COM SCI 122 Week 4

Aidan Jan

January 30, 2025

Graph Algorithms in Bioinformatics

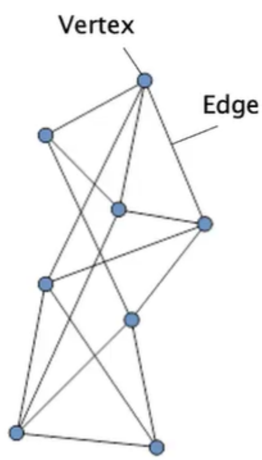
Introduction

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



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

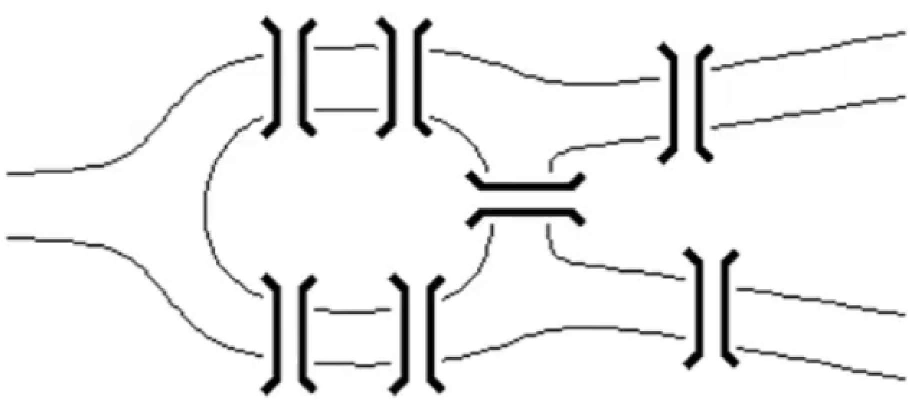
Traveling Salesman Problem (TSP)

- 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 below: 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).

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:



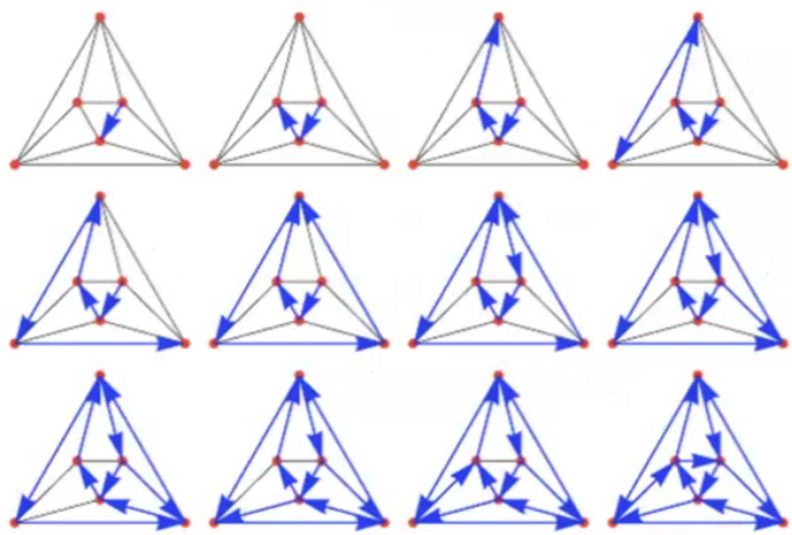
- The residents wanted to know if they could take a walk from anywhere in the city, cross each bridge exactly once, and wind up where they started.

In 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! We are now looking for 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: below is a demonstration of an Eulerian cycle.



Theorem:

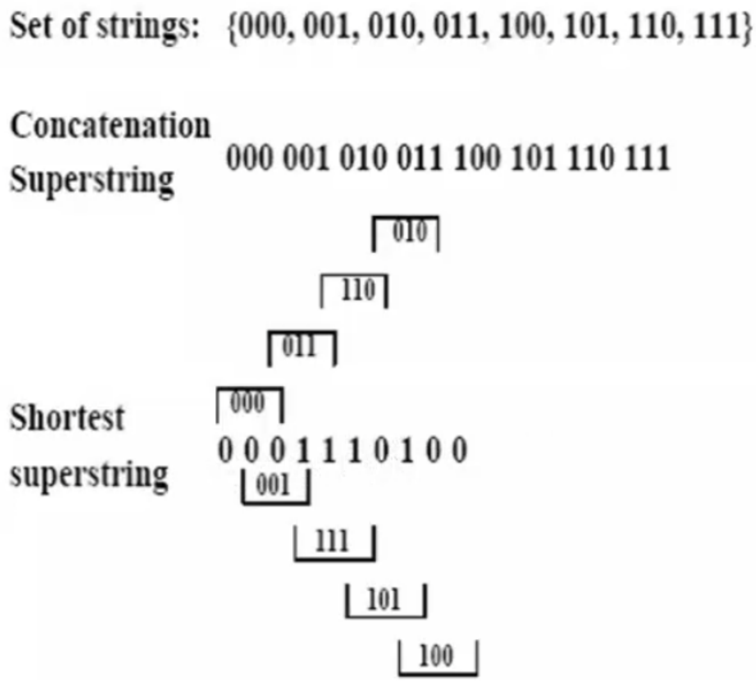
The Eulerian Cycle Problem can be solved in linear time

- So whereas finding a Hamiltonian cycle quickly becomes intractable for an arbitrary graph, finding an Eulerian cycle is relatively much easier

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.

Example:



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

To do this, we can define an *overlap function*.

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 =                AAAGGCATCAAATCTAAAGGCATCAAA
```

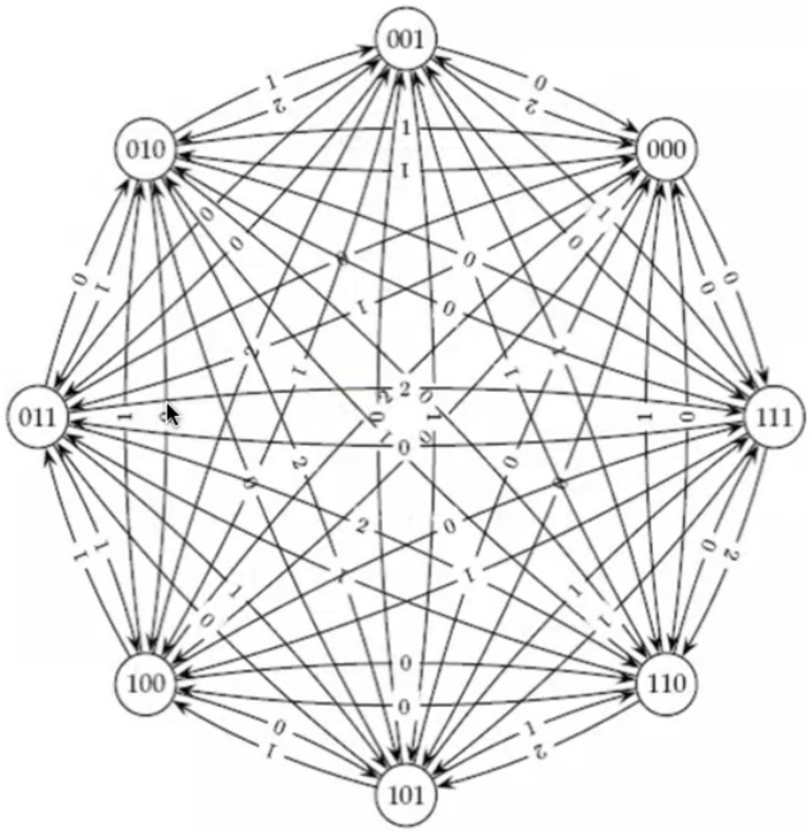
  - Therefore, $overlap(s_1, s_2) = 12$.

Why is SSP an NP-Complete Problem?

- Construct a graph G as follows:
  - The n vertices represent the n strings $s_1, s_2, \ldots, s_n$.
  - 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!

Reducing SSP to TSP: Example 1

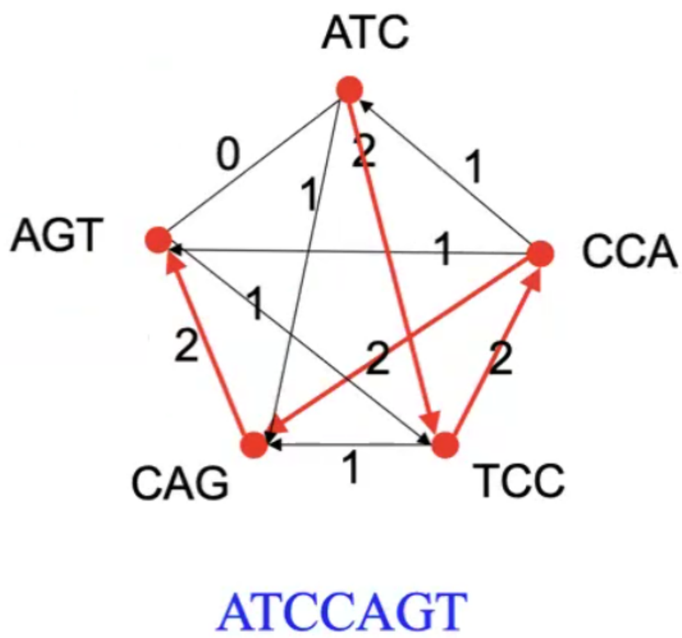
- Take our previous set of strings S = {000, 001, 010, 011, 100, 101, 110, 111}
- Then the graph for S is below:



- One minimal Hamiltonian path gives our previous superstring, 0001110100.

Reducing SSP to TSP: Example 2

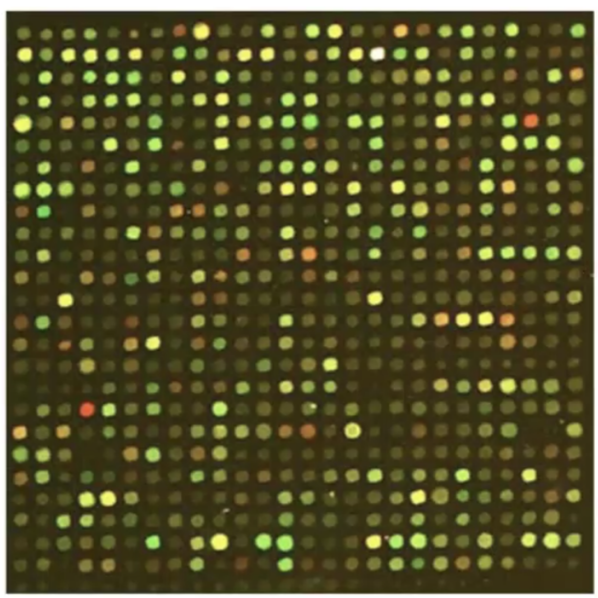
- S = {ATC, CCA, CAG, TCC, AGT}
- This graph is provided below.



- A minimal Hamiltonian path gives as shortest superstring ATCCAGT.

Sequencing By Hybridization

- 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.

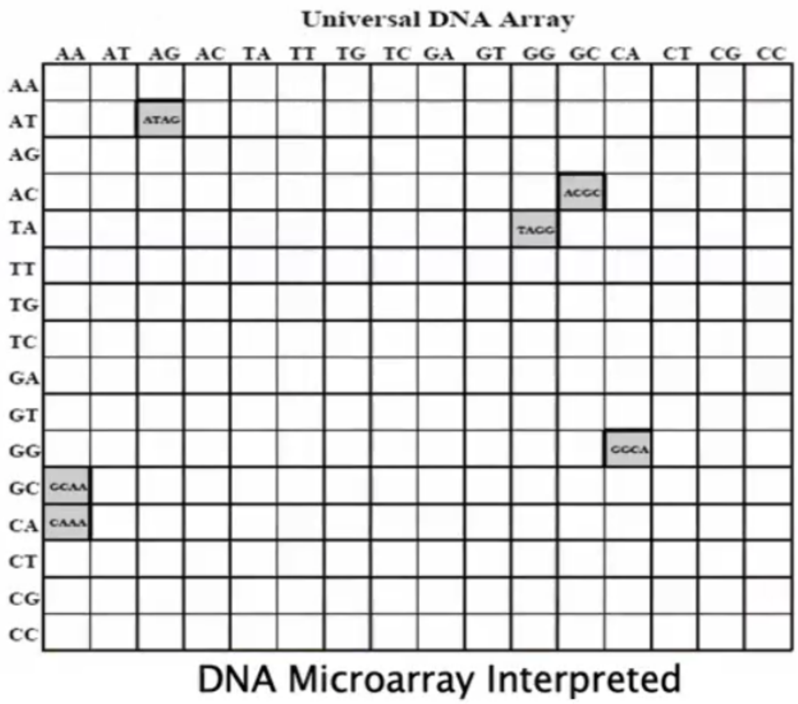


DNA Microarray

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.



l-mer Composition

- *Spectrum*(s, l): The *unordered* 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:

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

The SBH Problem

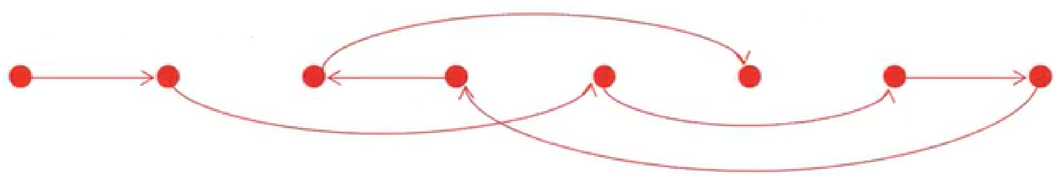
- Problem: Reconstruct a string from its l-mer composition
- Input: A set S, representing all l-mers from an (unknown) string s
- Output: A string s such that $Spectrum(s, l) = S$
- **Note:** As we have seen, there may be more than one correct answer. Determining which DNA sequence is actually correct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

Example:

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



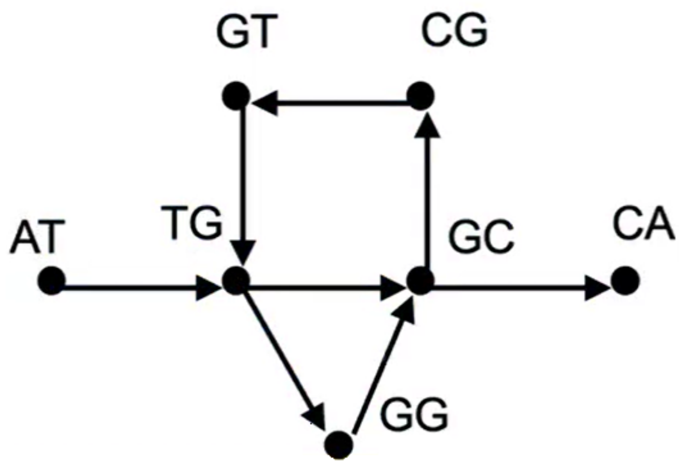
- There are actually two Hamiltonian paths in this graph:
  - Path 1: Gives the string S = ATGCGTGGCA
  - Path 2: Gives the string S = ATGGCGTGCA

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.

SPH: 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 below.



- **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:
  - ATGGCGTGCA
  - ATGCGTGGCA

But, How do we know an Eulerian Path exists?

- A graph is **balanced** if for every vertex the number of incoming edes equals to the number of outgoign 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 be demonstrating the following:
  1. Every Eulerian graph is balanced.
  2. Every balanced graph is Eulerian.

Proof: Every Eulerian Graph is Balanced

- Suppose we have an Eulerian graph G. Call C the Eulerain 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.

Proof: 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*.
- We have two simple cases for our cycle, which we call C:
  1. C is an Eulerian cycle $\rightarrow$ G is Eulerian $\rightarrow$ 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'.
- Combine our cycles C and C' into a bigger cycle C* by swapping edges at w.
- Once again, we test C*:
  1. C* is an Eulerian cycle $\rightarrow$ G is Eulerian $\rightarrow$ 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

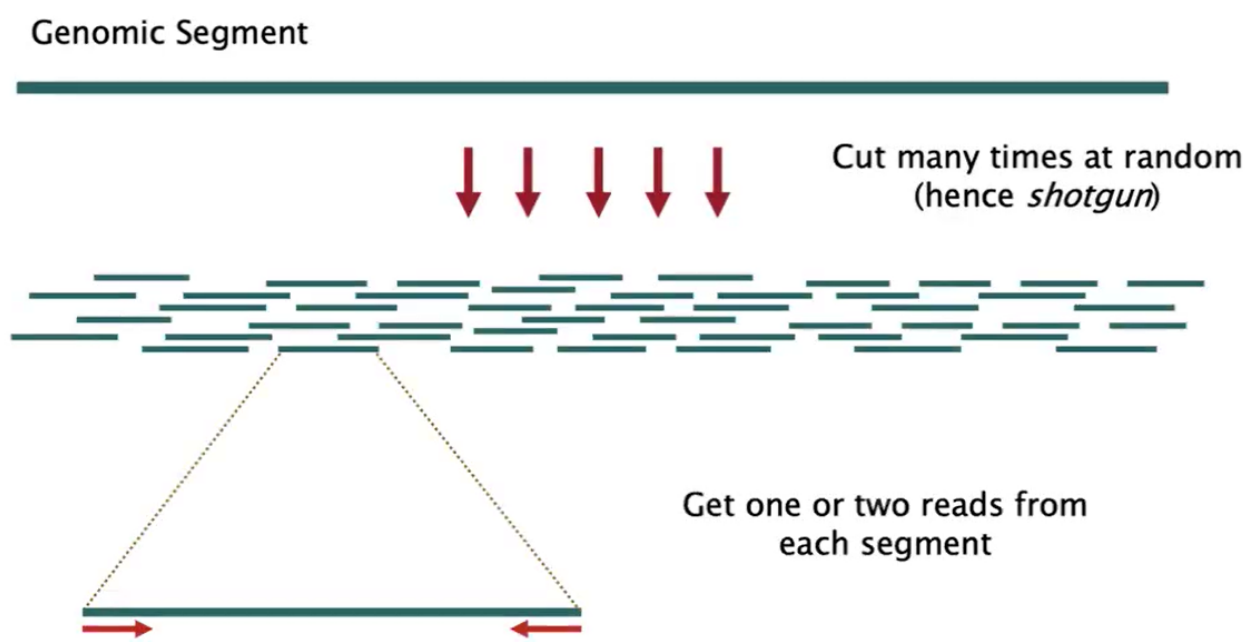
- A vertex v is **semi-balanced** if either $in(v) = out(v) + 1$ or $in(v) = out(v) - 1$.
- **Theorem:** A connected graph has na Eulerian path if and only if it contains at most two semi-balanced vertices and all other vertices are balanced.
  - If G has no semi-balanced vertices, DONE.
  - If G has 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.
- **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.

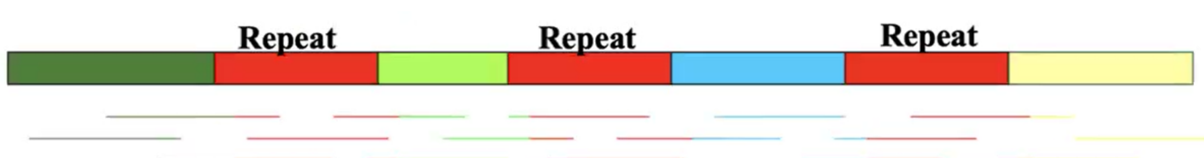
Fragment Assembly and Repeats in DNA

Shotgun Sequencing



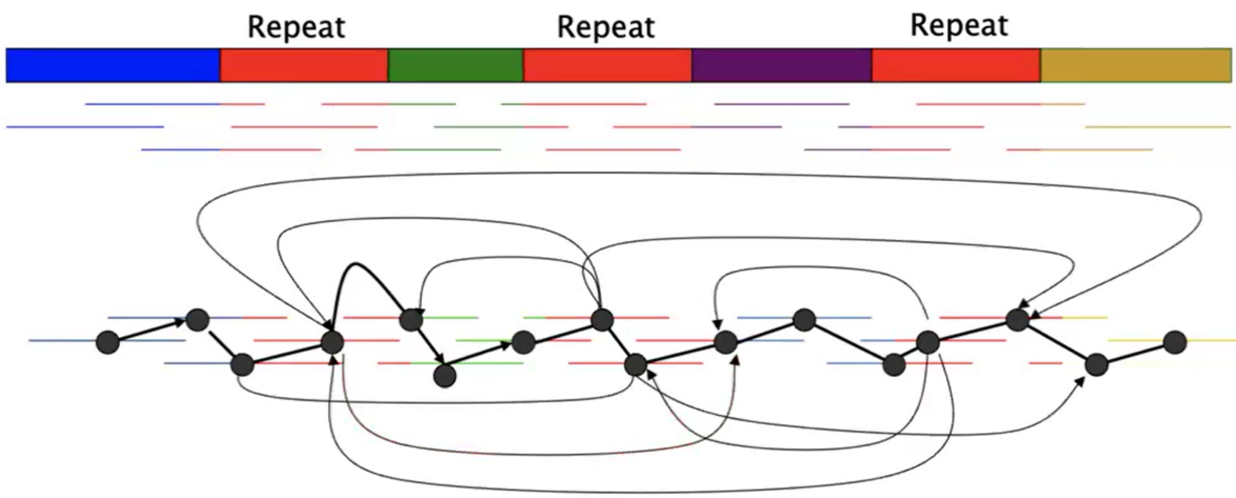
Challenges in Fragment Assembly

- 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)



Overlap Graph: Hamiltonian Approach

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

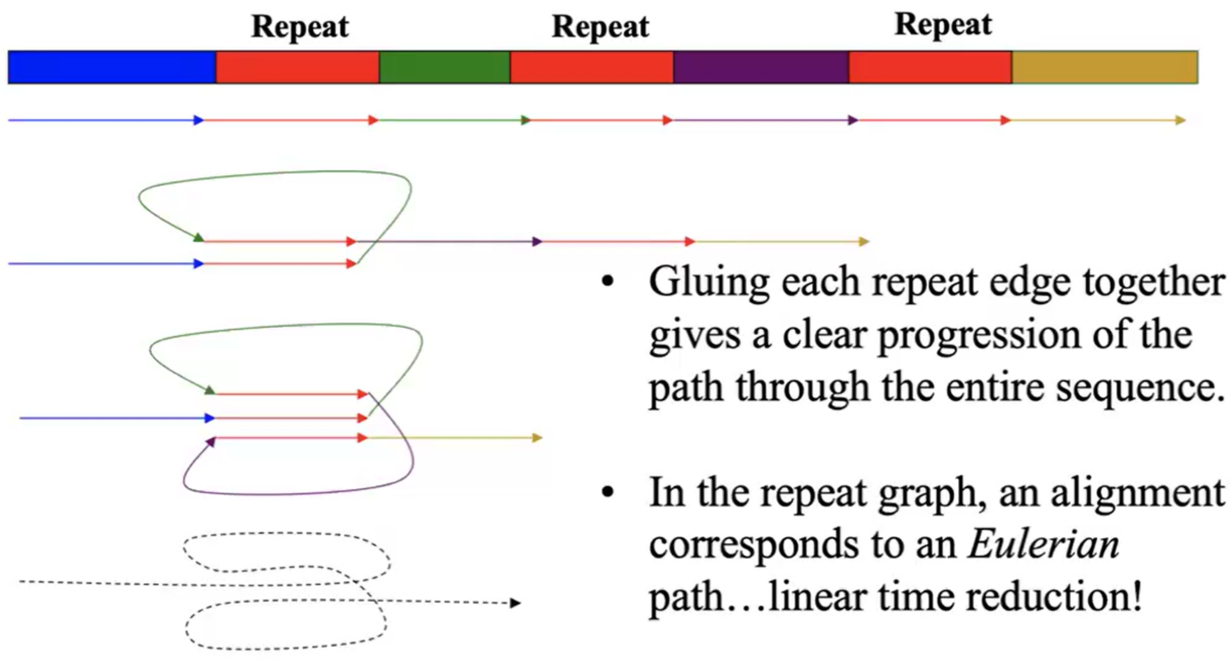


- So finding an alignment corresponds to finding a Hamiltonian path in the overlap graph.
- Recall that the Hamiltonian path/cycle problem is *NP-Complete*: no efficient algorithms are known.

Euler Approach to Fragment Assembly

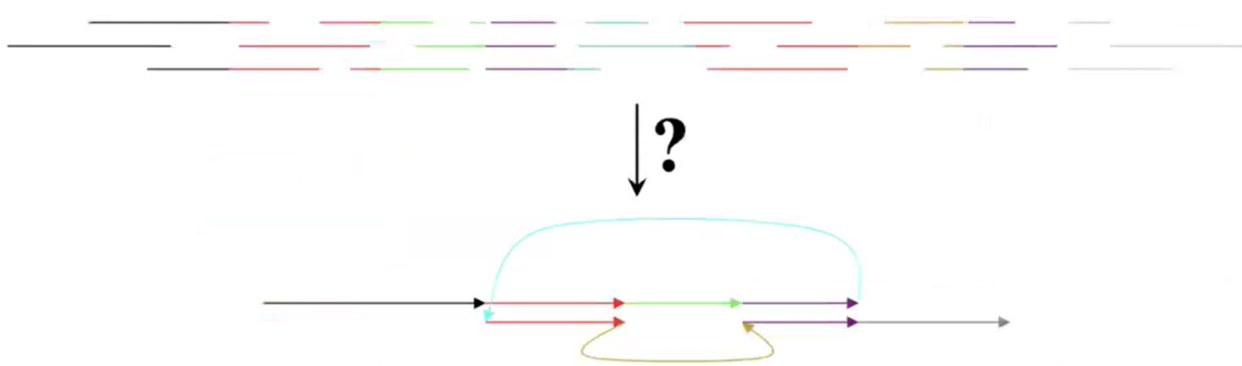
- The "overlap-layout-consensus" technique implicitly solves the Hamiltonian path problem and has a high rate of mis-assembly.
- Can we adapt the Eulerian Path approach borrowed from the SBH problem?
- Fragment assembly without repeat masking can be done in linear time with greater accuracy.

Repeat Graph: Eulerian Approach



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 fragmets?
- **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.
- 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.)

Practical Sequence Assembly

- Split reads into kmers
- Error correct kmers based on occurrence threshold
- Construct De Bruijn Graph (Eulerian Graph)
- Find Eulerian Path (or as many long paths as possible)

Project 2 - Sequence Assembly

- Given reads - reconstruct the genome using assembly
- Project 2a - Input is "spectrum" of a sequence.
- Project 2b - Input is "reads" of a sequence.
- Output - Sort the "input" in order on the reconstructed sequence.

Practical Genome Assembly

- Convert reads to spectrum
  - Convert reads to kmers
  - Error corretion and copy number count
- Obtain Sequence from Spectrum
  - Generate de Bruijn graph using spectrum
  - Find Eulerian Path and obtain sequence
  - Use original reads to resolve sequence ambiguities

Reads to Kmers

- Break up reads into kmers

Kmer Error Correction and Copy Number

- Assume Coverage is 30x

Kmers	Count	True Count
ACTAG	29	1
AGGAC	1	0
CGATC	61	2
TCAGA	31	1
TGATC	2	0

- Since coverage is 30x, each kmer that appears in 30 reads would show up in the genome once.
- Kmers that show up once or twice likely result from read errors.

Spectrum to Sequence

- De Bruijn Graph Approach
- Build Graph
- Find Eulerian Paths
- Use reads to "Correct graph/paths" and choose the best path.
  - De Bruijn Graph used only kmers - less information
  - Reads contain how the kmers go together
  - Paired reads contain even more information

Difficulties

- Exponential number of paths through the graph
- Errors in the spectrum/graph
  - No Eulerian Path
  - Many dead ends in graph

  - Unconnected graph

- Read information/paired end sequences