582670 Algorithms for Bioinformatics

Lecture 5: Graph Algorithms and DNA Sequencing

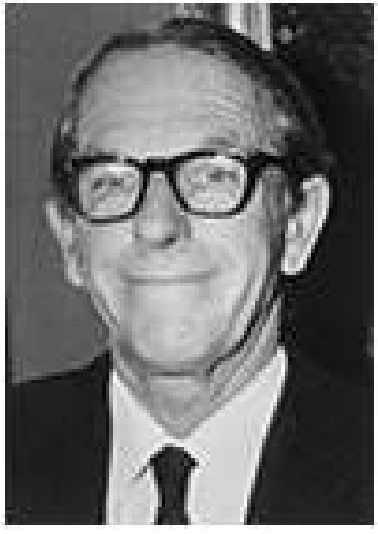
27.9.2012

Adapted from slides by Veli Mäkinen / Algorithms for Bioinformatics 2011 which are partly from `http://bix.ucsd.edu/bioalgorithms/slides.php`

DNA Sequencing: History

Sanger method (1977):

- Labeled ddNTPs terminate DNA copying at random points.



Gilbert method (1977):

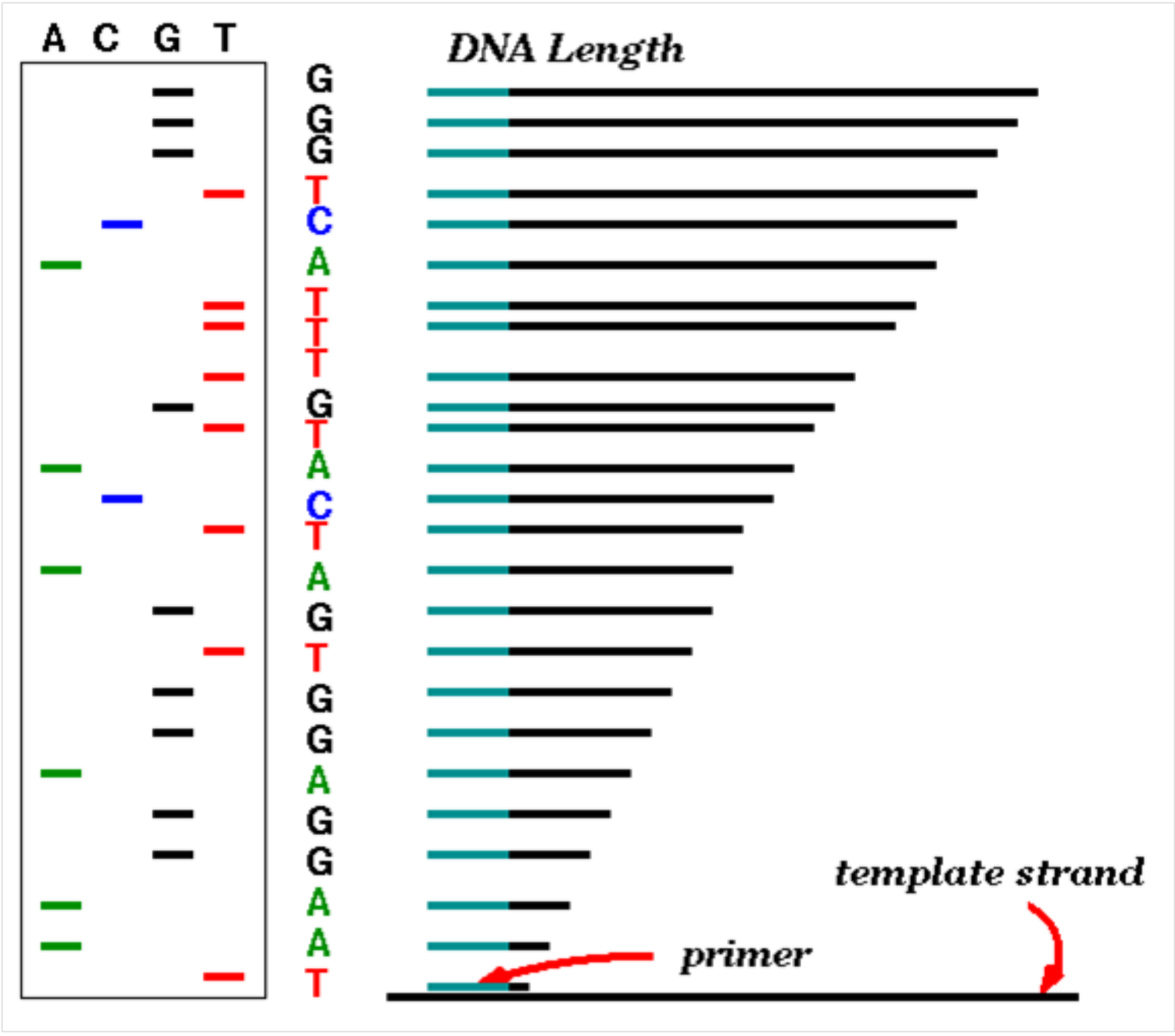
- Chemical method to cleave DNA at specific points (G, G+A, T+C, C).

- Both methods generate labeled fragments of varying lengths that are further measured by electrophoresis.

Sanger Method: Generating a Read

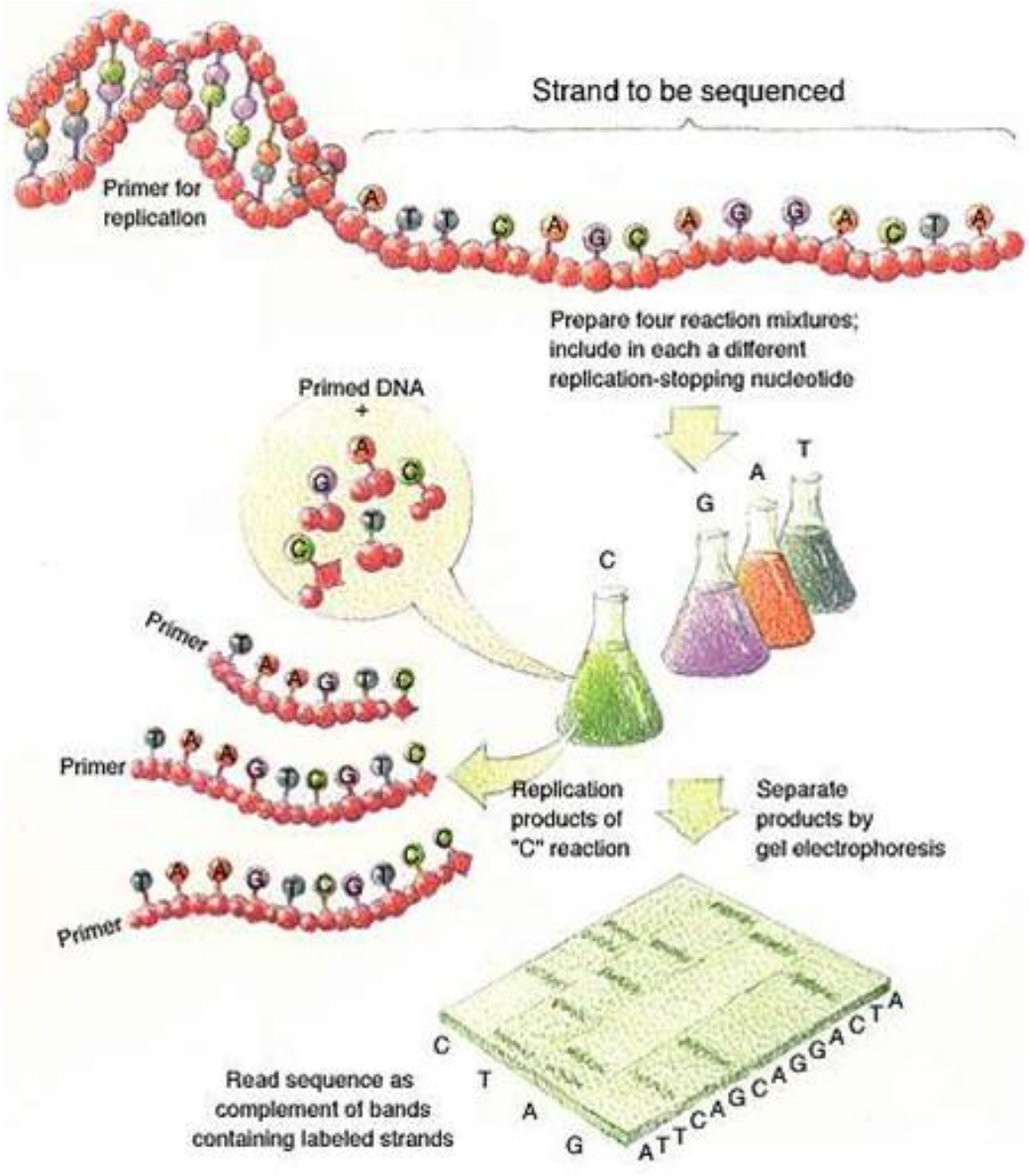
1. Divide sample into four.
2. Each sample will have available all normal nucleotides and modified nucleotides of one type (A, C, G or T) that will terminate DNA strand elongation.
3. Start at primer (restriction site).
4. Grow DNA chain.
5. In each sample the reaction will stop at all points ending with the modified nucleotide.
6. Separate products by length using gel electrophoresis.

Sanger Method: Generating a Read



DNA Sequencing

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



Fragment Assembly

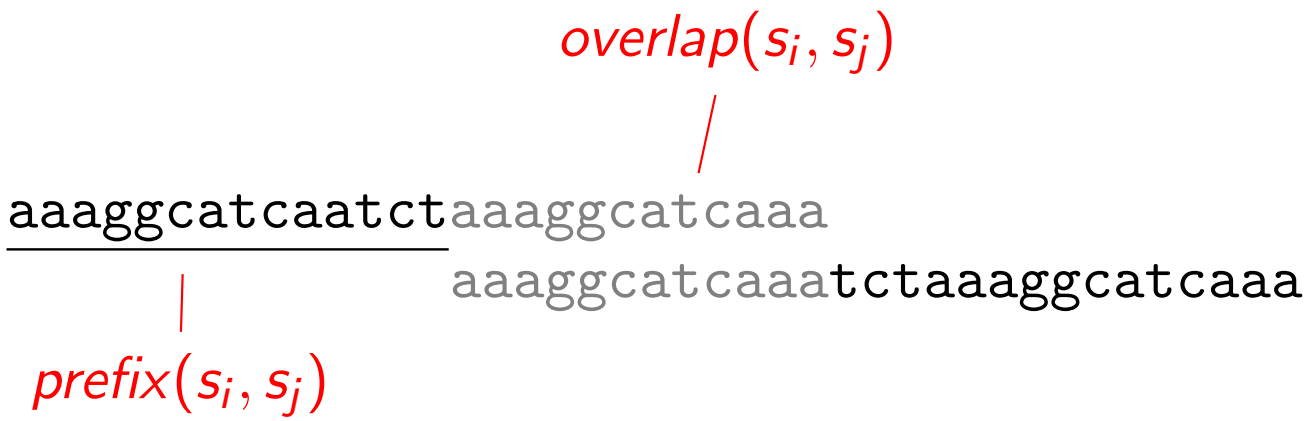
- **Computational Challenge:** assemble individual short fragments (reads) into a single genomic sequence ("superstring")
- Until late 1990s the shotgun fragment assembly of human genome was viewed as intractable problem
  - Now there exists "complete" sequences of human genomes of several individuals
- For small and "easy" genomes, such as bacterial genomes, fragment assembly is tractable with many software tools
- Remains to be difficult problem for more complex genomes

Shortest Superstring Problem

- Problem: Given a set of strings, find a shortes string that contains all of them
- Input: Strings $S = \{s_1, s_2, \ldots, s_n\}$
- Output: A string s that contains all string $s_1, s_2, \ldots s_n$ as substrings, such that the lenght of s is minimized
- **Complexity**: NP-hard
- Recall:
  - Greedy approximation algorithm at the study group
  - Extension to approximate case in the exercises

Overlaps and prefixes

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



- Define $prefix(s_i, s_j)$ as the part of s_i after its longest overlap with s_j is removed.

SSP as a Graph Problem

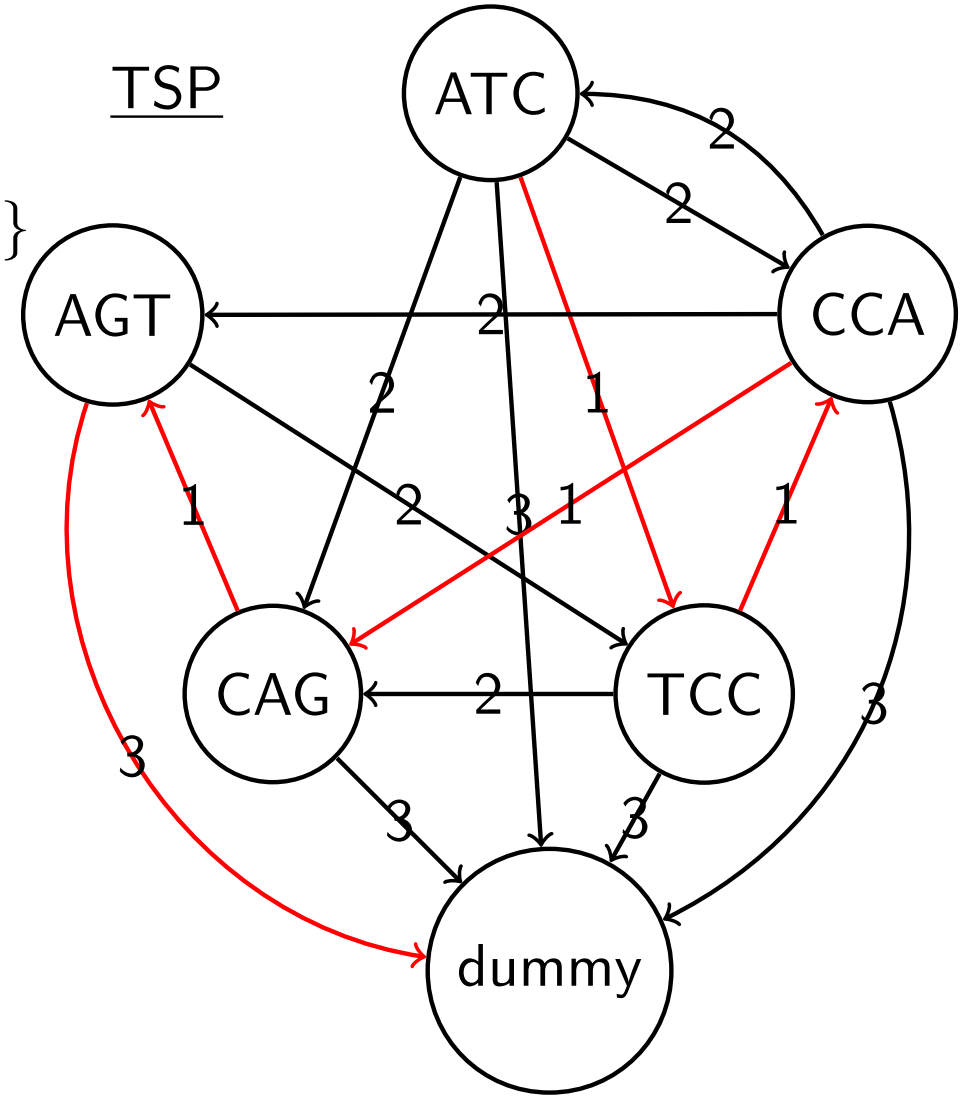
- Construct a *prefix graph* with
  - n vertices representing the n strings $s_1, s_2, \ldots, s_n$ and
  - edges of length $|prefix(s_i, s_j)|$ between vertices s_i and s_j
- Add a dummy vertex d to prefix graph with edges of length $|s_i|$ between each s_i and d.
- Find the shortest path which visits every vertex exactly once.
- This is the *Asymmetric Travelling Salesman Problem (ATSP)*, which is also NP-complete

SSP to TSP: An example

$S = \{\mathrm{ATC}, \mathrm{CCA}, \mathrm{CAG}, \mathrm{TCC}, \mathrm{AGT}\}$

SSP

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



ATCCAGT

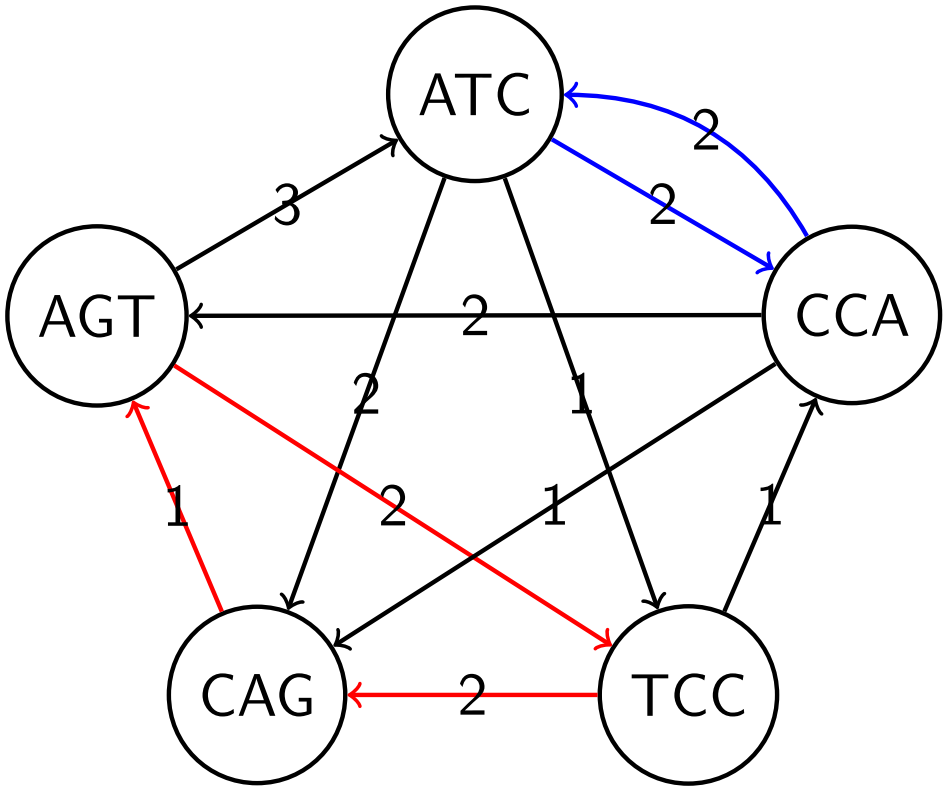
(note: only subset of edges shown)

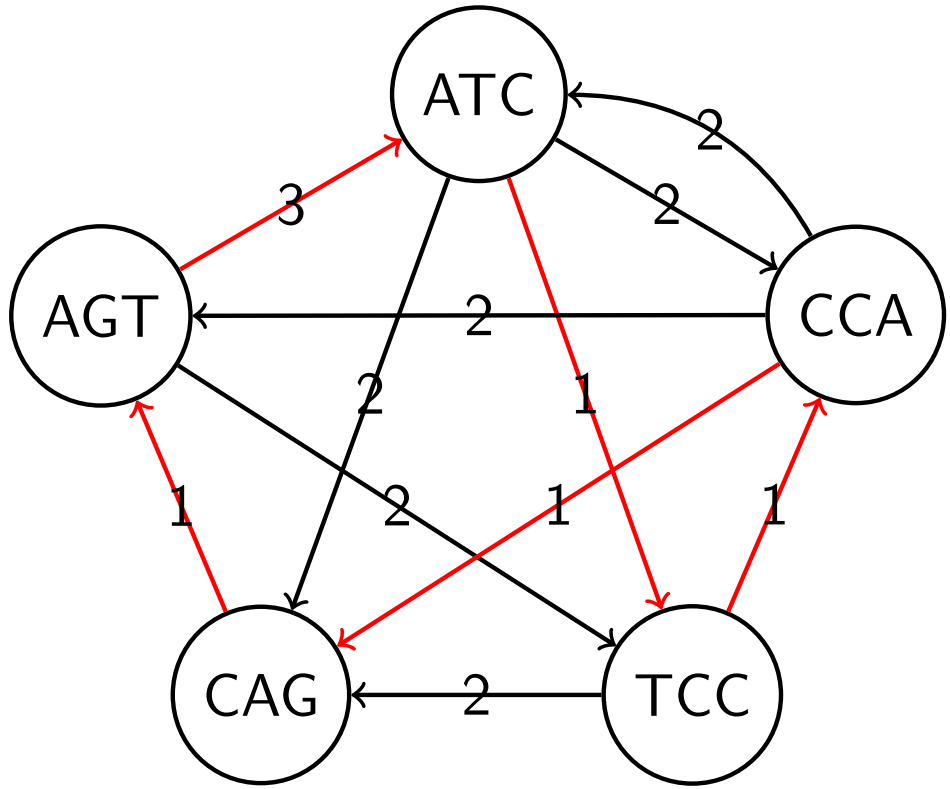
Shortest superstring: 4-approximation

- There are logarithm-factor approximation algorithms for ATSP, but the prefix graph instances admit constant factor approximations algorithms:
  - Resulting superstring is at most c times longer than the optimal OPT, for some constant c.
- 4-approximation algorithm:
  - Construct the prefix graph corresponding to strings in S
  - Find a *minimum weight cycle cover* on the prefix graph
  - Read the superstring defined by the cycle cover
  - Proof of approximation ratio in a study group.

Cycle cover

- A cycle cover is a set of disjoint cycles covering all vertices.
- ATSP tour is a special case: cycle cover with exactly one cycle.

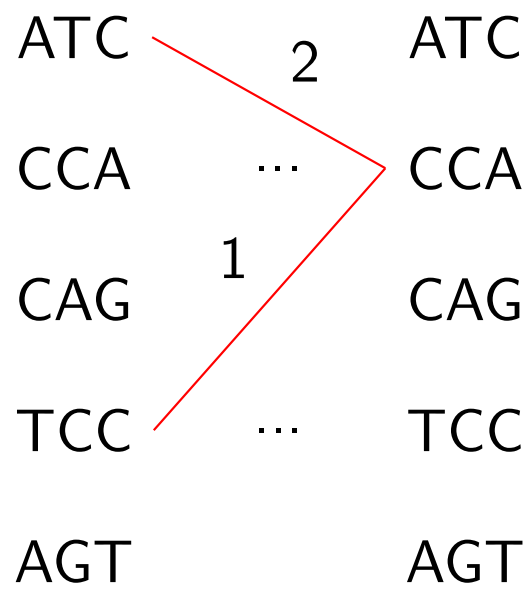




Minimum weight cycle cover

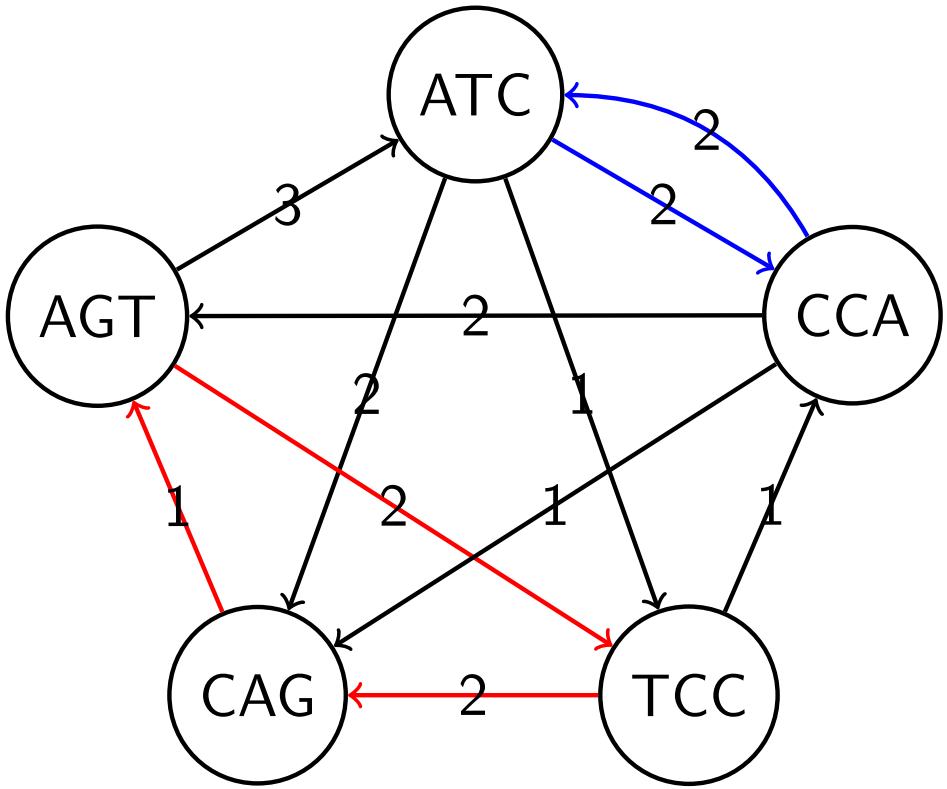
- Minimum weight cycle cover is polynomial time solvable!
- Reduction to *minimum weight perfect mathing on a bipartite graph*:
  - Bipartite graph: vertices can be divided into two sets so that all edges have one endpoint in one set and the other endpoint in the other set
  - Perfect matching: a set of disjoint edges that covers all vertices

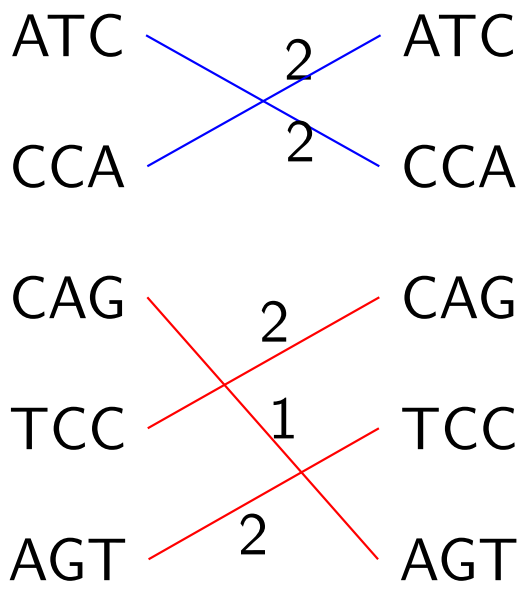
- Create two vertices u_i and v_i for each string s_i to a graph H
- Add edge (u_i, v_j) with weight $|prefix(s_i, s_j)|$ for $i \neq j$
- Each cycle cover in prefix graph corresponds to a minimum weight perfect matching on H and vice versa.



Minimum weight perfect matching

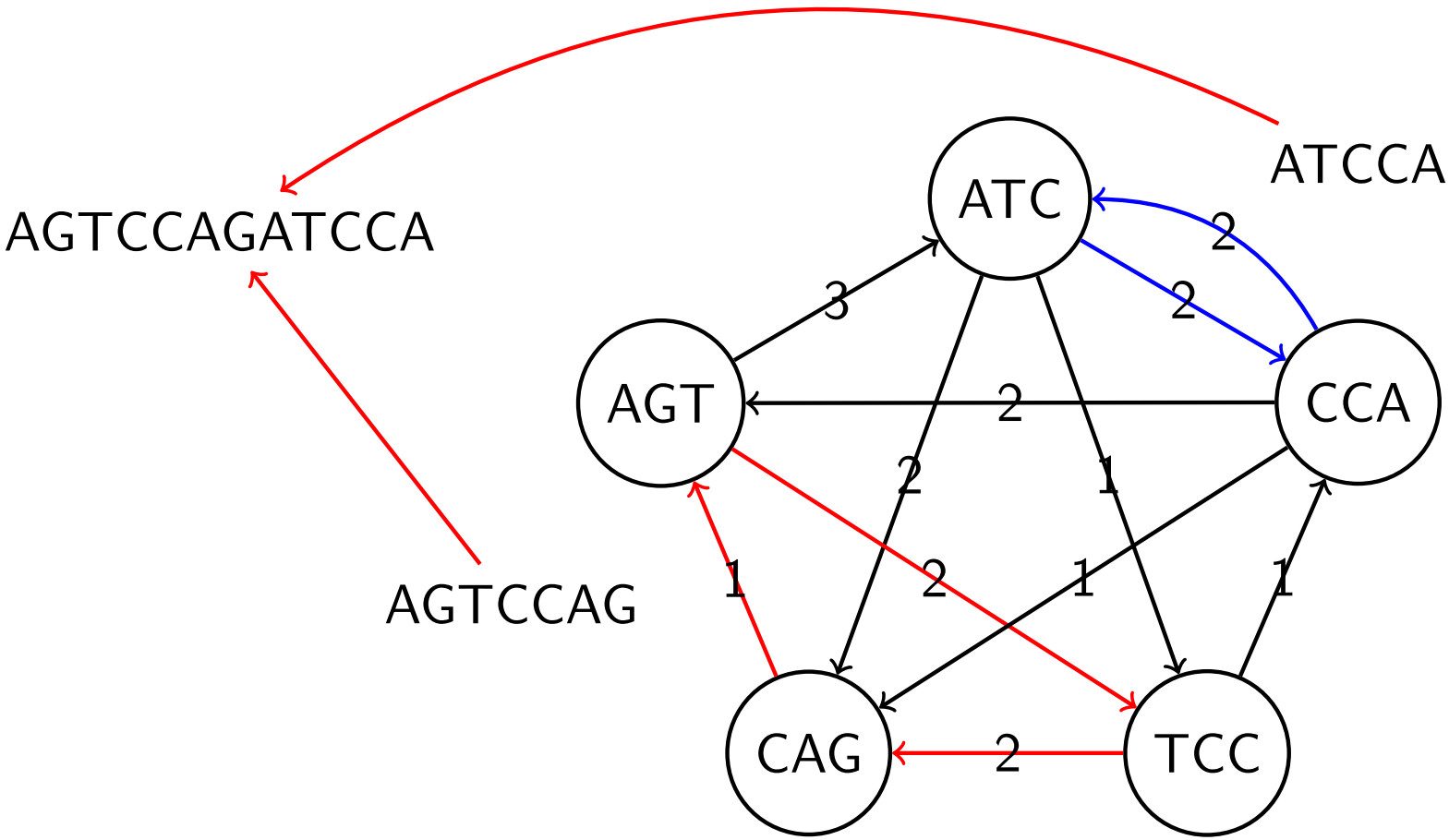
- Classical non-trivial graph problem with polynomial time solutions.





Reading superstring from cycle cover

- For each cycle
  - concatenate prefixes corresponding to weight starting from any vertex
  - append the overlap of last and first vertex
- Concatenate the string read from each cycle

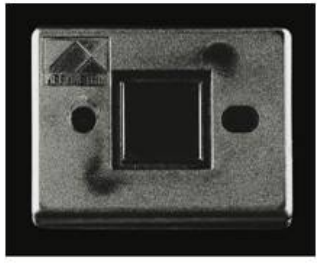


Sequencing by Hybridization (SBH): History

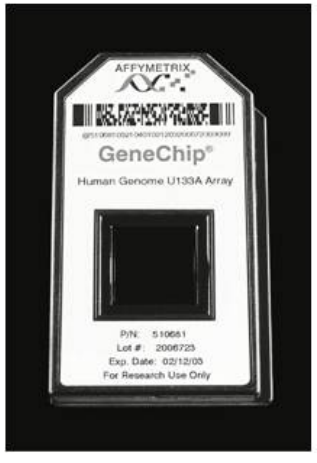
- 1988: SBH suggested as an alternative sequencing method. Nobody believed it will ever work.
- 1991: Light directed polymer synthesis developed by Steve Fodor and colleagues.
- 1994: Affymetrix develops first 64-kb DNA microarray.

First microarray prototype (1989)

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



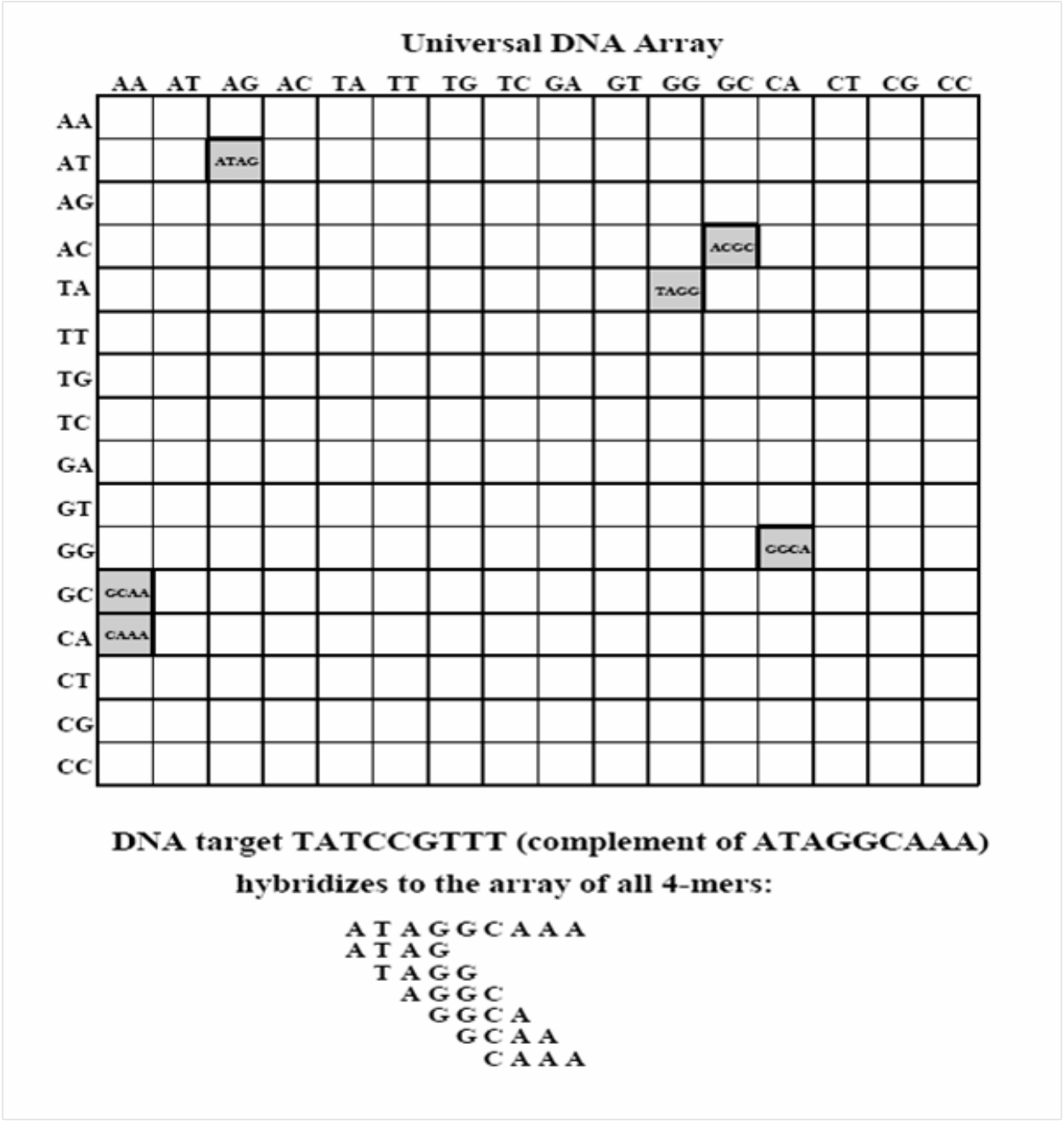
500,000 features per chip (2002)



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 microarray*.
- 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.
- Using a spectroscopic detector, determine which probes hybridize to the DNA fragment to obtain the ℓ-mer composition of the DNA fragment.
- Reconstruct the sequence of the DNA fragment from the ℓ-mer composition.

Hybridization on DNA Array



ℓ-mer composition

- *Spectrum*(s, ℓ) is a multiset of all possible $(n - \ell + 1)$ ℓ-mers in a string s of length n.
- E.g. for $s = \mathrm{TATGGTGC}$, *Spectrum*$(s, 3)$:

$$S = \{\mathrm{TAT}, \mathrm{ATG}, \mathrm{TGG}, \mathrm{GGT}, \mathrm{GTG}, \mathrm{TGC}\}$$

- Different sequences may have the same spectrum:

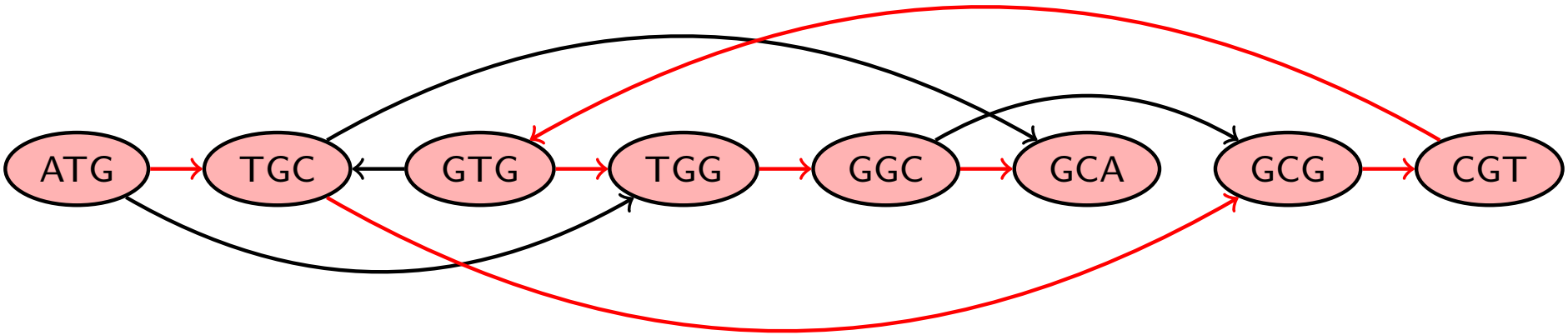
$$\textit{Spectrum}(\mathrm{GTATCT}, 2) =$$
$$\textit{Spectrum}(\mathrm{GTCTAT}, 2) =$$
$$\{\mathrm{AT}, \mathrm{CT}, \mathrm{GT}, \mathrm{TA}, \mathrm{TC}\}$$

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: A string s such that $Spectrum(s, \ell) = S$

SBH: Hamiltonian Path Approach

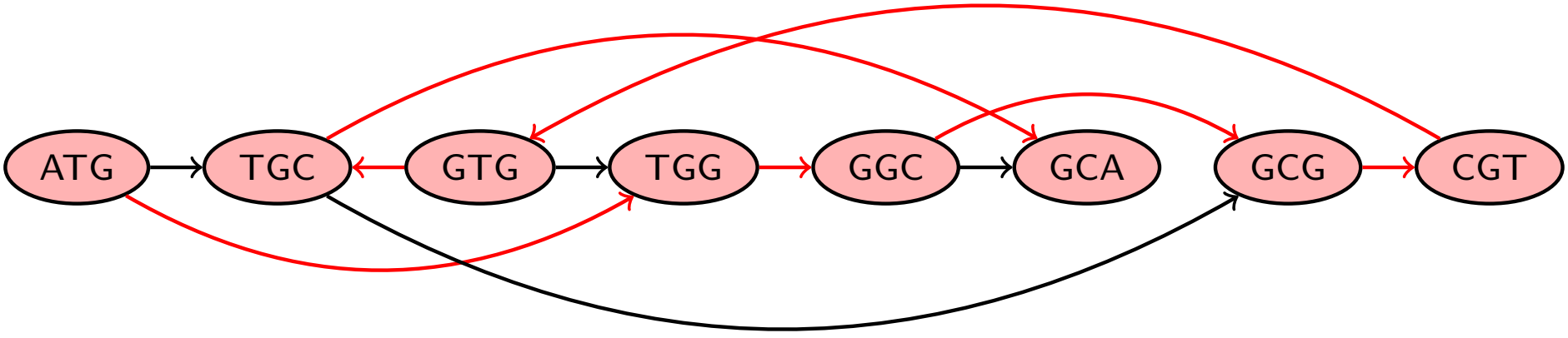
- Construct a graph
  - One vertex for each ℓ-mer in the input spectrum
  - Draw an edge between two vertices if the ℓ-mers overlap by $\ell - 1$ nucleotides
- Find a path that visits each **vertex** once.
- Example: $S = \{\mathrm{ATG}, \mathrm{TGC}, \mathrm{GTG}, \mathrm{TGG}, \mathrm{GGC}, \mathrm{GCA}, \mathrm{GCG}, \mathrm{CGT}\}$



ATGCGTGGCA

SBH: Hamiltonian Path Approach

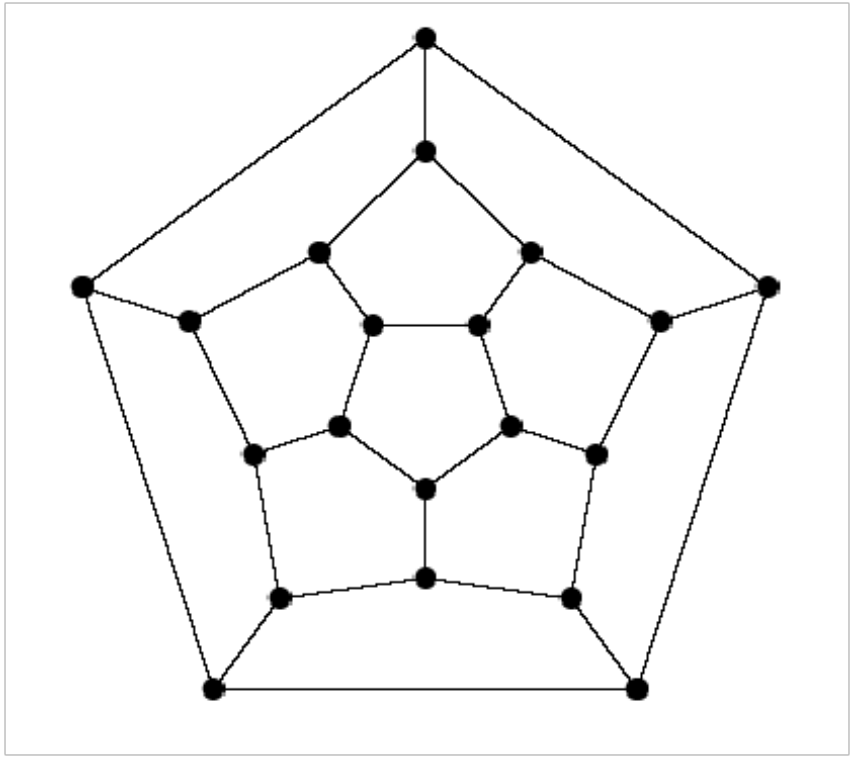
- Another path for:
  $S = \{\mathrm{ATG}, \mathrm{TGC}, \mathrm{GTG}, \mathrm{TGG}, \mathrm{GGC}, \mathrm{GCA}, \mathrm{GCG}, \mathrm{CGT}\}$



ATGGCGTGCA

Hamiltonian Cycle Problem

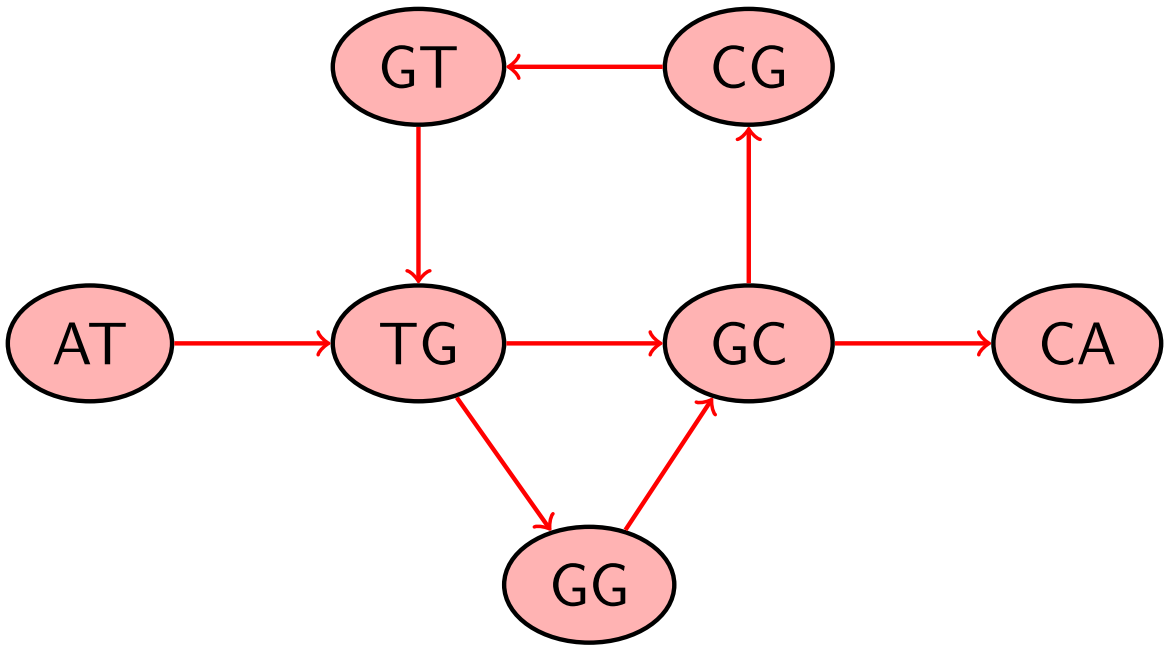
- Find a cycle that visit every **vertex** exactly once.
- NP-complete



Game invented by Sir William Hamilton in 1857

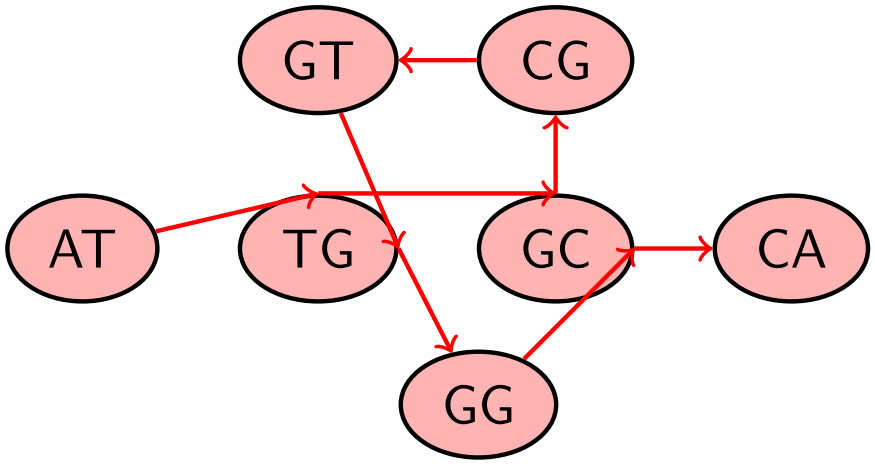
SBH: Eulerian Path Approach

- Construct a graph
  - A vertex for each $(\ell - 1)$-mer
  - An edge between two vertices corresponds to an ℓ-mer from S
  - Find a path that visits each **edge** once.
  - Example: $S = \{\mathrm{ATG}, \mathrm{TGC}, \mathrm{GTG}, \mathrm{TGG}, \mathrm{GGC}, \mathrm{GCA}, \mathrm{GCG}, \mathrm{CGT}\}$

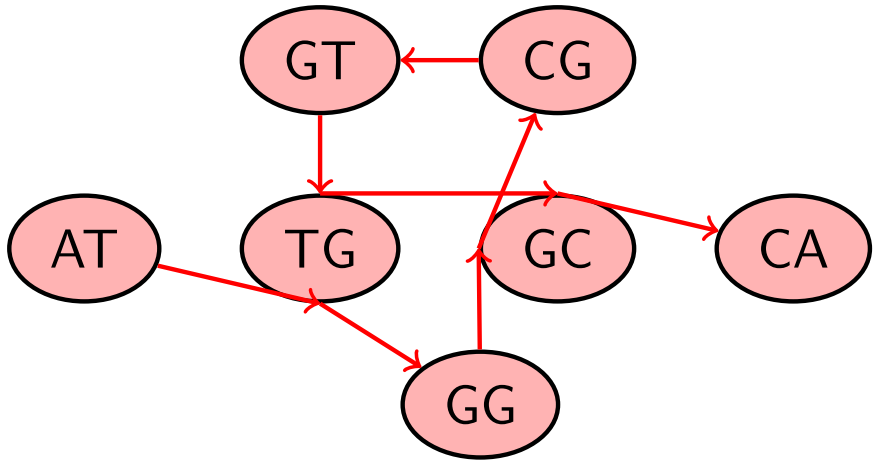


SBH: Eulerian Path Approach

- $S = \{\mathrm{ATG}, \mathrm{TGC}, \mathrm{GTG}, \mathrm{TGG}, \mathrm{GGC}, \mathrm{GCA}, \mathrm{GCG}, \mathrm{CGT}\}$ corresponds to two different paths:



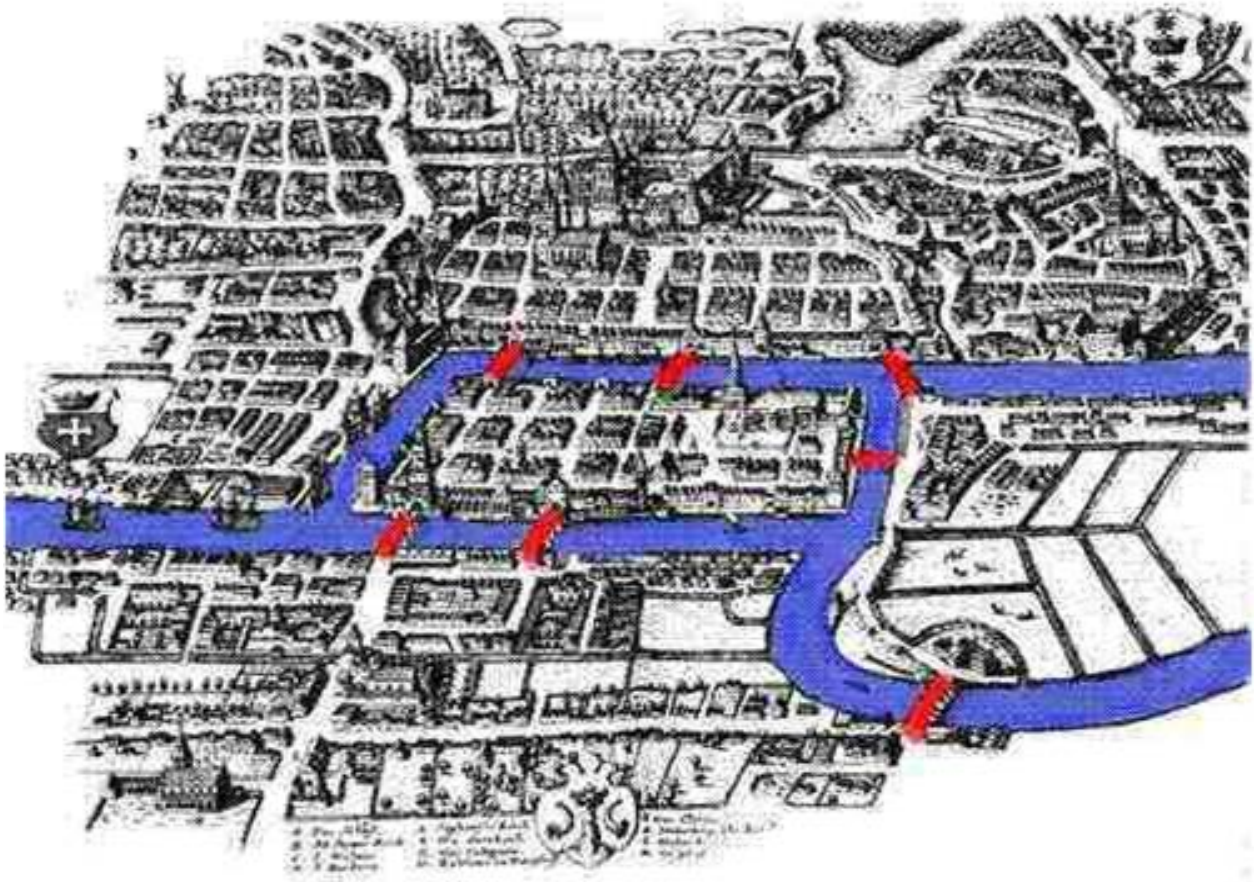
ATGCGTGGCA



ATGGCGTGCA

The Bridge Obsession Problem

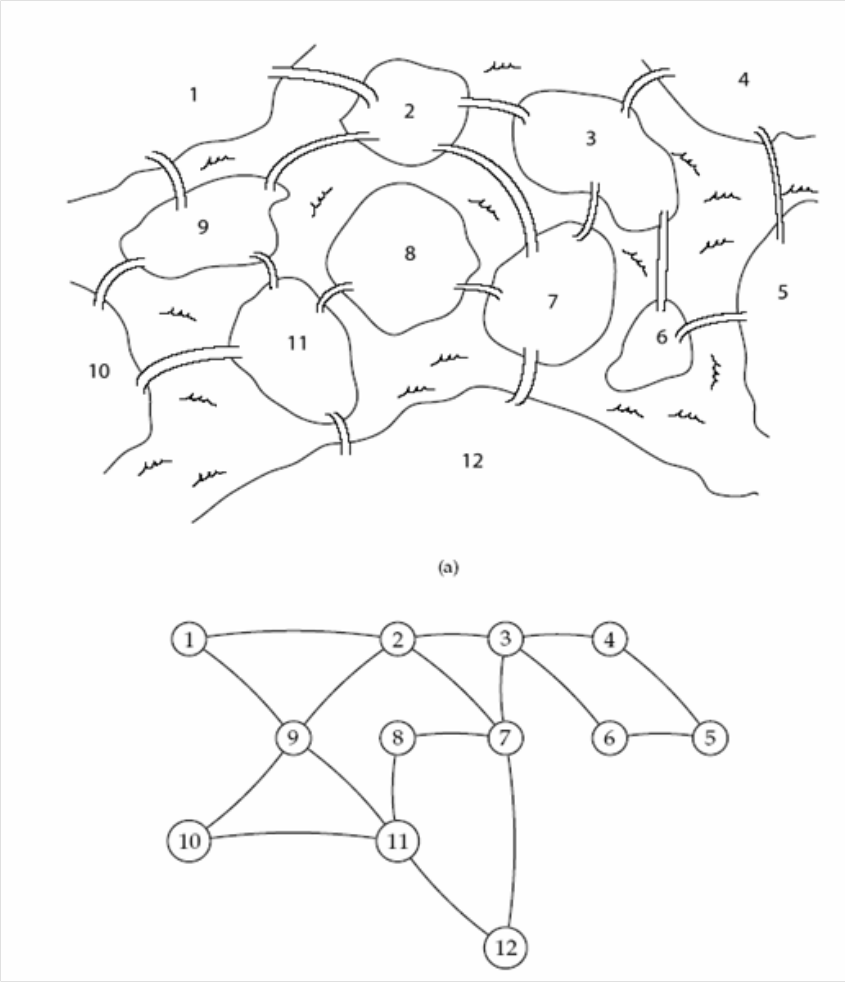
Find a tour crossing every bridge just once
Leonhard Euler, 1735



Bridges of Königsberg

Eulerian Cycle Problem

- Find a cycle that visits every **edge** exactly once
- Linear time



More complicated Königsberg

Euler Theorems

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

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

- **Theorem**: A connected graph has an *Eulerian cycle* if and only if each of its vertices is balanced.
- A vertex is *semi-balanced* if $in(v) = out(v) + 1$ or $in(v) = out(v) - 1$
- A graph is *balanced* is for every vertex the number of incoming edges equals the number of ougoing edges:
- **Theorem**: A connected graph has an *Eulerian path* if and only if it contains a vertex v with $in(v) = out(v) - 1$, a vertex w with $in(w) = out(w) + 1$ and all other vertices are balanced.

Some Difficulties with SBH

- In practise, ℓ-mer composition can never be measured with 100% accuracy
  - With inaccurate data, the computational problem is again NP-hard.
    - Find minimum completion (insertion/deletion of edges and vertices) of the graph so that it becomes Eulerian
    - Jacek Błazewicz and Marta Kasprzak: Complexity of DNA sequencing by hyridization. *Theoretical Computer Science*, 290(3):1459–1473, 2003.
- Microarray technology has found other uses:
  - Widely used in expression analysis and SNP analysis
- Virtual ℓ-mer compositions are used in many fragment assembly tools, leading to heuristics exploiting the Eulerian path approach.

Outline

Study Group 1: Lastnames A-C

- Read pages 284–290 from Jones and Pevzner.
  - The peptide sequencing problem
- At study group draw an example spectrum graph.

Study Group 2: Lastnames D-L

- Read pages 61–64 from Vazirani: Approximation Algorithms, Springer, 2001.
  - Analysis of the 4-approximation algorithm for Shortest Superstring Problem.
  - Copies distributed at lecture. Ask lecturer for a pdf if you were not present.
- At study group explain visually the proofs of Lemmas 7.2. and 7.3. Explain how Lemma 7.3 leads to the proof of Theorem 7.4.

Study Group 3: Lastnames M-Z

- Read pages 272–275 from Jones and Pevzner.
  - Eulerian cycles and paths.
- At study group explain the algorithm for finding a Eulerian cycle using an example. How can the algorithm be modified for finding a Eulerian path?