Lecture 13: Graph Algorithms

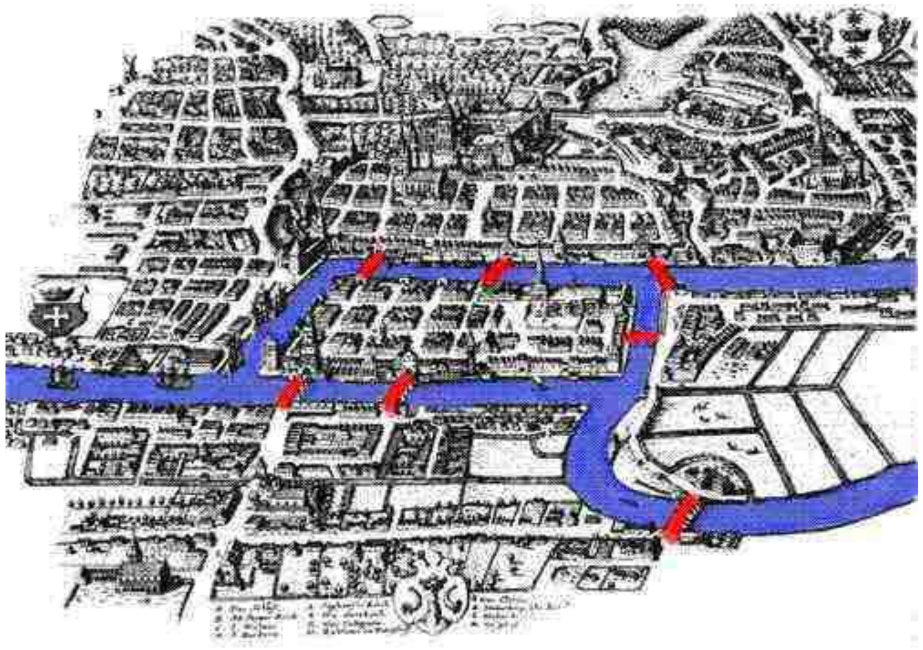
Study Chapter 8.1 – 8.8

Midterm a week from today, March 5

It will cover through Lecture 12, Chapter 7

The Bridge Obsession Problem

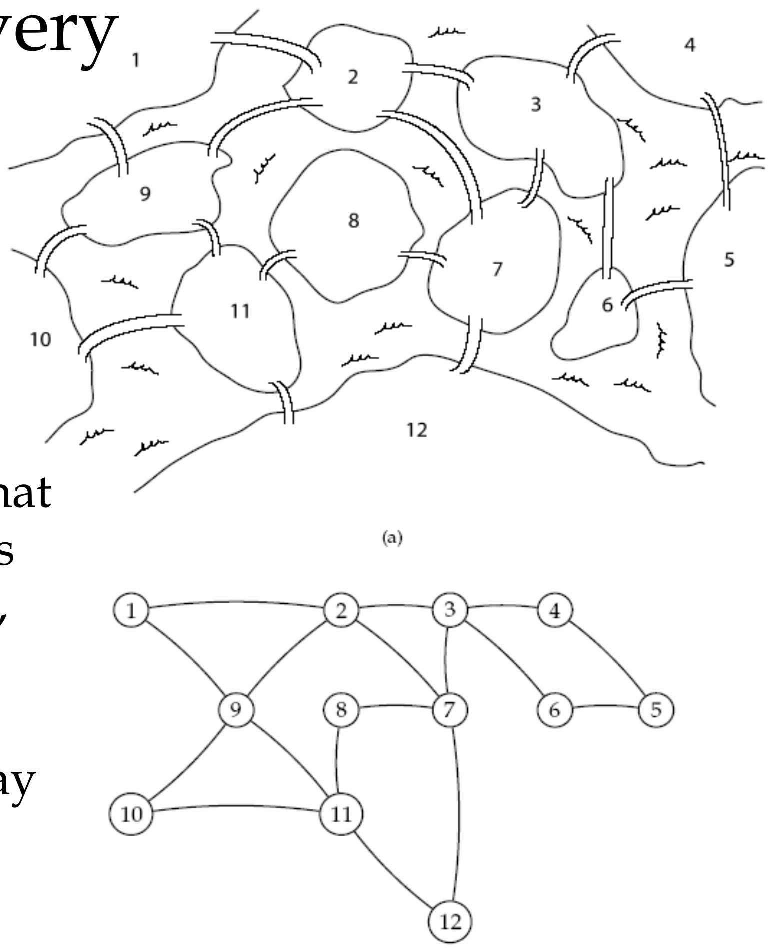
Find a tour crossing every bridge just once
Leonhard Euler, 1735



Bridges of Königsberg

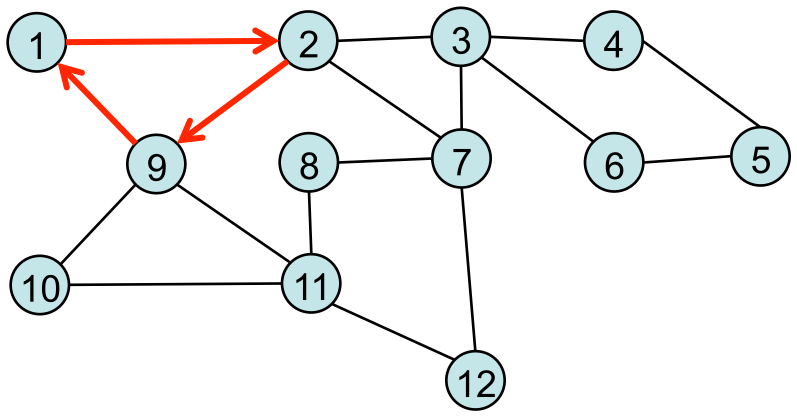
Eulerian Cycle Problem

- Find a cycle that visits every ***edge*** exactly once
- Linear time
  - Starting at any vertex v, and follow a trail of edges until returning to v.
  - As long as there exists a vertex v that belongs to the current tour, but has adjacent edges not part of the tour, start a new trail from v, following unused edges until returning to v, and join the tour formed in this way to the previous tour.



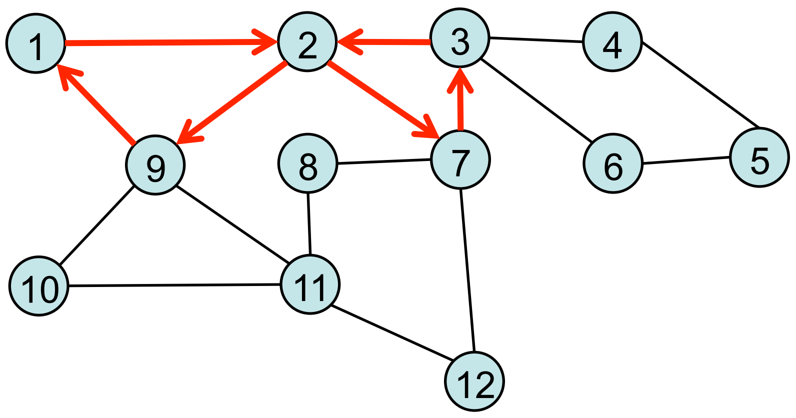
More complicated Königsberg

Finding an Eulerian Cycle



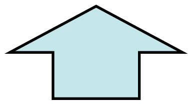
A. 1 → 2 → 9

Finding an Eulerian Cycle



B. 2 → 7 → 3 → 2

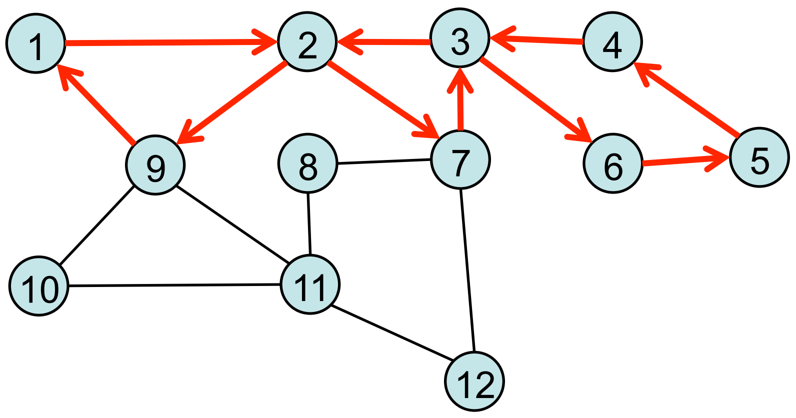
1 → 2 → 9 → 1



2 → 7 → 3 → 2

1 → 2 → 7 → 3 → 2 → 9 → 1

Finding an Eulerian Cycle

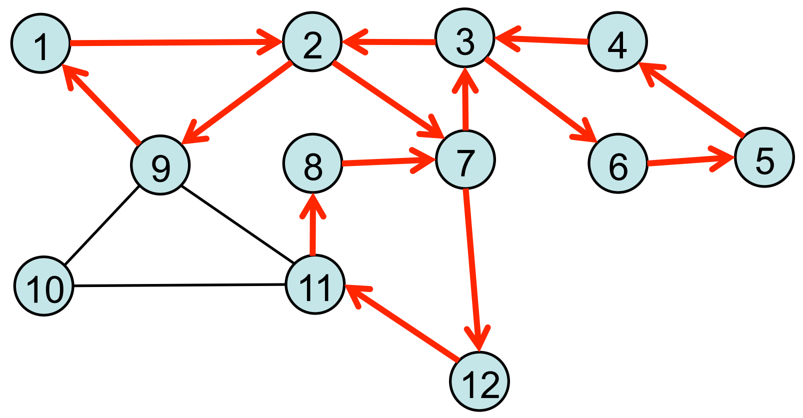


C. 3 → 6 → 5 → 4 → 3

1 → 2 → 7 → 3 → 2 → 9 → 1

1 → 2 → 7 → 3 → 6 → 5 → 4 → 3 → 2 → 9 → 1

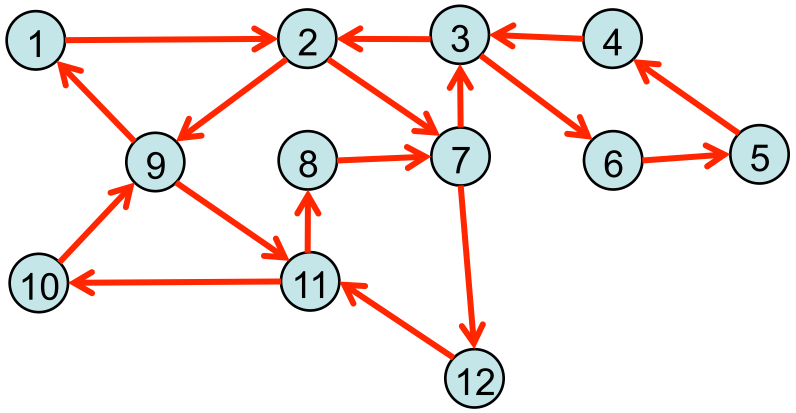
Finding an Eulerian Cycle



D. 7 → 12 → 11 → 8 → 7

1 → 2 → 7 → 3 → 6 → 5 → 4 → 3 → 2 → 9 → 1
1 → 2 → 7 → 12 → 11 → 8 → 7 → 3 → 6 → 5 → 4 → 3 → 2 → 9 → 1

Finding an Eulerian Cycle

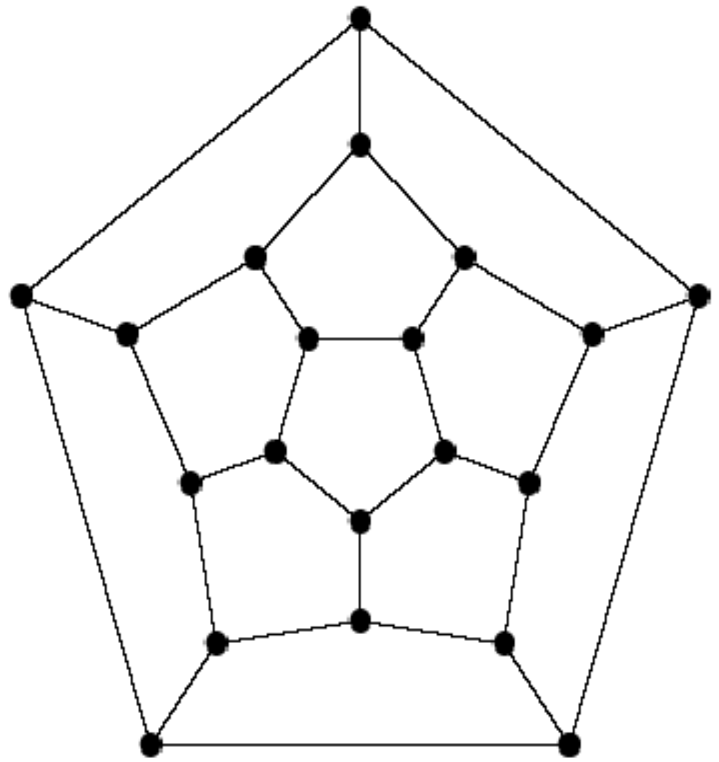


D. 9 → 11 → 10 → 9

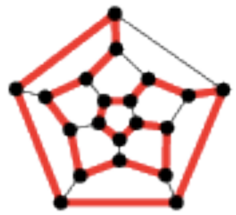
1 → 2 → 7 →12 →11 → 8 → 7 → 3 → 6 → 5 → 4 → 3 → 2 → 9 → 1
1 → 2 → 7 →12 →11 → 8 → 7 → 3 → 6 → 5 → 4 → 3 → 2 →
9 → 11 → 10 → 9 → 1

Hamiltonian Cycle Problem

- Find a cycle that visits every ***vertex*** exactly once
- Deceptively similar to the Eulerian path
- NP-complete

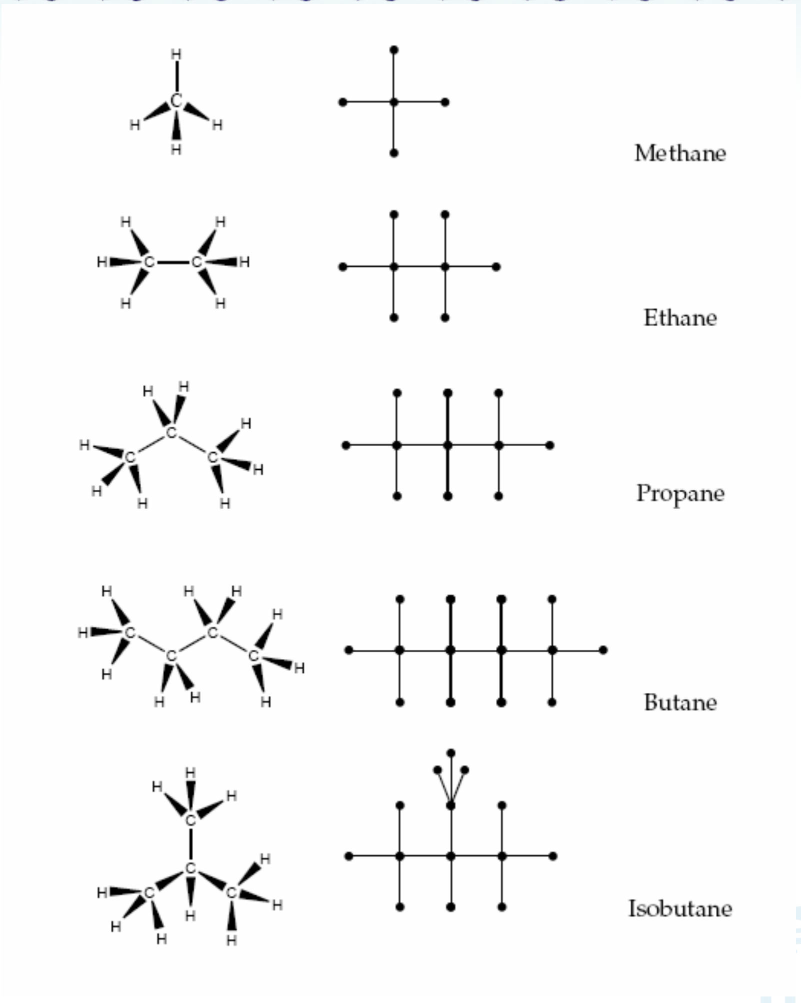


Game invented by Sir William Hamilton in 1857



Mapping Problems to Graphs

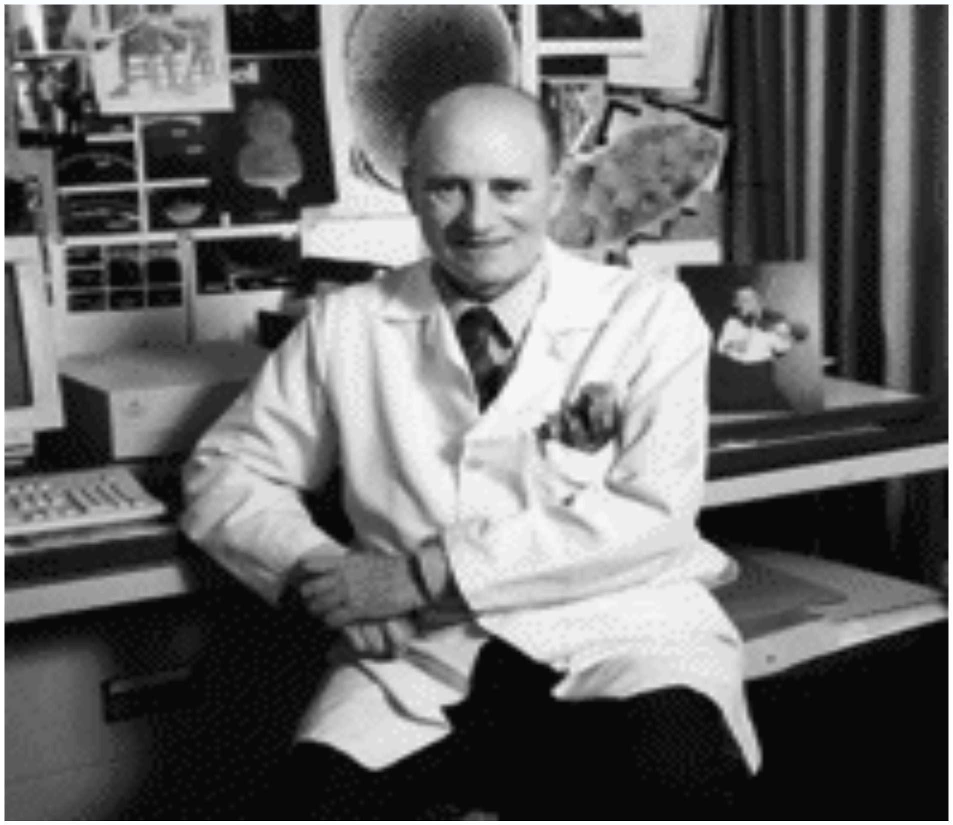
- *Arthur Cayley* studied chemical structures of hydrocarbons in the mid-1800s
- He used **trees** (acyclic connected graphs) to enumerate structural isomers



Beginning of Graph Theory in Biology

Benzer's work

- Developed deletion mapping
- "Proved" linearity of the gene
- Demonstrated internal structure of the gene



Seymour Benzer, 1950s

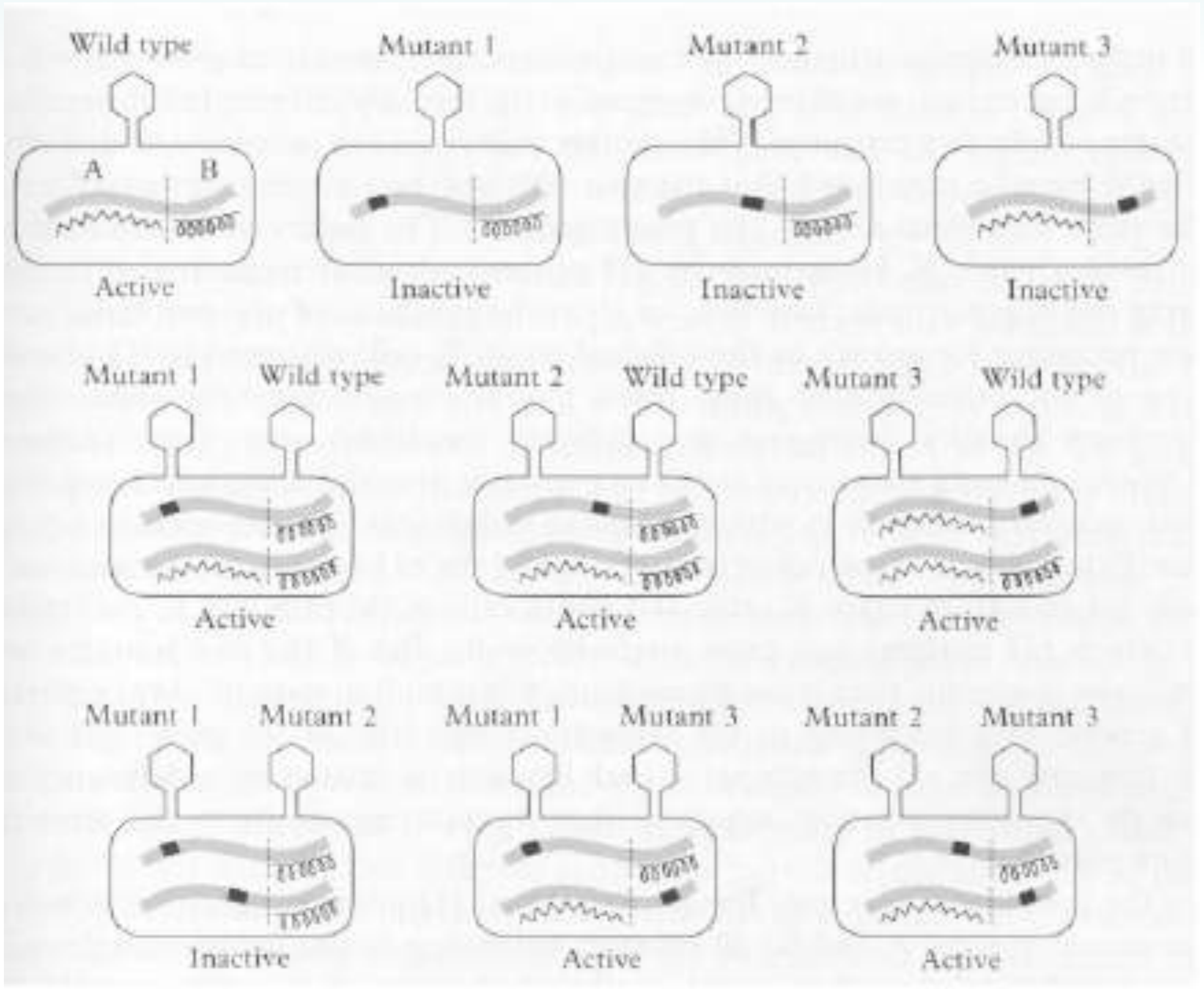
Viruses Attack Bacteria

- Normally bacteriophage T4 kills bacteria
- However if T4 is mutated (e.g., an important subsequence is deleted) it is disabled and looses its ability to kill bacteria
- Suppose the bacteria is infected with two different mutants each of which is disabled – would the bacteria still survive?
- Amazingly, a pair of disabled viruses can kill a bacteria even if each of them is disabled.
- How can it be explained?

Benzer's Experiment

- 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 die

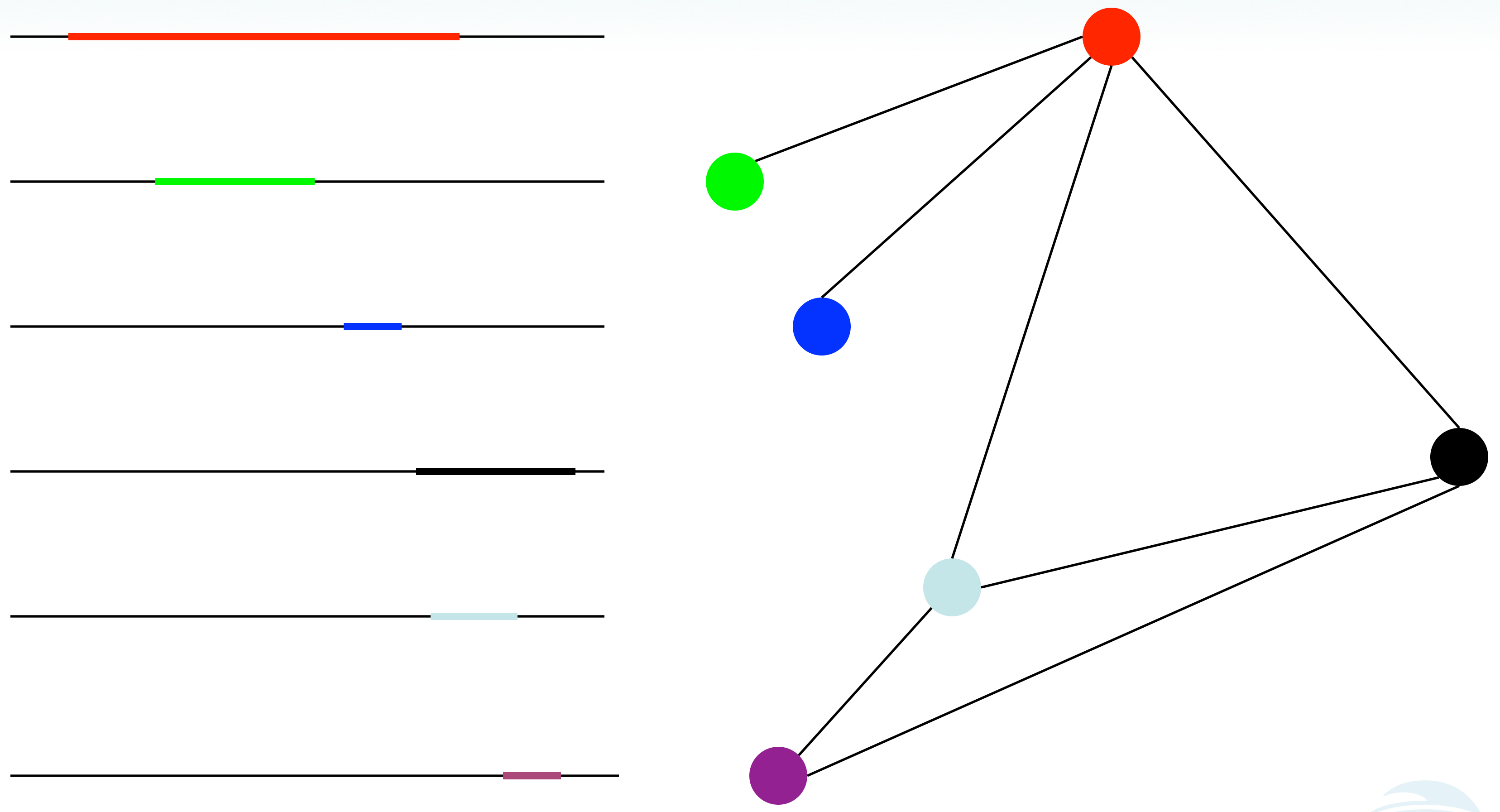
Complementation between pairs of mutant T4 bacteriophages



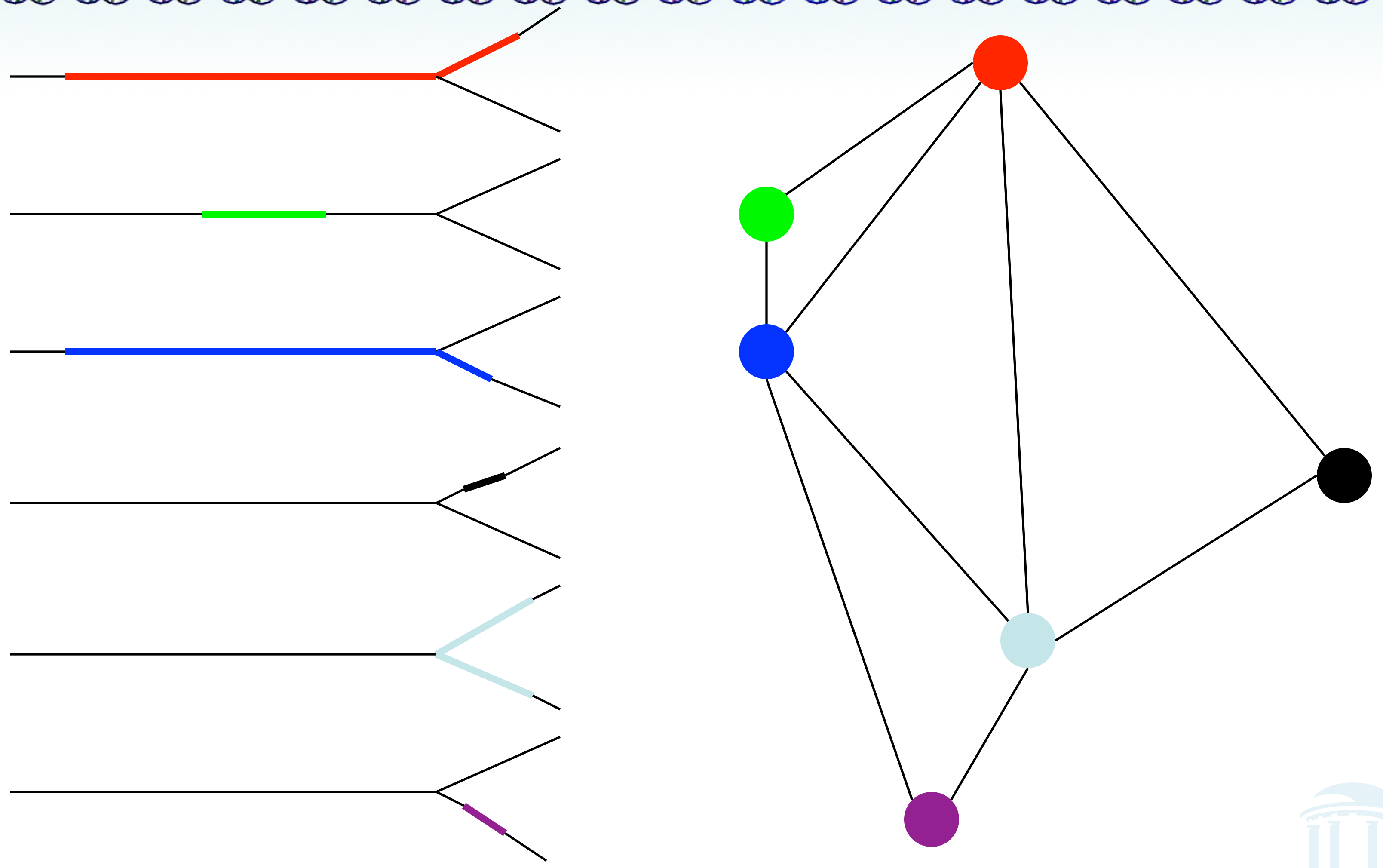
Benzer’s Experiment and Graphs

- Construct an **interval graph**: each T4 mutant is a vertex, place an edge between mutant pairs where bacteria survived (i.e., the deleted intervals in the pair of mutants overlap)
- Interval graph structure reveals whether DNA is linear or branched DNA

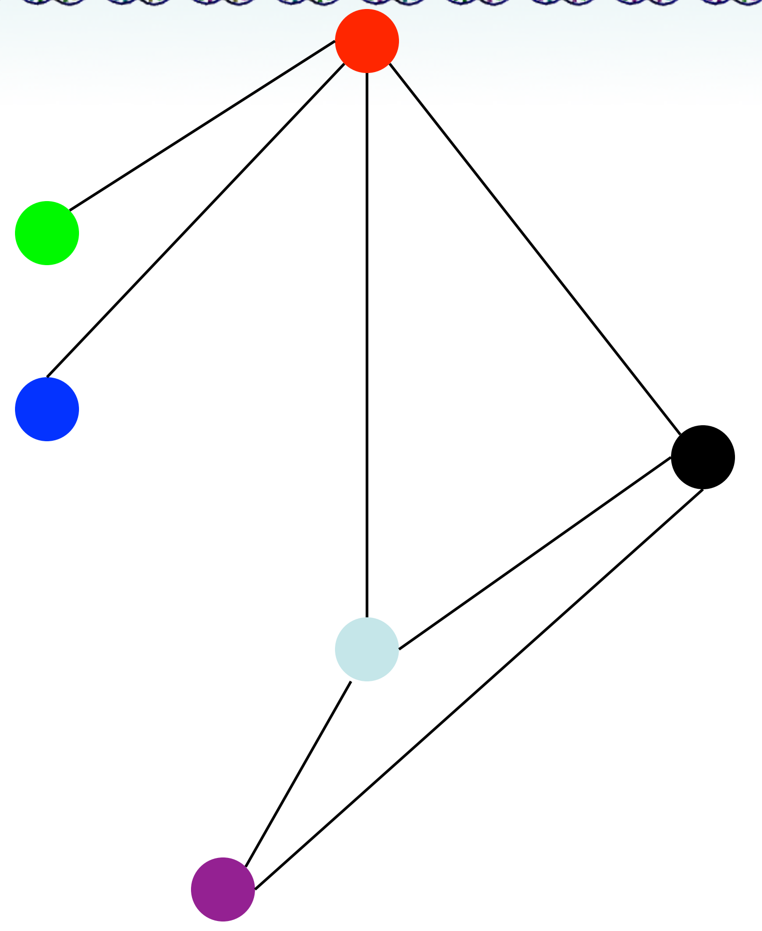
Interval Graph: Linear Genes

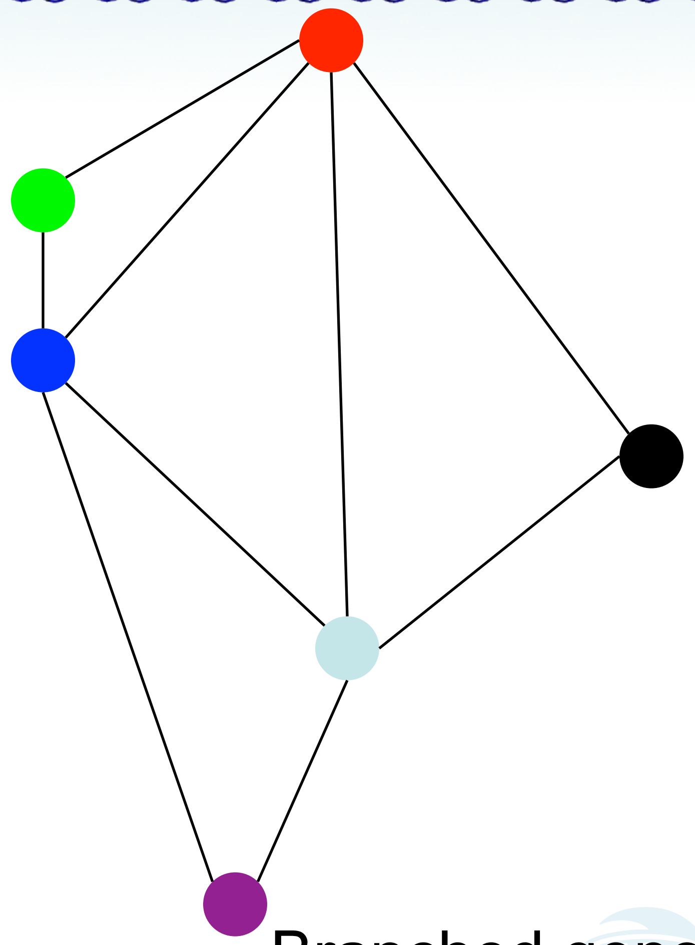


Interval Graph: Branched Genes



Interval Graph: Comparison


Linear genome


Branched genome

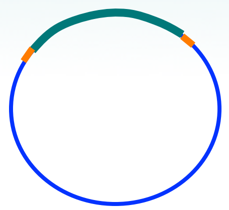
DNA Sequencing: History

Sanger method (1977): labeled ddNTPs terminate DNA copying at random points.

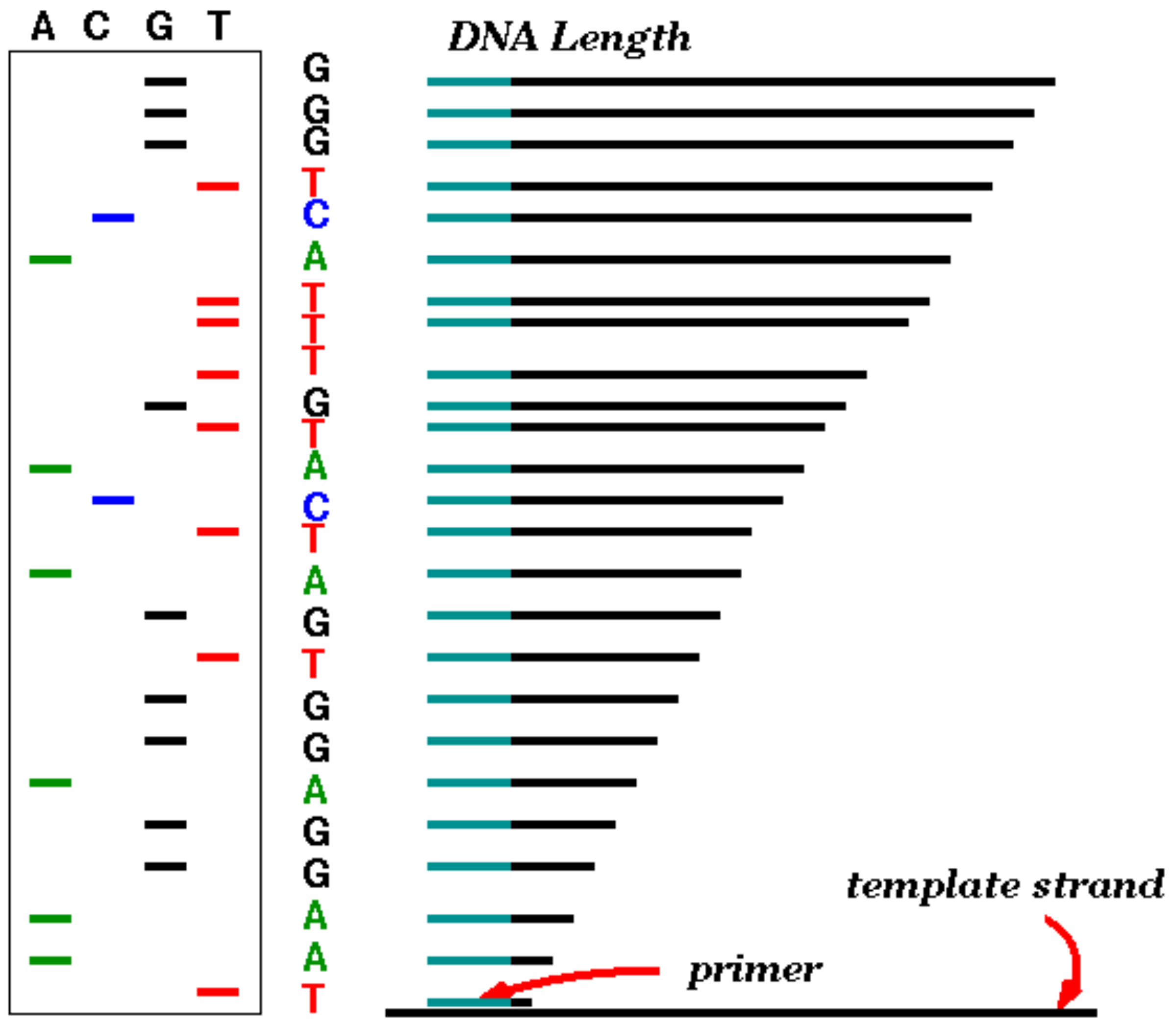
Gilbert method (1977): chemical method to cleave DNA at specific points (G, G+A, T+C, C).

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

Sanger Method: Generating Read

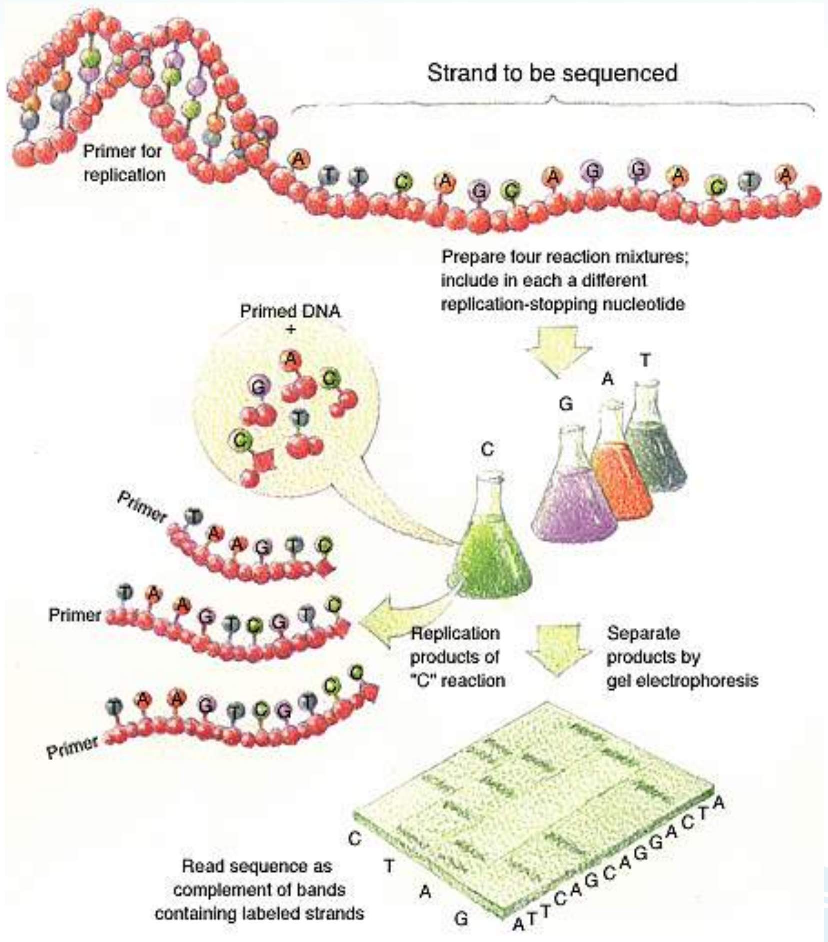


1. Start at primer (restriction site)
2. Grow DNA chain
3. Include ddNTPs
4. Stops reaction at all possible points
5. Separate products by length, using gel electrophoresis



DNA Sequencing

- Shear DNA into millions of small fragments
- Read 500 – 700 nucleotides at a time from the small fragments (Sanger method)


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 +
G A T C
Primer
Primer
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

- **<u>Computational Challenge</u>:** assemble individual short fragments (reads) into a single genomic sequence ("superstring")
- Until late 1990s the shotgun fragment assembly of human genome was viewed as intractable problem

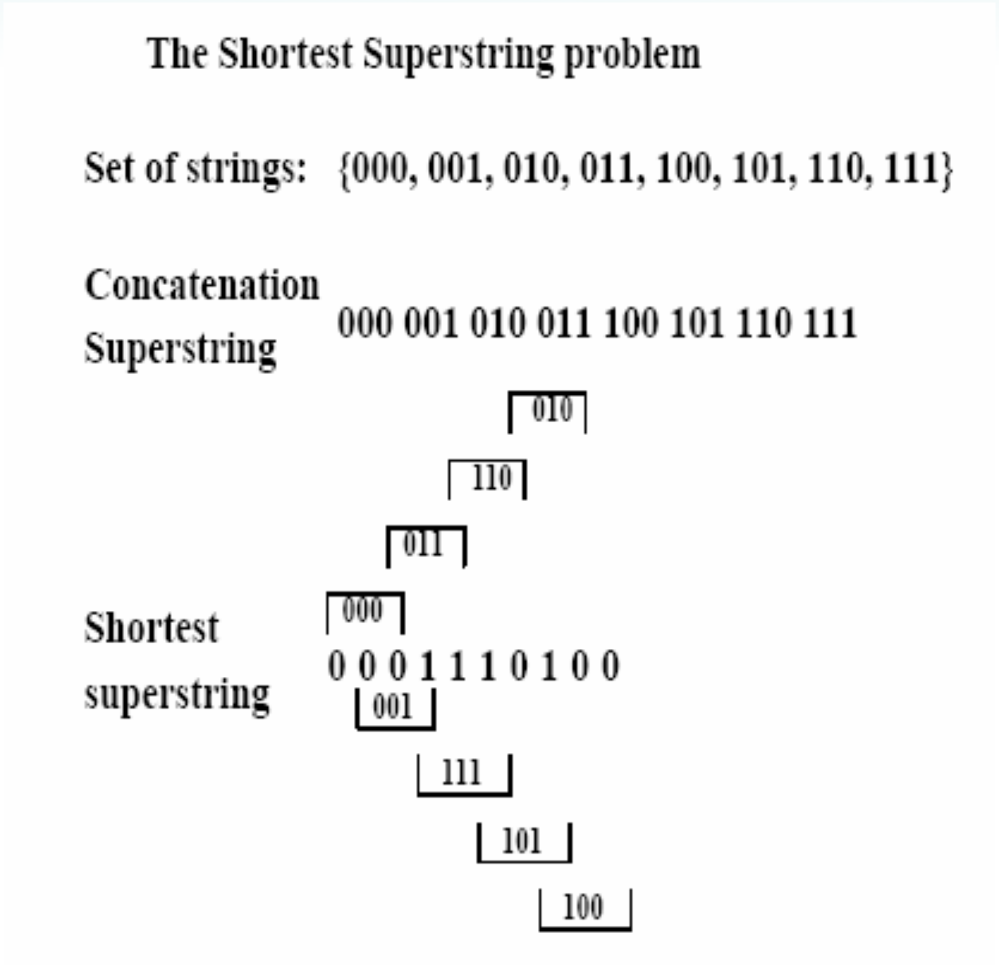
Shortest Superstring Problem

- 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 string s that contains all strings

  $s_1, s_2, \ldots, s_n$ as substrings, such that the length of s is minimized

- **Complexity:** NP – complete
- **Note:** this formulation does not take into account sequencing errors

Shortest Superstring Problem: Example

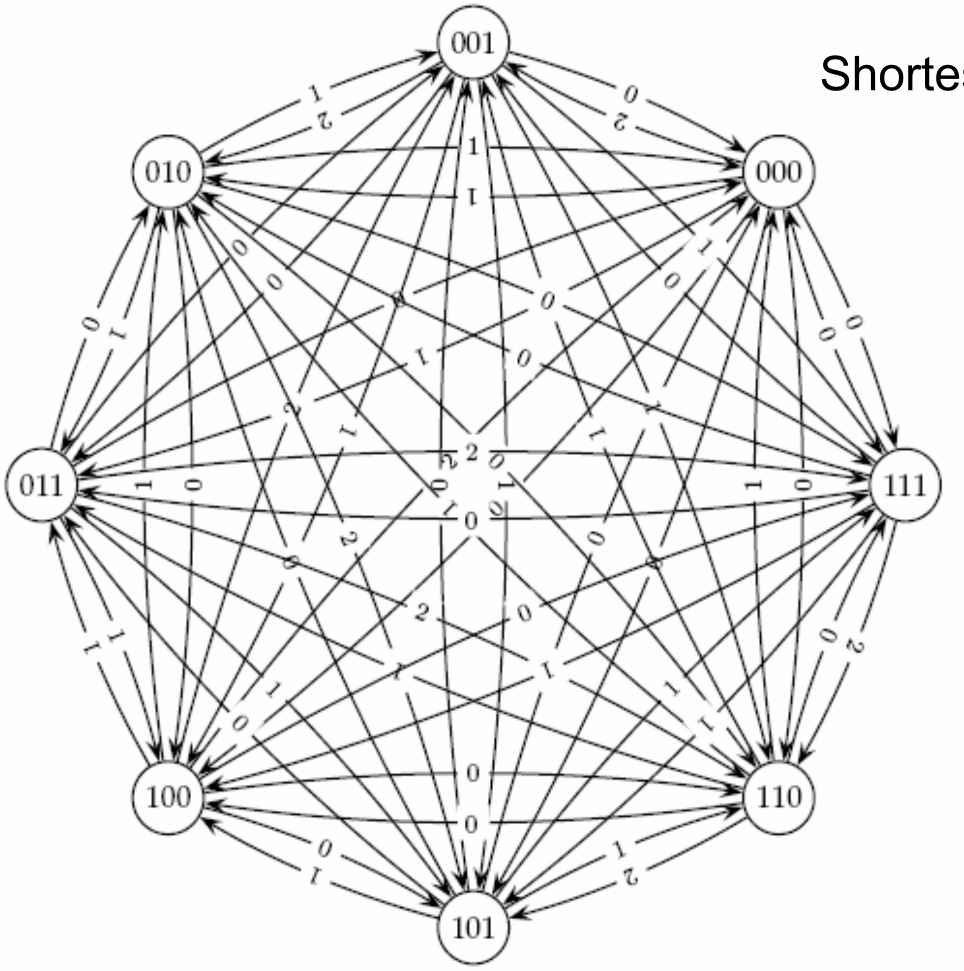

The Shortest Superstring problem

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

Overlap graph of 8 3-bit strings



Shortest Superstring Solution?

Find the heaviest path (by summing edge weights) that includes all vertices.

Sound familiar?

Reducing SSP to TSP

- Define *overlap* (s_i, s_j) as the length of the longest prefix of s_j that matches a suffix of s_i.

aaaggcatcaaatctaaaggcatcaaa

aaaggcatcaaatctaaaggcatcaaa

What is** overlap (s_i, s_j) **for these strings?

Reducing SSP to TSP

- Define *overlap* (s_i, s_j) as the length of the longest prefix of s_j that matches a suffix of s_i.

```
aaaggcatcaaatctaaaggcatcaaa
               aaaggcatcaaatctaaaggcatcaaa
```

***overlap*=12**

Reducing SSP to TSP

- Define *overlap* (s_i, s_j) as the length of the longest prefix of s_j that matches a suffix of s_i.

  aaaggcatcaaatctaaaggcatcaaa

                    aaaggcatcaaatctaaaggcatcaaa

- Construct a graph with *n* vertices representing the *n* strings s_1, s_2, …., s_n.
- Compute *overlap* (s_i, s_j) between vertices s_i and s_j.
- Insert edge Max($|s_i|$ - *overlap*, $|s_j|$ - *overlap*)
- Find the shortest path which visits every vertex exactly once. This is the **Traveling Salesman Problem** (TSP), which is *NP* – complete.

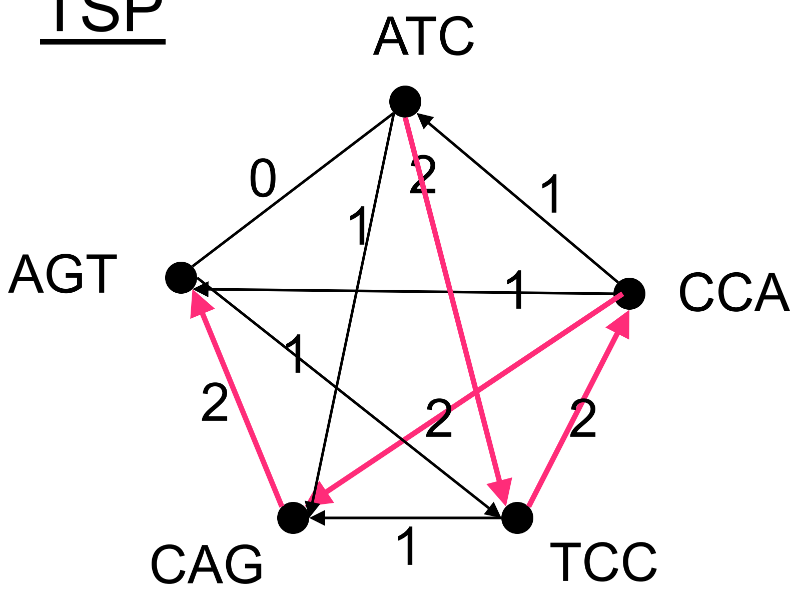
SSP to TSP: An Example

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

<u>SSP</u>

```
    AGT
  CCA
ATC
ATCCAGT
 TCC
   CAG
```

<u>TSP</u>



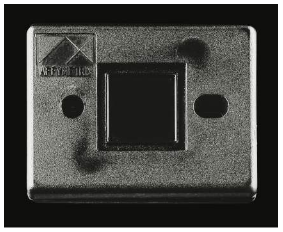
ATCCAGT

Sequencing by Hybridization (SBH): History

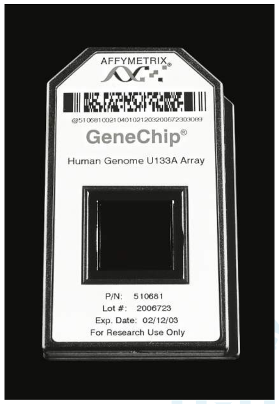
- **1988:** SBH suggested as an alternative sequencing method. Nobody believed it will ever work
- **1991:** Light directed polymer synthesis developed by Steve Fodor and colleagues.
- **1994:** Affymetrix develops first 64-kb DNA microarray

*First microarray prototype **(1989)***

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



*500,000 features per chip **(2002)***



How SBH Works

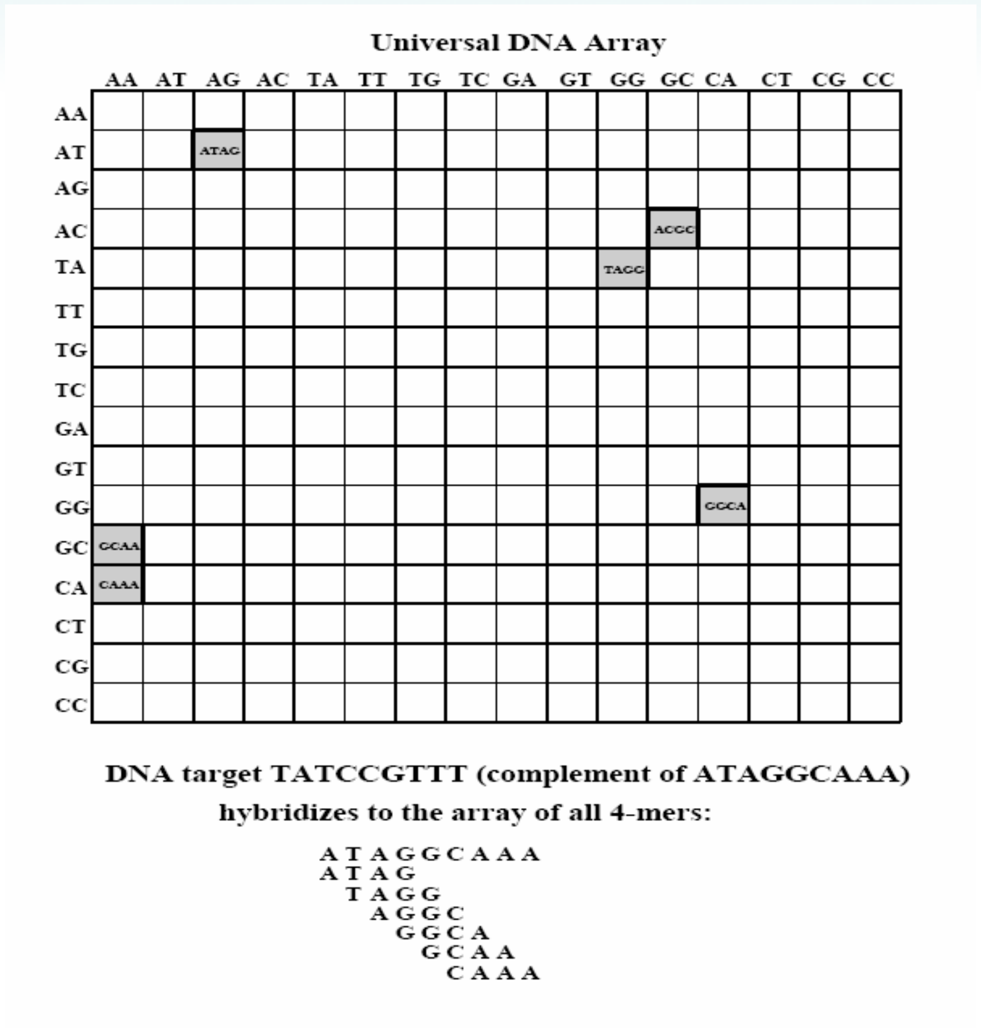
- Attach all possible DNA probes of length l to a flat surface, each probe at a distinct and known location. This set of probes is called the DNA array.
- Apply a solution containing fluorescently labeled DNA fragment to the array.
- The DNA fragment hybridizes with those probes that are complementary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

How SBH Works (cont'd)

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

- Apply the combinatorial algorithm (below) to reconstruct the sequence of the target DNA fragment from the *l*–mer composition.

Hybridization on DNA Array



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 target TATCCGTTT (complement of ATAGGCAAA)
hybridizes to the array of all 4-mers:

```
A T A G G C A A A
A T A G
  T A G G
    A G G C
      G G C A
        G C A A
          C A A A
```

l-mer composition

- ***Spectrum* (*s, l*)** - *unordered* multiset of all possible *(n – l + 1)* *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*)** - *unordered* multiset of all possible *(n – l + 1) 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}
- We usually choose the lexicographically sorted representation as the canonical one.

Different sequences – the same spectrum

- Different sequences may have the same spectrum:

  Spectrum(GTATCT,2)=

  Spectrum(GTCTAT,2)=

  {AT, CT, GT, TA, TC}

The SBH Problem

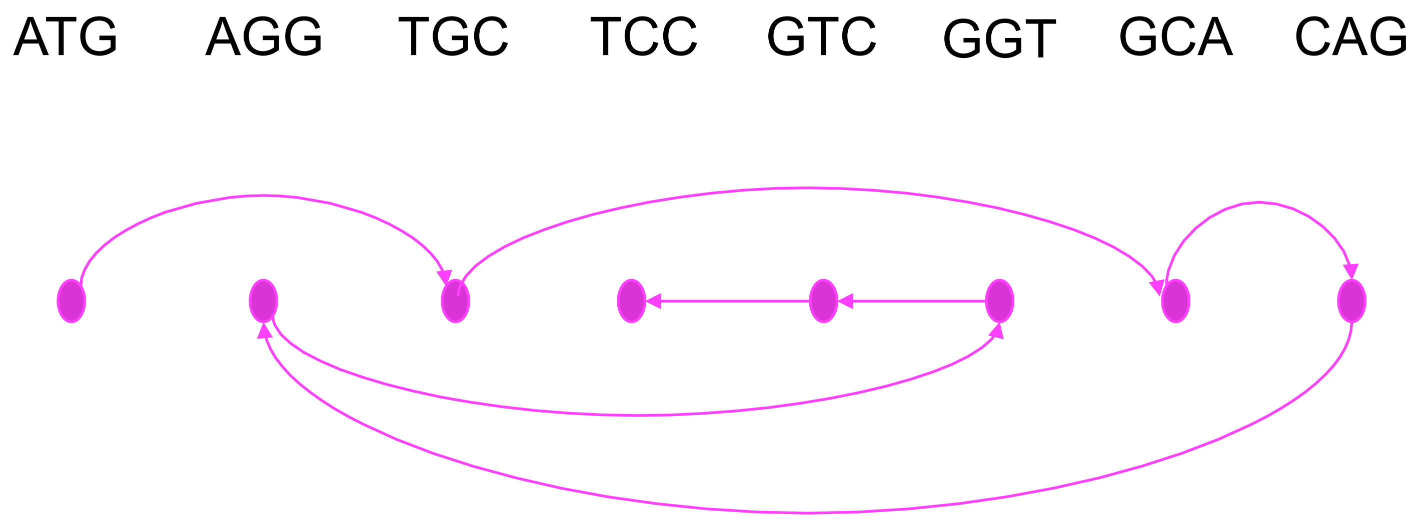
- Goal: Reconstruct a string from its *l*-mer composition

- Input: A set *S*, representing all *l*-mers from an (unknown) string *s*

- Output: String *s* such that *Spectrum* (*s*,*l*) = *S*

SBH: Hamiltonian Path Approach

S = { ATG AGG TGC TCC GTC GGT GCA CAG }



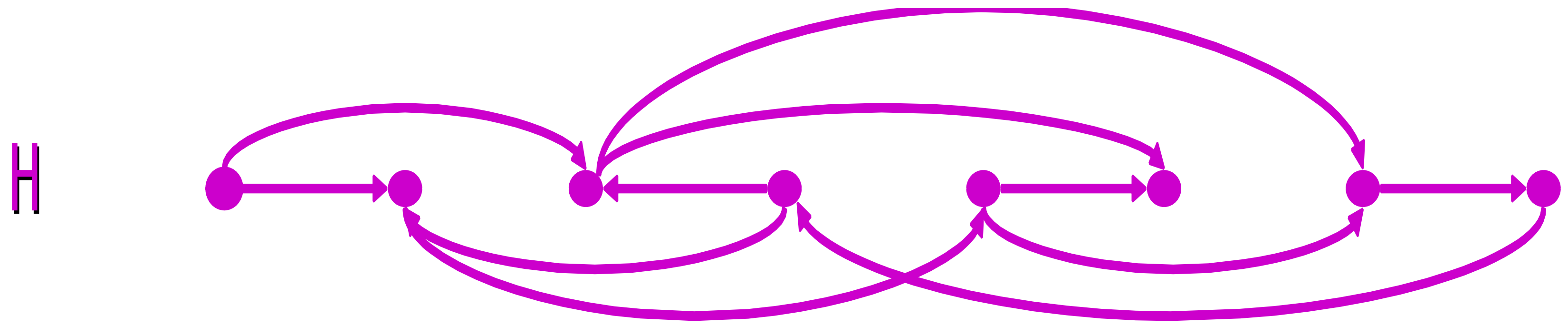
ATGCAGGTCC

Path visited every VERTEX once

SBH: Hamiltonian Path Approach

A more complicated graph:

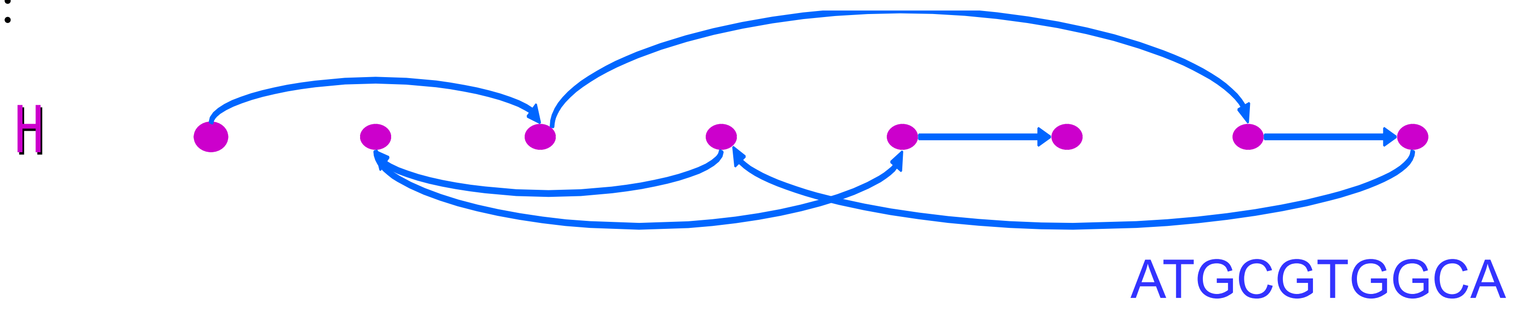
S = { ATG   TGG   TGC   GTG   GGC   GCA   GCG   CGT }



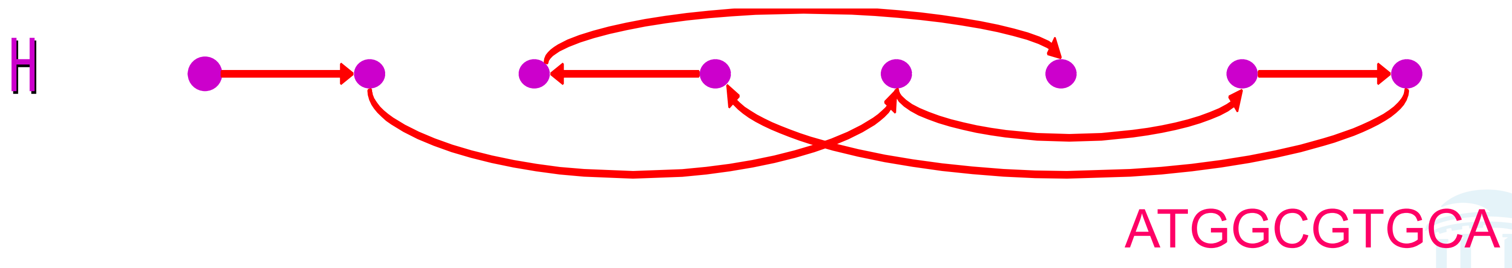
SBH: Hamiltonian Path Approach

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

Path 1:



Path 2:

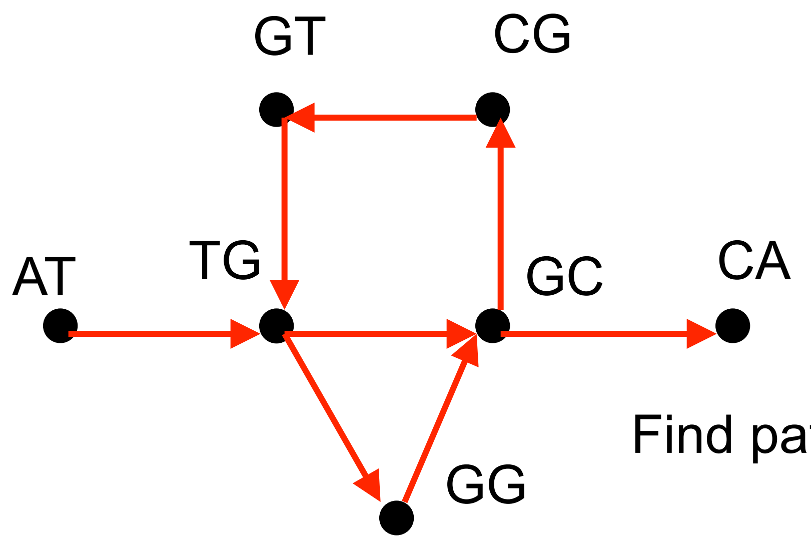


SBH: Eulerian Path Approach

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

Vertices correspond to (*l* – 1) – mers : { AT, TG, GC, GG, GT, CA, CG }

Edges correspond to *l* – mers from *S*

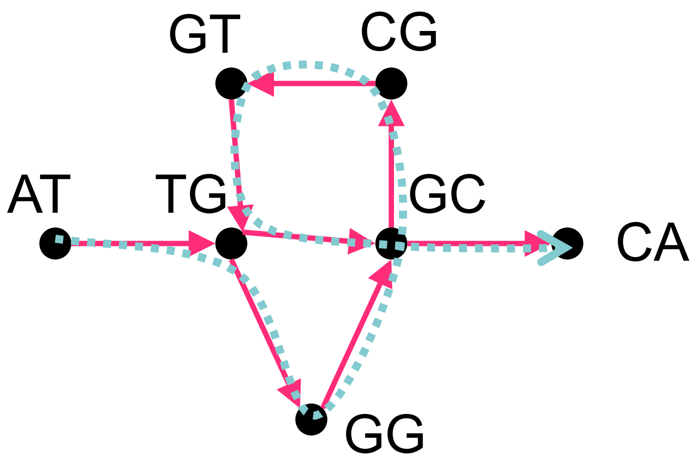


***de Bruijn Graph:** a graph representing the overlap of k-length sequences as vertices with directed edges from sequences whose k-1 suffix matches the k-1 prefix of a vertex*

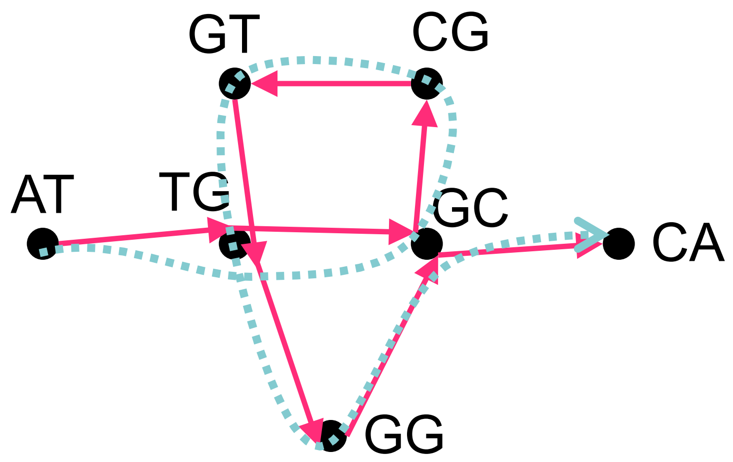
Find path that visits every EDGE once

SBH: Eulerian Path Approach

S = { AT, TG, GC, GG, GT, CA, CG } corresponds to two different paths:



ATGGCGTGCA



ATGCGTGGCA

Euler Theorem

- A graph is balanced if for every vertex the number of incoming edges equals to the number of outgoing edges:

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

- **Theorem**: *A connected graph is Eulerian if and only if each of its vertices are balanced.*

Euler Theorem: Proof

- Eulerian → balanced

  for every edge entering *v* (incoming edge) there exists an edge leaving *v* (outgoing edge). Therefore

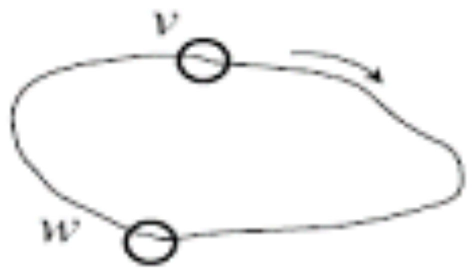
  $$in(v)=out(v)$$

- Balanced → Eulerian

  ???

Algorithm for Constructing an Eulerian Cycle

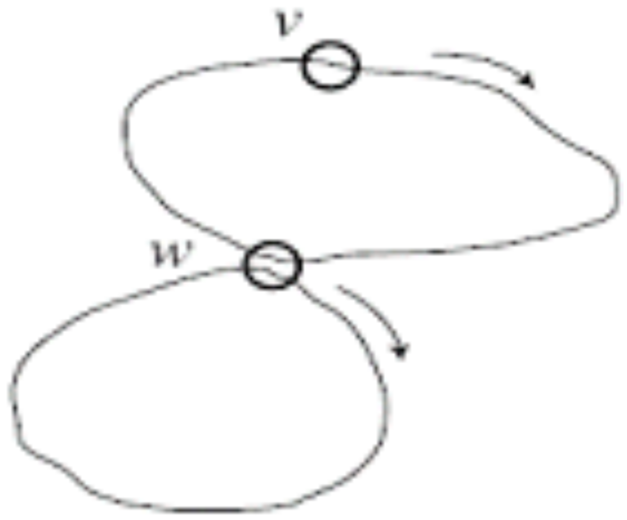
a. Start with an arbitrary vertex *v* and form an arbitrary cycle with unused edges until a dead end is reached. Since the graph is Eulerian this dead end is necessarily the starting point, i.e., vertex *v*.



(a)

Algorithm for Constructing an Eulerian Cycle (cont'd)

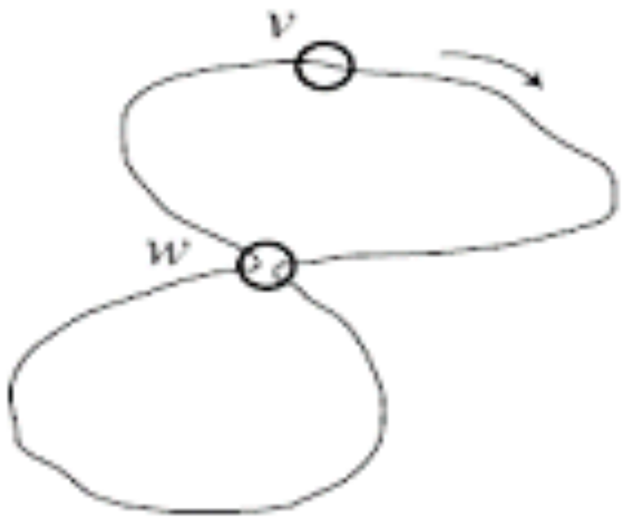
b. If cycle from (a) above is not an Eulerian cycle, it must contain a vertex *w*, which has untraversed edges. Perform step (a) again, using vertex *w* as the starting point. Once again, we will end up in the starting vertex *w*.



(b)

Algorithm for Constructing an Eulerian Cycle (cont'd)

c. Combine the cycles from (a) and (b) into a single cycle and iterate step (b).



(c)

Running time: linear to the number of edges

Euler Theorem: Extension

- **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.*
  - Semi-balanced vertex: $in(v)$ and $out(v)$ differ by 1

Some Difficulties with SBH

- **Fidelity of Hybridization:** difficult to detect differences between probes hybridized with perfect matches and 1 or 2 mismatches
- **Array Size:** 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
- **Practicality again**: Although SBH is still impractical, it spearheaded expression analysis and SNP analysis techniques