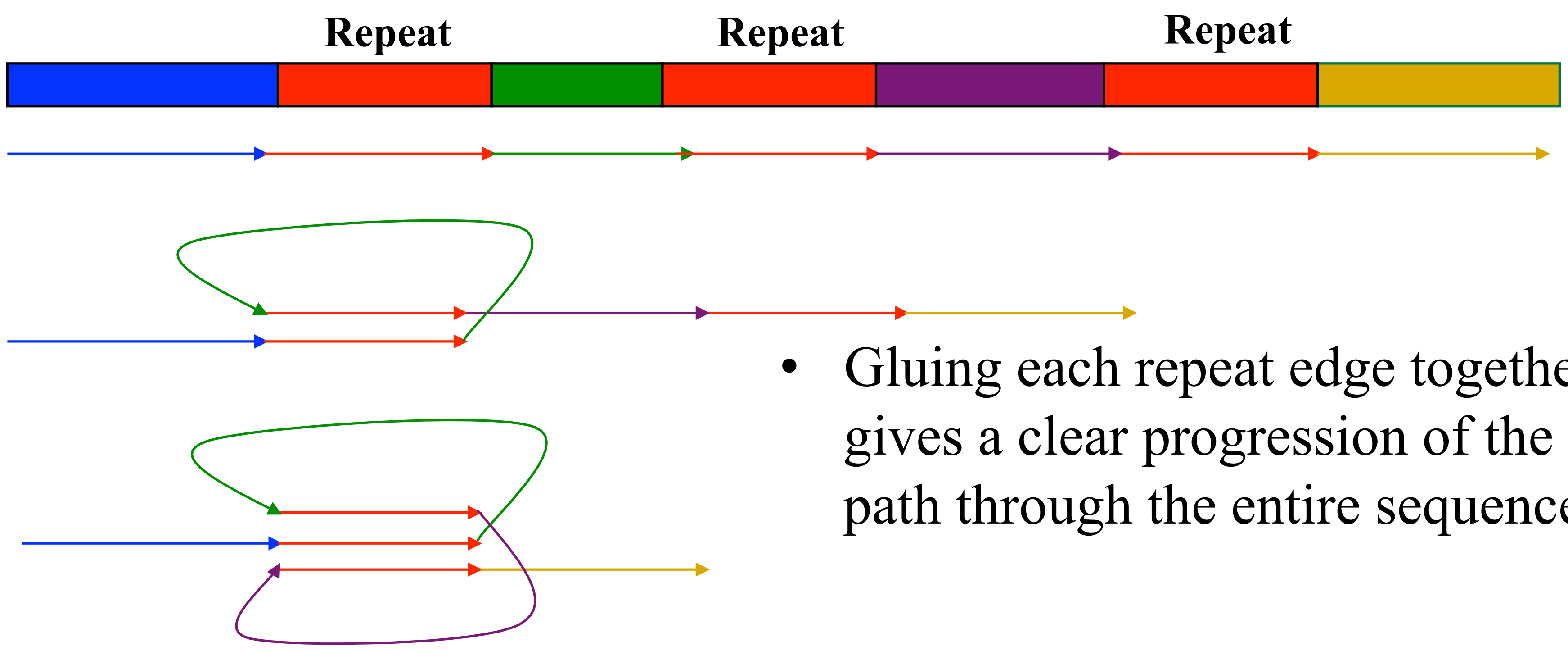
- Gluing each repeat edge together gives a clear progression of the path through the entire sequence.

Repeat Graph: Eulerian Approach



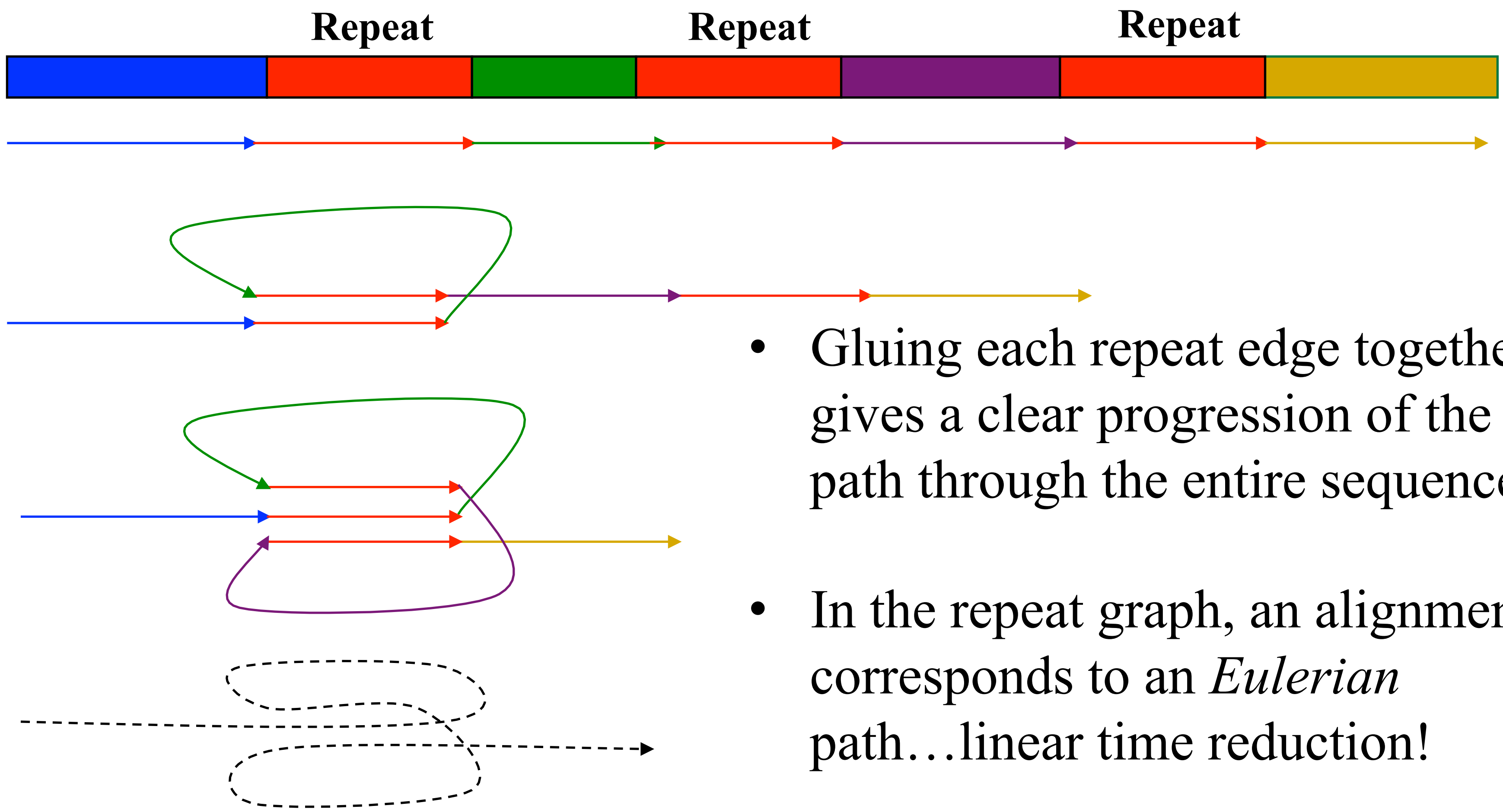
- Gluing each repeat edge together gives a clear progression of the path through the entire sequence.

Repeat Graph: Eulerian Approach



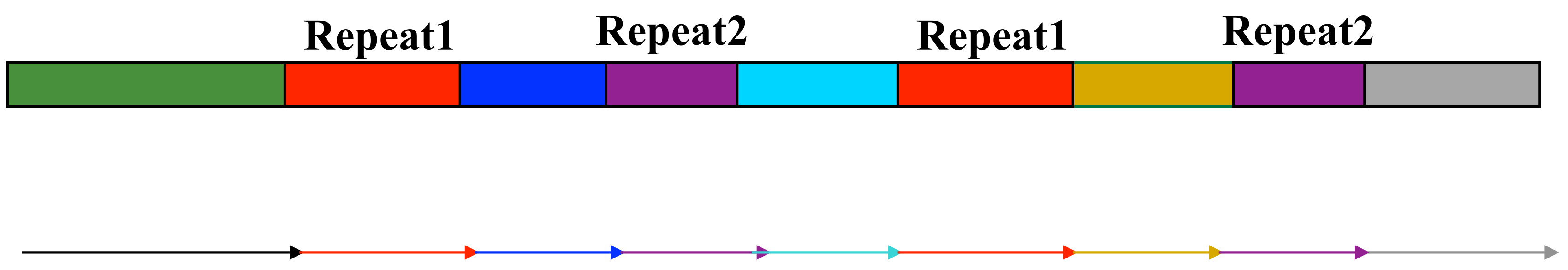
- Gluing each repeat edge together gives a clear progression of the path through the entire sequence.

Repeat Graph: Eulerian Approach



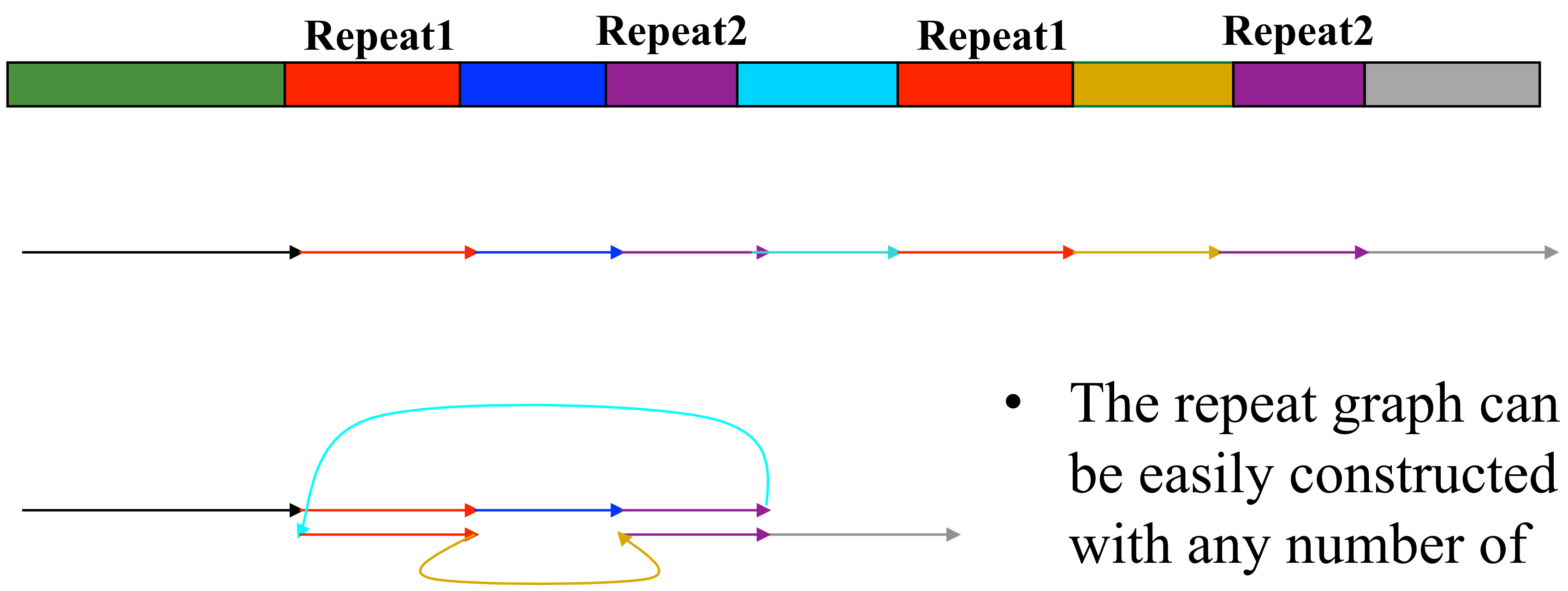
- Gluing each repeat edge together gives a clear progression of the path through the entire sequence.
- In the repeat graph, an alignment corresponds to an *Eulerian* path…linear time reduction!

Repeat Graph: Eulerian Approach



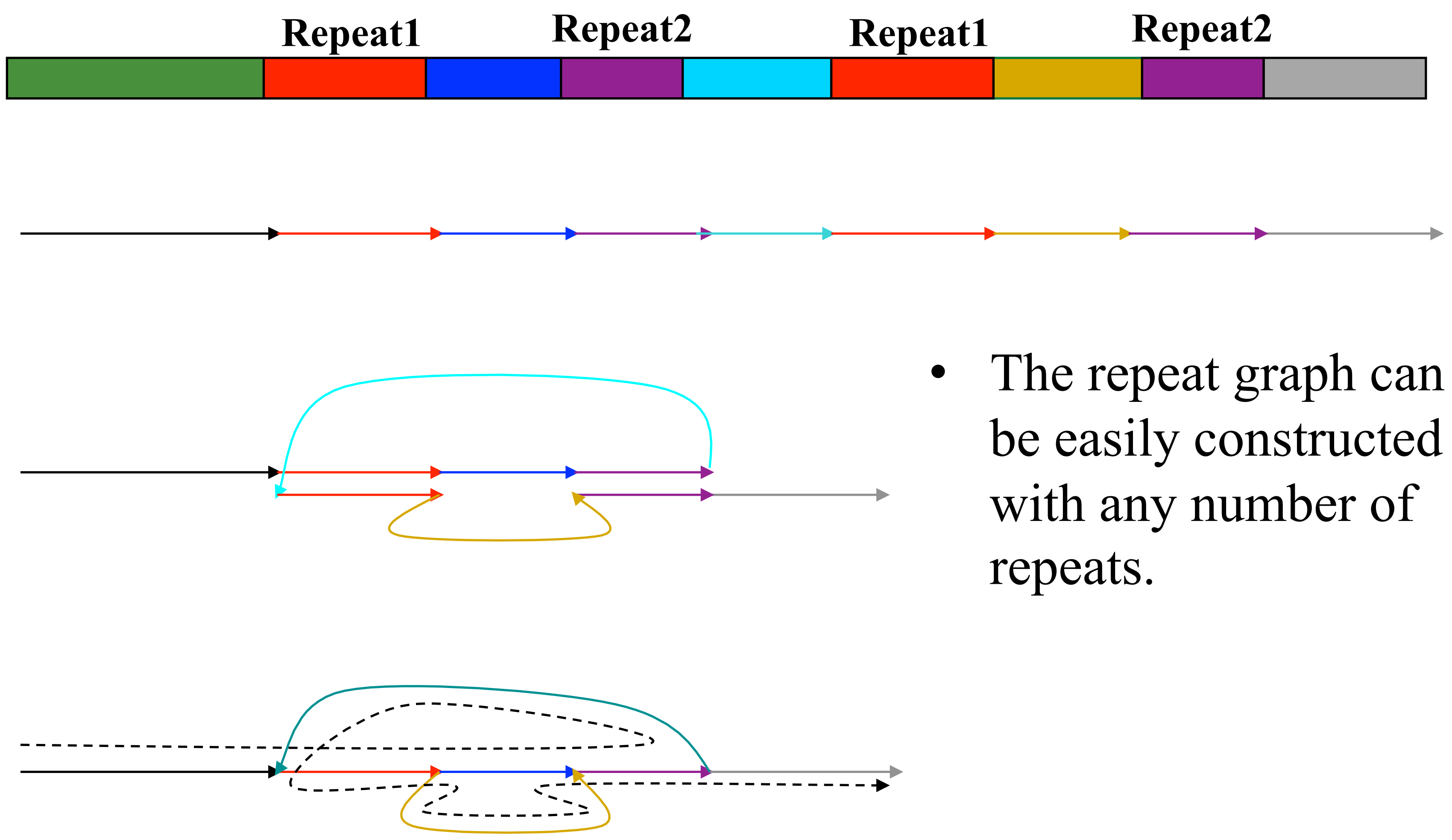
- The repeat graph can be easily constructed with any number of repeats.

Repeat Graph: Eulerian Approach



- The repeat graph can be easily constructed with any number of repeats.

Repeat Graph: Eulerian Approach

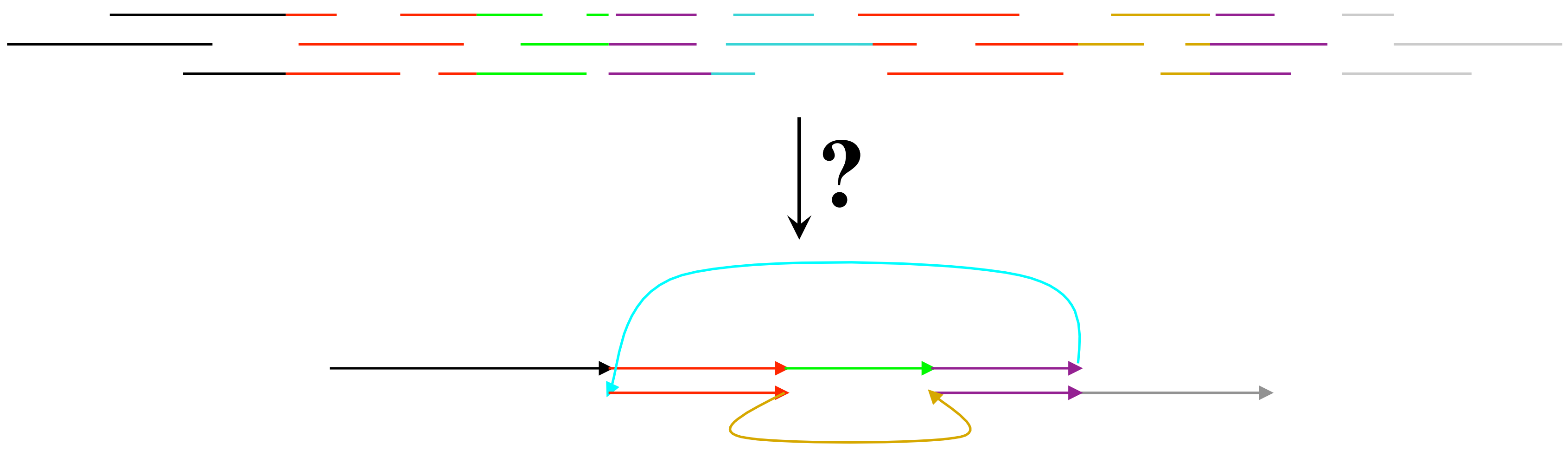


- The repeat graph can be easily constructed with any number of repeats.

Making Repeat Graph From Reads Only

- **Problem**: In previous slides, we have constructed the repeat graph while *already knowing* the genome structure.

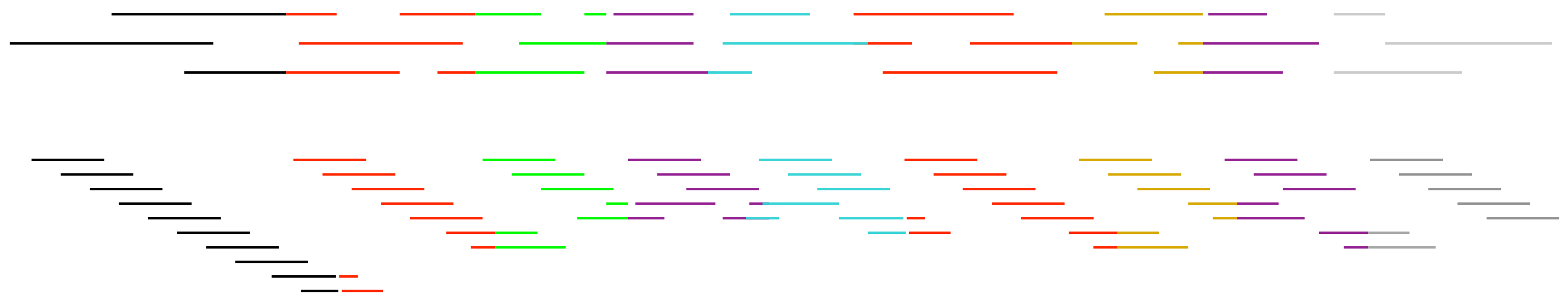
- How do we construct the repeat graph just from fragments?



- **Solution**: Break the reads into smaller pieces.

Repeat Sequences: Emulating a DNA Chip

- A virtual DNA chip allows one to solve the fragment assembly problem using our SBH algorithm.



Construction of Repeat Graph

- **Construction of repeat graph from *k*-mers**: emulates an SBH experiment with a huge (virtual) DNA chip.

- **Breaking reads into *k*-mers**: Transforms sequencing data into virtual DNA chip data.

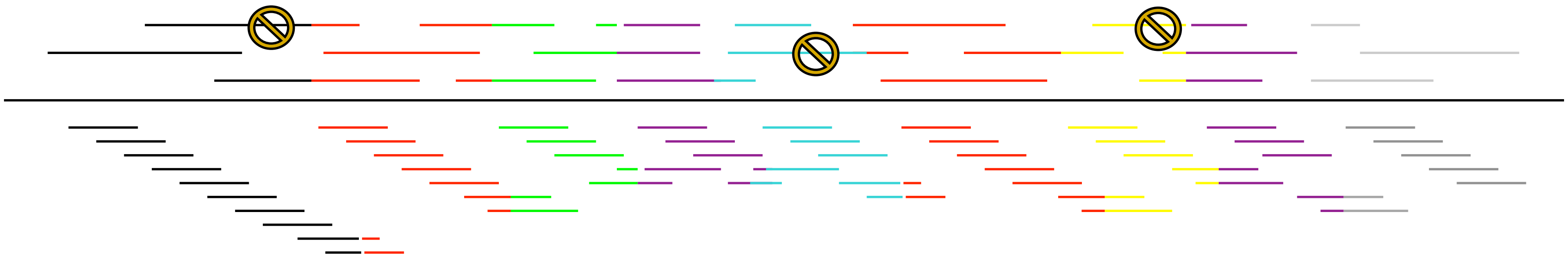
Construction of Repeat Graph

- Error correction in reads: “Consensus first” approach to fragment assembly.
  - Makes reads (almost) error-free BEFORE the assembly even starts.

- Uses reads and mate-pairs to simplify the repeat graph (Eulerian Superpath Problem).

Minimizing Errors

- If an error exists in one of the 20-mer reads, the error will be perpetuated among all of the smaller pieces broken from that read.



- However, that error will not be present in the other instances of the 20-mer read.

- So it is possible to eliminate most point mutation errors before reconstructing the original sequence.

Graph Theory in Bioinformatics

- Graph theory has a wide range of applications throughout bioinformatics, including sequencing, motif finding, protein networks, and many more.

References

- Simons, Robert W. *Advanced Molecular Genetics Course*, UCLA (2002). http://www.mimg.ucla.edu/bobs/C159/Presentations/Benzer.pdf

- Batzoglou, S. *Computational Genomics Course*, Stanford University (2004). http://www.stanford.edu/class/cs262/handouts.html