Genome Reconstruction: A Puzzle with a Billion Pieces

Phillip Compeau
Carnegie Mellon University
Computational Biology Department
Eternity II: The Highest-Stakes Puzzle in History

Courtesy: Matej Bat'ha
The Newspaper Problem

- Stack of NY Times, June 27, 2000
- Stack of NY Times, June 27, 2000 on a pile of dynamite
- This is just hypothetical

- BOOM
- So, what did the June 27, 2000 NY Times say?
What is a Genome?
The Molecular Structure of DNA

DNA’s Double Helix (1953)  DNA’s Molecular Structure
Nucleotides

**Nucleotide**: Half of one “rung” of DNA.

There are only four choices for the nucleic acid of a nucleotide:

1. Adenine (A)
2. Cytosine (C)
3. Guanine (G)—bonds to C
4. Thymine (T)—bonds to A

The order of nucleotides determines genetics!
DNA Sequencing

**Genome:** The nucleotide sequence read down one side of an organism’s chromosominal DNA.

A human genome has about 3 billion nucleotides. **Sequencing** a genome means “reading” all these letters.

**Key Point:** DNA is submicroscopic! How do we read something that we cannot see?
Introduction to Genome Sequencing
All Humans Are Genetically Very Similar

Any two humans share 99.9% of their genomes.

The 0.1% difference accounts for height, color, high cholesterol susceptibility, etc.
All Humans Are Genetically Very Similar

Any two humans share 99.9% of their genomes.

The 0.1% difference accounts for height, color, high cholesterol susceptibility, etc.
Species vs. Individual Sequencing

Species Sequencing: What is a species’s “consensus genome”?
Species Sequencing vs. Individual Sequencing

**Individual Sequencing:** What makes an individual unique?
Why Would We Sequence a Species’s Genome?
Why Would We Sequence an Individual’s Genome?

**Personalized Medicine:** Tailoring medical treatment to the individual based on their genome.

**2010:** First person whose life was saved because of genome sequencing.
Brief History of Genome Sequencing

**Late 1970s:** Walter Gilbert and Frederick Sanger develop independent sequencing methods.

**1980:** They share the Nobel Prize in Chemistry.

Still, their sequencing methods were too expensive for large genomes: with a $1 per nucleotide cost, it would cost $3 billion to sequence a human genome.
**Brief History of Genome Sequencing**

**1990**: The public Human Genome Project, headed by Francis Collins, aims to sequence the human genome.

**1997**: Craig Venter founds Celera Genomics, a private firm, with the same goal.
Brief History of Genome Sequencing

2000: The draft of the human genome is simultaneously completed by the (public) Human Genome Consortium and (private) Celera Genomics.
Brief History of Genome Sequencing

- **2000s**: Many more mammalian genomes are sequenced.
The Arrival of Personal Genomics

2010s: The market for sequencing machines takes off.

- Illumina reduces the cost of sequencing an individual human genome from $3 billion to $1,000.
- Beijing Genome Institute orders hundreds of sequencing machines.
- Companies offer partial genome sequencing for ~$500.
- UK funds genome sequencing for 100,000 citizens.
The Future of Genome Sequencing

The Near Future (?): Hopefully, sequencing an individual genome will soon become as routine as an X-ray.
The Newspaper Problem and Genome Sequencing
Returning to The Newspaper Problem

Given enough time, how might you solve this problem?
The Newspaper Problem as an “Overlap Puzzle”

The newspaper problem is not the same as a jigsaw puzzle:

- We have multiple copies of the same edition of a newspaper.
- Plus, some pieces of paper got blown to bits in the explosion.

Instead, we must use *overlapping* shreds of paper to reconstruct what the newspaper said.

This gives us a giant *overlap puzzle*. 
What Makes Genome Sequencing So Difficult?

When we read a book, we can read the entire book one letter at a time from beginning to end.

However, modern sequencing machines can only read short pieces of DNA (~250 nucleotides long), called reads.
Sequencing a Genome: Illustration

Multiple identical copies of a genome

Shatter the genome into reads

Sequence the reads

(Computational)

Assemble the genome using overlapping reads

(Lab)

What does genome sequencing remind you of?
Fragment Assembly = Overlap Puzzle!
Complications in Genome Assembly

1. DNA is **double-stranded** (and may consist of **multiple chromosomes**).

2. Reads have **imperfect “coverage”** of the underlying genome.

3. Sequencing machines are **error-prone**.
Simplifying Assumptions for Genome Assembly

1. DNA is single-stranded (and consists of a single circular chromosome, like bacteria).

2. Reads have perfect “coverage” of the underlying genome.

3. Sequencing machines are error-free.
Aim: Construct a shortest circular genome containing all our reads.

These reads have length 3, and the answer isn’t obvious.

How in the world would we solve this problem if we had a billion reads?

What method would you use?
From Biological Data to a Computational Problem

**Idea:** Look for reads having maximum overlap and build circular string.

Say we start with **AAT**. Which read should we pick next?

**AAT**
**Idea:** Look for reads having maximum overlap and build circular string.

Say we start with **AAT**. Which read should we pick next?

<table>
<thead>
<tr>
<th></th>
<th>TTT</th>
<th>TCT</th>
<th>TTG</th>
<th>TCG</th>
<th>TTC</th>
<th>TCC</th>
<th>TTA</th>
<th>TCA</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTT</td>
<td>GCT</td>
<td><strong>GTG</strong></td>
<td>GCG</td>
<td>GTC</td>
<td>GCC</td>
<td>GTC</td>
<td>GTA</td>
<td><strong>GCA</strong></td>
</tr>
<tr>
<td>CTT</td>
<td>CCT</td>
<td>CTG</td>
<td>CCG</td>
<td>CTC</td>
<td>CCC</td>
<td>CTC</td>
<td>CTA</td>
<td>CCA</td>
</tr>
<tr>
<td>ATT</td>
<td>ACT</td>
<td><strong>ATG</strong></td>
<td>ACG</td>
<td>ATC</td>
<td>ACC</td>
<td>ATC</td>
<td>ATA</td>
<td>ACA</td>
</tr>
<tr>
<td>TGT</td>
<td>TAT</td>
<td><strong>TGG</strong></td>
<td>TAG</td>
<td><strong>TGC</strong></td>
<td>TAC</td>
<td>TGA</td>
<td>TAA</td>
<td></td>
</tr>
<tr>
<td>GGT</td>
<td>GAT</td>
<td>GGG</td>
<td>GAG</td>
<td><strong>GGC</strong></td>
<td>GAC</td>
<td>GGA</td>
<td>GAA</td>
<td></td>
</tr>
<tr>
<td>CGT</td>
<td>CAT</td>
<td>CGG</td>
<td>CAG</td>
<td>CGC</td>
<td>CAC</td>
<td>CGA</td>
<td><strong>CAA</strong></td>
<td></td>
</tr>
<tr>
<td>AGT</td>
<td>AAT</td>
<td>AGG</td>
<td>AAG</td>
<td>AGC</td>
<td>AAC</td>
<td>AGA</td>
<td>AAA</td>
<td></td>
</tr>
</tbody>
</table>
**Idea:** Look for reads having maximum overlap and build circular string.

Now we have a *choice* between **TGC** and **TGG**.

<table>
<thead>
<tr>
<th></th>
<th>TTT</th>
<th>TCT</th>
<th>TTG</th>
<th>TCG</th>
<th>TTC</th>
<th>TCC</th>
<th>TTA</th>
<th>TCA</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTT</td>
<td></td>
<td></td>
<td><strong>GTG</strong></td>
<td>GCG</td>
<td>GTC</td>
<td>GCC</td>
<td>GTA</td>
<td><strong>GCA</strong></td>
</tr>
<tr>
<td>CTG</td>
<td></td>
<td></td>
<td></td>
<td>CCG</td>
<td>CTC</td>
<td>CCC</td>
<td>CTA</td>
<td><strong>CCA</strong></td>
</tr>
<tr>
<td>ATG</td>
<td></td>
<td></td>
<td></td>
<td>ACG</td>
<td>ATC</td>
<td>ACC</td>
<td>ATA</td>
<td><strong>ACA</strong></td>
</tr>
<tr>
<td>TGT</td>
<td></td>
<td><strong>TGG</strong></td>
<td>TAG</td>
<td><strong>TGC</strong></td>
<td>TAC</td>
<td>TGA</td>
<td>TAA</td>
<td></td>
</tr>
<tr>
<td>GGT</td>
<td></td>
<td></td>
<td></td>
<td>GAG</td>
<td><strong>GGC</strong></td>
<td>GAC</td>
<td>GGA</td>
<td>GAA</td>
</tr>
<tr>
<td>CGT</td>
<td></td>
<td></td>
<td></td>
<td>CAG</td>
<td>CGC</td>
<td>CAC</td>
<td>CGA</td>
<td><strong>CAA</strong></td>
</tr>
<tr>
<td>AGT</td>
<td><strong>AAT</strong></td>
<td></td>
<td>AGG</td>
<td>AAG</td>
<td>AGC</td>
<td>AAC</td>
<td>AGA</td>
<td>AAA</td>
</tr>
</tbody>
</table>
Idea: Look for reads having maximum overlap and build circular string.

If we get lucky with **TGC**, again we have a choice between **GCA** and **GCG**.

<table>
<thead>
<tr>
<th></th>
<th>TTT</th>
<th>TCT</th>
<th>TTG</th>
<th>TCG</th>
<th>TTC</th>
<th>TCC</th>
<th>TTA</th>
<th>TCA</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GCT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GTG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GCG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GTC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GCC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GTA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GCA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CTT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CTG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CTC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CTA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ATT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ATG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ATC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ATA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TGT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TAT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TGG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TAG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TGC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TAC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TGA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TAA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GGT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GAT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GGG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GAG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GGC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GAC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GGA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GAA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AGT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AAT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AGG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AAG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AGC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AAC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AGA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AAA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
50% of the human genome is made up of **repeats**: strings that appear multiple times with minor variations.

**Analogy**: The “Triazzle” contains lots of repeated figures, which makes it difficult to solve (even with just 16 pieces).

Repeated information is what makes Eternity II so difficult to solve!
Taking a Walk
The Bridges of Königsberg

The people of Königsberg, Prussia (present-day Kaliningrad, Russia) enjoyed taking walks.
The Bridges of Königsberg

They wondered if they could start at home, walk through the city, cross each bridge (blue) exactly once, and return home.

Can you see a solution?
The Bridges of Königsberg

1735: Leonhard Euler develops an approach to answer this question for any city, even for a “city” with a million islands.

We will soon see how Euler did this.
The Icosian Game

Over a century passes…

1857: Irish mathematician William Hamilton designs a game consisting of a board representing 20 “islands” connected by “bridges.”

Goal: find a walk that visits every island exactly once and returns back where it started.

Can you see a solution?
These Two Stories Have Something in Common ...

Find a walk that crosses every bridge once and returns home.

Find a walk that visits every island once and returns home.
But how do these problems relate to genome sequencing?

Assemble the genome using overlapping reads

AGAATATCA
GAGAATATC
TGAGAATAT
...TGAGAATATCA...
Hamiltonian and Eulerian Cycles
Königsberg Bridges Network

For the Königsberg Bridge Problem, we create a network:

- Nodes = 4 land masses of the city
- Edges = 7 bridges connecting land masses

We are looking for an **Eulerian cycle**: crossing every edge.

Now can you see a solution?
Hamiltonian Cycles

A **Hamiltonian cycle** touches each node exactly once and returns to its starting node.
Hamiltonian Cycles

A Hamiltonian cycle touches each node exactly once and returns to its starting node.
Hamiltonian Cycles

A Hamiltonian cycle touches each node exactly once and returns to its starting node.
Hamiltonian Cycles

A Hamiltonian cycle touches each node exactly once and returns to its starting node.

Note: We didn’t use every edge here.
Finding Eulerian and Hamiltonian Cycles

Given a network $G$, we now have two questions that we can program a computer to answer about $G$.

**Eulerian Cycle Problem (ECP):** Find an Eulerian cycle in $G$ or prove that $G$ does not have an Eulerian cycle.

**Hamiltonian Cycle Problem (HCP):** Find a Hamiltonian cycle in $G$ or prove that $G$ does not have a Hamiltonian cycle.
Eulerian Cycles

If there were a solution to the Königsberg Bridge Problem, then we could find an Eulerian cycle in this network.

However, no such cycle exists. Why?
Eulerian Cycles

If there were a solution to the Königsberg Bridge Problem, then we could find an Eulerian cycle in this network.

However, no such cycle exists. Why?

If we add two more edges, there will be such a cycle.
Euler’s Theorem
**Directed Networks**

**Directed Network**: A network in which each edge has a *direction* (represented by an arrow).

- Think of directed edges as “one-way bridges.”
Eulerian Cycles in Directed Networks

An **Eulerian cycle** in a directed network must travel down all the edges in the correct direction.

Does this network have an Eulerian cycle?
We can label each node with the number of edges in and the number of edges out.

A network is **balanced** if at each node, the number of incoming edges equals the number departing.

This network isn’t balanced.

However, adding some edges makes it balanced.
Euler’s Theorem

**Euler’s Theorem**: A directed network contains an Eulerian cycle when the network is balanced and “connected”.
Euler’s Theorem

**Euler’s Theorem**: A directed network contains an Eulerian cycle when the network is balanced and “connected”.

**Key point**: It is also possible to program a computer to quickly find an Eulerian cycle in a balanced, connected directed network …

... *even one with a billion edges!*
Difficult Computational Problems
So What’s the Big Deal?

“I thought computers were supermachines!”

Universal Pictures
So What’s the Big Deal?

“Computers don’t need 300-year old math to solve problems.”
So What’s the Big Deal?

“Aren’t computers going to take over the world anyway?”

MGM
So What’s the Big Deal?

So let’s examine the case of finding a Hamiltonian cycle…
Searching for an Efficient Solution for the HCP

**Key Point**: No one has ever found a similar efficient test if a network has a Hamiltonian cycle.

Of course, we could examine every possible ant walk through the network to solve the HCP.

However, this **brute force** approach is **infeasible**: there are more walks through the average network with just 1,000 nodes than there are atoms in the universe!
Searching for an Efficient Solution for the HCP

Attempting to solve the HCP is difficult.

“I can't find an efficient algorithm, I guess I'm just too dumb.”

From Garey and Johnson. *Computers and Intractability*, 1979
Searching for an Efficient Solution for the HCP

Attempting to solve the HCP is difficult.

The hope is that you could verify that you failed because an efficient method solving the HCP doesn’t exist.

“\textit{I can't find an efficient algorithm, because no such algorithm is possible.}”

From Garey and Johnson. \textit{Computers and Intractability}. 1979
Searching for an Efficient Solution for the HCP

Attempting to solve the HCP is difficult.

The hope is that you could verify that you failed because an efficient method solving the HCP doesn’t exist.

The present state of affairs is somewhere in between.

“I can't find an efficient algorithm, but neither can all these smart people.”

From Garey and Johnson. *Computers and Intractability*. 1979
High-Stakes Mathematics

The question of whether the HCP can be solved efficiently is one of seven Millennium Problems (Clay Institute).

Find an efficient algorithm for the HCP, or demonstrate that no such algorithm exists, and you will win $1 million.

Only one problem has been solved, by Grigory Perelman, who refused the prize.
From Euler and Hamilton to Assembling Genomes
Returning to our Toy Example

**Goal:** Use *overlapping* DNA reads in order to reconstruct the original genome.

**Idea:** Let’s construct a *network* that represents the overlap information in our reads.
Constructing a Network from Reads

Create a node for every read.
Constructing a Network from Reads

Create a node for every read.
Constructing a Network from Reads

Create a node for every read.

- **Prefix**: First 2 nucleotides of a read (CAA)
- **Suffix**: Last 2 nucleotides of a read (CAA)

Different 3-mers may share a prefix/suffix: ATG, TGA, CTG

ATG  CGT  GGC  AAT  GTG  TGG  TGC  CAA  GCA  GCG
Constructing a Network from Reads

As for the edges, connect node $v$ to node $w$ with a directed edge if the suffix of $v$ matches the prefix of $w$. 
Constructing a Network from Reads

As for the edges, connect node $v$ to node $w$ with a *directed edge* if the suffix of $v$ matches the prefix of $w$. 
Constructing a Network from Reads

As for the edges, connect node \( v \) to node \( w \) with a directed edge if the suffix of \( v \) matches the prefix of \( w \).
Constructing a Network from Reads

As for the edges, connect node $v$ to node $w$ with a directed edge if the suffix of $v$ matches the prefix of $w$. 
Constructing a Network from Reads

As for the edges, connect node \( v \) to node \( w \) with a *directed edge* if the suffix of \( v \) matches the prefix of \( w \).
Constructing a Network from Reads

As for the edges, connect node $v$ to node $w$ with a directed edge if the suffix of $v$ matches the prefix of $w$. 

ATG  CGT  GGC  AAT  GTG  TGG  TGC  CAA  GCA  GCG
Constructing a Network from Reads

As for the edges, connect node $v$ to node $w$ with a directed edge if the suffix of $v$ matches the prefix of $w$. 
Constructing a Network from Reads

As for the edges, connect node $v$ to node $w$ with a directed edge if the suffix of $v$ matches the prefix of $w$. 
Constructing a Network from Reads

As for the edges, connect node $v$ to node $w$ with a directed edge if the suffix of $v$ matches the prefix of $w$. 

ATG CGT GGC AAT GTG TGG TGC CAA GCA GCG
Constructing a Network from Reads

As for the edges, connect node $v$ to node $w$ with a directed edge if the suffix of $v$ matches the prefix of $w$. 

ATG  CGT  GGC  AAT  GTG  TGG  TGC  CAA  GCA  GCG
Constructing a Network from Reads

As for the edges, connect node $v$ to node $w$ with a *directed edge* if the suffix of $v$ matches the prefix of $w$. 
Here we have a Hamiltonian cycle:
A Hamiltonian Cycle Recovers the Genome

Here we have a Hamiltonian cycle:

ATG
A Hamiltonian Cycle Recovers the Genome

Here we have a Hamiltonian cycle:

ATG $\rightarrow$ TGG

[Diagram showing a Hamiltonian cycle with nodes ATG, CGT, GGC, AAT, GTG, TGG, TGC, CAA, GCA, and GCG with arrows indicating the cycle.]
A Hamiltonian Cycle Recovers the Genome

Here we have a Hamiltonian cycle:

\[
\text{ATG} \rightarrow \text{TGG} \rightarrow \text{GGC}
\]
Here we have a Hamiltonian cycle:

\[ \text{ATG} \rightarrow \text{TGG} \rightarrow \text{GGC} \rightarrow \text{GCG} \]
A Hamiltonian Cycle Recovers the Genome

Here we have a Hamiltonian cycle:

\[ \text{ATG} \rightarrow \text{TGG} \rightarrow \text{GGC} \rightarrow \text{GCG} \rightarrow \text{CGT} \]
A Hamiltonian Cycle Recovers the Genome

Here we have a Hamiltonian cycle:

ATG → TGG → GGC → GCG → CGT → GTG
A Hamiltonian Cycle Recovers the Genome

Here we have a Hamiltonian cycle:

ATG → TGG → GGC → GCG → CGT → GTG → TGC
A Hamiltonian Cycle Recovers the Genome

Here we have a Hamiltonian cycle:

ATG $\rightarrow$ TGG $\rightarrow$ GGC $\rightarrow$ GCG $\rightarrow$ CGT $\rightarrow$ GTG $\rightarrow$
TGC $\rightarrow$ GCA
A Hamiltonian Cycle Recovers the Genome

Here we have a Hamiltonian cycle:

ATG $\rightarrow$ TGG $\rightarrow$ GGC $\rightarrow$ GCG $\rightarrow$ CGT $\rightarrow$ GTG $\rightarrow$
TGC $\rightarrow$ GCA $\rightarrow$ CAA
A Hamiltonian Cycle Recovers the Genome

Here we have a Hamiltonian cycle:

ATG → TGG → GGC → GCG → CGT → GTG → TGC → GCA → CAA → AAT
A Hamiltonian Cycle Recovers the Genome

Here we have a Hamiltonian cycle:

\[
\text{ATG} \rightarrow \text{TGG} \rightarrow \text{GGC} \rightarrow \text{GCG} \rightarrow \text{CGT} \rightarrow \text{GTG} \rightarrow \\
\text{TGC} \rightarrow \text{GCA} \rightarrow \text{CAA} \rightarrow \text{AAT} \rightarrow \text{ATG}
\]
A Hamiltonian Cycle Recovers the Genome

Here we have a Hamiltonian cycle:

\[
\begin{align*}
\text{ATG} & \to \text{TGG} & \to \text{GGC} & \to \text{GCG} & \to \text{CGT} & \to \text{GTG} & \to \\
\text{TGC} & \to \text{GCA} & \to \text{CAA} & \to \text{AAT} & \to \text{ATG}
\end{align*}
\]
A Hamiltonian Cycle Recovers the Genome

Here we have a Hamiltonian cycle:

\[
\begin{align*}
\text{ATG} & \rightarrow \text{TGG} \rightarrow \text{GGC} \rightarrow \text{GCG} \rightarrow \text{CGT} \rightarrow \text{GTG} \rightarrow \\
\text{TGC} & \rightarrow \text{GCA} \rightarrow \text{CAA} \rightarrow \text{AAT} \rightarrow \text{ATG} \\
\text{ATG} & \\
\end{align*}
\]

Genome: ATG
A Hamiltonian Cycle Recovers the Genome

Here we have a Hamiltonian cycle:

\[
\begin{align*}
ATG & \rightarrow TGG & \rightarrow GGC & \rightarrow GCG & \rightarrow CGT & \rightarrow GTG & \rightarrow \\
TGC & \rightarrow GCA & \rightarrow CAA & \rightarrow AAT & \rightarrow ATG
\end{align*}
\]

Genome: ATGG
A Hamiltonian Cycle Recovers the Genome

Here we have a Hamiltonian cycle:

\[
\begin{align*}
\text{ATG} & \rightarrow \text{TGG} \rightarrow \text{GGC} \rightarrow \text{GCG} \rightarrow \text{CGT} \rightarrow \text{GTG} \rightarrow \\
\text{TGC} & \rightarrow \text{GCA} \rightarrow \text{CAA} \rightarrow \text{AAT} \rightarrow \text{ATG}
\end{align*}
\]

Genome: \text{ATGGC}
A Hamiltonian Cycle Recovers the Genome

Here we have a Hamiltonian cycle:

\[ \text{ATG} \rightarrow \text{TGG} \rightarrow \text{GGC} \rightarrow \text{GCG} \rightarrow \text{CGT} \rightarrow \text{GTG} \rightarrow \text{TGC} \rightarrow \text{GCA} \rightarrow \text{CAA} \rightarrow \text{AAT} \rightarrow \text{ATG} \]

Genome: \text{ATGGCG}
A Hamiltonian Cycle Recovers the Genome

Here we have a Hamiltonian cycle:

ATG → TGG → GGC → GCG → CGT → GTG → TGC → GCA → CAA → AAT → ATG

Genome: ATGGCGT
Here we have a Hamiltonian cycle:

ATG → TGG → GGC → GCG → CGT → GTG →
TGC → GCA → CAA → AAT → ATG

Genome: ATGGCGTG
A Hamiltonian Cycle Recovers the Genome

Here we have a Hamiltonian cycle:

ATG → TGG → GGC → GCG → CGT → GTG → TG
TGC → GCA → CAA → AAT → ATG

Genome: ATGGCGTGC
A Hamiltonian Cycle Recovers the Genome

Here we have a Hamiltonian cycle:

\[
\begin{align*}
ATG & \rightarrow \ TGG & \rightarrow \ GGC & \rightarrow \ GCG & \rightarrow \ CGT & \rightarrow \ GTG & \rightarrow \\
TGC & \rightarrow \ GCA & \rightarrow \ CAA & \rightarrow \ AAT & \rightarrow \ ATG
\end{align*}
\]

Genome: ATGGCGTGCA
A Hamiltonian Cycle Recovers the Genome

Here we have a Hamiltonian cycle:

ATG → TGG → GGC → GCG → CGT → GTG → TGC → GCA → CAA → AAT → ATG

Genome: ATGGCGTGCAA
A Hamiltonian Cycle Recovers the Genome

Here we have a Hamiltonian cycle:

\[
\begin{align*}
ATG & \rightarrow TGG \\
TGG & \rightarrow GGC \\
GGC & \rightarrow GCG \\
GCG & \rightarrow CGT \\
CGT & \rightarrow GTG \\
GTG & \rightarrow TGC \\
TGC & \rightarrow GCA \\
GCA & \rightarrow CAA \\
CAA & \rightarrow AAT \\
AAT & \rightarrow ATG
\end{align*}
\]

Genome: \textcolor{red}{ATGGCGTGCAAT}
A Hamiltonian Cycle Recovers the Genome

Here we have a Hamiltonian cycle:

ATG → TGG → GGC → GCG → CGT → GTG →
TGC → GCA → CAA → AAT → ATG

Genome: ATGGCGTGCAATG
A Hamiltonian Cycle Recovers the Genome

Here we have a Hamiltonian cycle:

\[
\begin{align*}
\text{ATG} & \rightarrow \text{TGG} & \rightarrow \text{GGC} & \rightarrow \text{GCG} & \rightarrow \text{CGT} & \rightarrow \text{GTG} & \rightarrow \\
\text{TGC} & \rightarrow \text{GCA} & \rightarrow \text{CAA} & \rightarrow \text{AAT} & \rightarrow & \text{ATG}
\end{align*}
\]

Genome: \text{ATGGCGTGCAATG}
A Hamiltonian Cycle Recovers the Genome

Here we have a Hamiltonian cycle:

\[
\begin{align*}
ATG & \rightarrow TGG \\
TGG & \rightarrow GGC \\
GGC & \rightarrow GCG \\
GCG & \rightarrow CGT \\
CGT & \rightarrow GTG \\
GTG & \rightarrow TGC \\
TGC & \rightarrow GCA \\
GCA & \rightarrow CAA \\
CAA & \rightarrow AAT \\
AAT & \rightarrow ATG \\
\end{align*}
\]

Genome: \text{ATGGCGTGCAATG}
A Hamiltonian Cycle Recovers the Genome

Here we have a Hamiltonian cycle:

ATG → TGG → GGC → GCG → CGT → GTG → TGC → GCA → CAA → AAT → ATG

Genome: ATGGCGTGCAATG
A Hamiltonian Cycle Recovers the Genome

Here we have a Hamiltonian cycle:

ATG → TGG → GGC → GCG → CGT → GTG → TGC → GCA → CAA → AAT → ATG

Genome: ATGGCGTGCA
A Hamiltonian Cycle Recovers the Genome

Here we have a Hamiltonian cycle:

ATG → TGG → GGC → GCG → CGT → GTG → TGC → GCA → CAA → AAT → ATG

Genome: ATGGCGTGCA

What is wrong with this approach?
The Problem with Our Network

Ultimately, we must solve the HCP on our network (millions of nodes) in order to obtain a candidate genome …
Second Try: Assign Reads to *Edges*

Form a different network as follows:

- Create a node for each *distinct* prefix/suffix from reads.

<table>
<thead>
<tr>
<th>Reads</th>
<th>GTG</th>
<th>TGC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GCG</td>
<td>GGC</td>
</tr>
<tr>
<td></td>
<td>GCA</td>
<td>CGT</td>
</tr>
<tr>
<td></td>
<td>ATG</td>
<td>CAA</td>
</tr>
<tr>
<td></td>
<td>TGG</td>
<td>AAT</td>
</tr>
</tbody>
</table>
Second Try: Assign Reads to *Edges*

Form a different network as follows:

- Create a node for each *distinct* prefix/suffix from reads.

<table>
<thead>
<tr>
<th>Reads</th>
<th>GTG</th>
<th>TGC</th>
<th>GCG</th>
<th>GGC</th>
<th>GCA</th>
<th>CGT</th>
<th>ATG</th>
<th>CAA</th>
<th>TGG</th>
<th>AAT</th>
</tr>
</thead>
</table>
Second Try: Assign Reads to *Edges*

Form a different network as follows:

- Create a node for each *distinct* prefix/suffix from reads.

<table>
<thead>
<tr>
<th>Reads</th>
<th>GTG</th>
<th>TGC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GCG</td>
<td>GGC</td>
</tr>
<tr>
<td></td>
<td>GCA</td>
<td>CGT</td>
</tr>
<tr>
<td></td>
<td>ATG</td>
<td>CAA</td>
</tr>
<tr>
<td></td>
<td>TGG</td>
<td>AAT</td>
</tr>
</tbody>
</table>
Second Try: Assign Reads to *Edges*

Form a different network as follows:

- Create a node for each *distinct* prefix/suffix from reads.
Second Try: Assign Reads to *Edges*

Form a different network as follows:

- Create a node for each *distinct* prefix/suffix from reads.

<table>
<thead>
<tr>
<th>Reads</th>
<th>GTG</th>
<th>TGC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GCG</td>
<td>GGC</td>
</tr>
<tr>
<td></td>
<td>GCA</td>
<td>CGT</td>
</tr>
<tr>
<td></td>
<td>ATG</td>
<td>CAA</td>
</tr>
<tr>
<td></td>
<td>TGG</td>
<td>AAT</td>
</tr>
</tbody>
</table>

\[ \text{GT} \quad \text{CG} \]

\[ \text{TG} \quad \text{GC} \]
Second Try: Assign Reads to *Edges*

Form a different network as follows:

- Create a node for each *distinct* prefix/suffix from reads.

<table>
<thead>
<tr>
<th>Reads</th>
<th>GTG</th>
<th>TGC</th>
<th>GCG</th>
<th>GGC</th>
<th>GCA</th>
<th>CGT</th>
<th>ATG</th>
<th>CAA</th>
<th>TGG</th>
<th>AAT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>GT</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>CG</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>TG</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>GC</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Create a node for each distinct prefix/suffix from reads.
Second Try: Assign Reads to *Edges*

Form a different network as follows:

- Create a node for each distinct prefix/suffix from reads.
Second Try: Assign Reads to *Edges*

Form a different network as follows:

- Create a node for each *distinct* prefix/suffix from reads.

### Reads

<table>
<thead>
<tr>
<th>Reads</th>
<th>GTG</th>
<th>TGC</th>
<th>GCG</th>
<th>GGC</th>
<th>GCA</th>
<th>CGT</th>
<th>ATG</th>
<th>CAA</th>
<th>TGG</th>
<th>AAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTG</td>
<td>TGC</td>
<td>GCG</td>
<td>GGC</td>
<td>GCA</td>
<td>CGT</td>
<td>ATG</td>
<td>CAA</td>
<td>TGG</td>
<td>AAT</td>
<td></td>
</tr>
<tr>
<td>CG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Nodes

- **GT**
- **CG**
- **AT**
- **TG**
- **GC**
- **CA**
Second Try: Assign Reads to *Edges*

Form a different network as follows:

- Create a node for each *distinct* prefix/suffix from reads.

<table>
<thead>
<tr>
<th>Reads</th>
<th>GTG</th>
<th>TGC</th>
<th>GCG</th>
<th>GGC</th>
<th>GCA</th>
<th>CGT</th>
<th>AT</th>
<th>TG</th>
<th>GC</th>
<th>CA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Second Try: Assign Reads to *Edges*

Form a different network as follows:

- Create a node for each *distinct* prefix/suffix from reads.

<table>
<thead>
<tr>
<th>Reads</th>
<th>GTG</th>
<th>TGC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GCG</td>
<td>GGC</td>
</tr>
<tr>
<td></td>
<td>GCA</td>
<td>CGT</td>
</tr>
<tr>
<td></td>
<td>ATG</td>
<td>CAA</td>
</tr>
<tr>
<td></td>
<td>TGG</td>
<td>AAT</td>
</tr>
</tbody>
</table>

- GT
- CG
- AT
- TG
- GC
- CA
Second Try: Assign Reads to *Edges*

Form a different network as follows:

- Create a node for each *distinct* prefix/suffix from reads.
Second Try: Assign Reads to *Edges*

Form a different network as follows:

- Create a node for each *distinct* prefix/suffix from reads.

<table>
<thead>
<tr>
<th>Reads</th>
<th>GTG</th>
<th>TGC</th>
<th>GCG</th>
<th>GGC</th>
<th>GCA</th>
<th>CGT</th>
<th>ATG</th>
<th>CAA</th>
<th>TGG</th>
<th>AAT</th>
</tr>
</thead>
</table>

- **GT**
- **CG**
- **GG**
- **AT**
- **TG**
- **GC**
- **CA**
Second Try: Assign Reads to *Edges*

Form a different network as follows:

- Create a node for each *distinct* prefix/suffix from reads.

<table>
<thead>
<tr>
<th>Reads</th>
<th>GTG</th>
<th>TGC</th>
<th>AT</th>
<th>TG</th>
<th>GC</th>
<th>CA</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCG</td>
<td>GGC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GCA</td>
<td>CGT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ATG</td>
<td>CAA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TGG</td>
<td>AAT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Second Try: Assign Reads to *Edges*

Form a different network as follows:

- Create a node for each *distinct* prefix/suffix from reads.

```

<table>
<thead>
<tr>
<th>Reads</th>
<th>GTG</th>
<th>TGC</th>
<th>GCG</th>
<th>GGC</th>
<th>GCA</th>
<th>CGT</th>
<th>AT</th>
<th>TG</th>
<th>GC</th>
<th>CA</th>
</tr>
</thead>
<tbody>
<tr>
<td>GT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AAT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TGG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

- `GT` - `CG` - `GG`
Second Try: Assign Reads to *Edges*

Form a different network as follows:

- Create a node for each *distinct* prefix/suffix from reads.

<table>
<thead>
<tr>
<th>Reads</th>
<th>GTG</th>
<th>TGC</th>
<th>GCG</th>
<th>GCC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AT</td>
<td>TG</td>
<td>GC</td>
<td>CA</td>
</tr>
</tbody>
</table>

Note: The nodes GT, CG, GG, AT, TG, GC, and CA represent distinct prefix/suffix pairs from the reads provided.
Second Try: Assign Reads to *Edges*

Form a different network as follows:

- Create a node for each *distinct* prefix/suffix from reads.

<table>
<thead>
<tr>
<th>Reads</th>
<th>GTG</th>
<th>TGC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GCG</td>
<td>GGC</td>
</tr>
<tr>
<td></td>
<td>GCA</td>
<td>CGT</td>
</tr>
<tr>
<td></td>
<td>ATG</td>
<td>CAA</td>
</tr>
<tr>
<td></td>
<td>TGG</td>
<td>AAT</td>
</tr>
</tbody>
</table>

Nodes:
- GT
- CG
- GG
- AT
- TG
- GC
- CA
Second Try: Assign Reads to *Edges*

Form a different network as follows:

- Create a node for each *distinct* prefix/suffix from reads.

<table>
<thead>
<tr>
<th>Reads</th>
<th>GTG</th>
<th>TGC</th>
<th>GCG</th>
<th>GGC</th>
<th>GCA</th>
<th>CGT</th>
<th>AT</th>
<th>TG</th>
<th>GC</th>
<th>CA</th>
</tr>
</thead>
<tbody>
<tr>
<td>GT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Second Try: Assign Reads to *Edges*

Form a different network as follows:

- Create a node for each *distinct* prefix/suffix from reads.
Second Try: Assign Reads to *Edges*

Form a different network as follows:

- Create a node for each *distinct* prefix/suffix from reads.

<table>
<thead>
<tr>
<th>Reads</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>GTG</td>
<td>TGC</td>
</tr>
<tr>
<td>GCG</td>
<td>GGC</td>
</tr>
<tr>
<td>GCA</td>
<td>CGT</td>
</tr>
<tr>
<td>ATG</td>
<td>CAA</td>
</tr>
<tr>
<td>TGG</td>
<td>AAT</td>
</tr>
</tbody>
</table>

- GT
- CG
- GG
- GC
- CA
- AT
- TG
- AA
Second Try: Assign Reads to Edges

Form a different network as follows:

- Create a node for each distinct prefix/suffix from reads.

<table>
<thead>
<tr>
<th>Reads</th>
<th>GTG</th>
<th>TGC</th>
<th>GCG</th>
<th>GGC</th>
<th>GCA</th>
<th>CGT</th>
<th>ATG</th>
<th>CAA</th>
<th>TGG</th>
<th>AAT</th>
</tr>
</thead>
</table>
Second Try: Assign Reads to *Edges*

Form a different network as follows:

- Create a node for each *distinct* prefix/suffix from reads.
Second Try: Assign Reads to *Edges*

Form a different network as follows:

- Create a node for each *distinct* prefix/suffix from reads.

Reads

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>GTG</td>
<td>TGC</td>
<td></td>
</tr>
<tr>
<td>GCG</td>
<td>GGC</td>
<td></td>
</tr>
<tr>
<td>GCA</td>
<td>CGT</td>
<td></td>
</tr>
<tr>
<td>ATG</td>
<td>CAA</td>
<td></td>
</tr>
<tr>
<td>TGG</td>
<td>AAT</td>
<td></td>
</tr>
</tbody>
</table>

Nodes:

- GT
- CG
- GG
- GC
- CA
- AT
- TG
- AA
Second Try: Assign Reads to *Edges*

Form a different network as follows:

- Create a node for each *distinct* prefix/suffix from reads.
- Connect node $v$ to node $w$ with a directed edge if there is a read whose prefix is $v$ and whose suffix is $w$.

<table>
<thead>
<tr>
<th>Reads</th>
<th>GTG</th>
<th>TGC</th>
<th>GCG</th>
<th>GGC</th>
<th>GCA</th>
<th>CGT</th>
<th>AT</th>
<th>TG</th>
<th>GC</th>
<th>CA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Second Try: Assign Reads to *Edges*

Form a different network as follows:

- Create a node for each distinct prefix/suffix from reads.
- Connect node $v$ to node $w$ with a directed edge if there is a read whose prefix is $v$ and whose suffix is $w$.

<table>
<thead>
<tr>
<th>Reads</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>GTG</td>
<td>TGC</td>
</tr>
<tr>
<td>GCG</td>
<td>GGC</td>
</tr>
<tr>
<td>GCA</td>
<td>CGT</td>
</tr>
<tr>
<td>ATG</td>
<td>CAA</td>
</tr>
<tr>
<td>TGG</td>
<td>AAT</td>
</tr>
</tbody>
</table>

---

Diagram:
- Nodes: GT, CG, AT, TG, GC, CA, GG
- Directed edges: GT → TG, TG → GC, GC → CA, CA → AA
Second Try: Assign Reads to *Edges*

Form a different network as follows:

- Create a node for each *distinct* prefix/suffix from reads.
- Connect node \( v \) to node \( w \) with a directed edge if there is a read whose prefix is \( v \) and whose suffix is \( w \).

<table>
<thead>
<tr>
<th>Reads</th>
<th>GTG</th>
<th>GCG</th>
<th>GCA</th>
<th>ATG</th>
<th>TGG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TGC</td>
<td>GGC</td>
<td>CGT</td>
<td>CAA</td>
<td>AAT</td>
</tr>
</tbody>
</table>

\( \text{GT} \) \rightarrow \text{GTG} \rightarrow \text{TG} \rightarrow \text{AA} \quad \text{CA} \rightarrow \text{GC} \rightarrow \text{GCG} \rightarrow \text{GG} \rightarrow \text{GT} \
Second Try: Assign Reads to *Edges*

Form a different network as follows:

- Create a node for each *distinct* prefix/suffix from reads.
- Connect node $v$ to node $w$ with a directed edge if there is a read whose prefix is $v$ and whose suffix is $w$.

<table>
<thead>
<tr>
<th>Reads</th>
<th>GTG</th>
<th>TGC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GCG</td>
<td>GGC</td>
</tr>
<tr>
<td></td>
<td>GCA</td>
<td>CGT</td>
</tr>
<tr>
<td></td>
<td>ATG</td>
<td>CAA</td>
</tr>
<tr>
<td></td>
<td>TGG</td>
<td>AAT</td>
</tr>
</tbody>
</table>
Second Try: Assign Reads to Edges

Form a different network as follows:

- Create a node for each distinct prefix/suffix from reads.
- Connect node $v$ to node $w$ with a directed edge if there is a read whose prefix is $v$ and whose suffix is $w$.

<table>
<thead>
<tr>
<th>Reads</th>
<th>AT</th>
<th>GCG</th>
<th>GCA</th>
<th>ATG</th>
<th>TGG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ATG</td>
<td>TGC</td>
<td>GGC</td>
<td>GCA</td>
<td>CGT</td>
</tr>
<tr>
<td></td>
<td>ATG</td>
<td>CAA</td>
<td>CGT</td>
<td>CAA</td>
<td>AAT</td>
</tr>
</tbody>
</table>
Second Try: Assign Reads to *Edges*

Form a different network as follows:

- Create a node for each distinct prefix/suffix from reads.
- Connect node \( v \) to node \( w \) with a directed edge if there is a read whose prefix is \( v \) and whose suffix is \( w \).

### Reads

<table>
<thead>
<tr>
<th></th>
<th>GTG</th>
<th>TGC</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCG</td>
<td>GGC</td>
<td></td>
</tr>
<tr>
<td>GCA</td>
<td>CGT</td>
<td></td>
</tr>
<tr>
<td>ATG</td>
<td>CAA</td>
<td></td>
</tr>
<tr>
<td>TGG</td>
<td>AAT</td>
<td></td>
</tr>
</tbody>
</table>

### Graph

- Node `GT` connected by directed edge to `TG`
- Node `GTG` connected by directed edge to `TGG`
- Node `GG` connected by directed edge to `TGG`
- Node `CG` connected by directed edge to `GCG`
- Node `GC` connected by directed edge to `GCA`
- Node `CA` connected by directed edge to `AA`
- Node `AT` connected by directed edge to `TG`
Second Try: Assign Reads to *Edges*

Form a different network as follows:

- Create a node for each *distinct* prefix/suffix from reads.
- Connect node \( v \) to node \( w \) with a directed edge if there is a read whose prefix is \( v \) and whose suffix is \( w \).

<table>
<thead>
<tr>
<th>Reads</th>
<th>GTG</th>
<th>TGC</th>
<th>GCG</th>
<th>GGC</th>
<th>GCA</th>
<th>CGT</th>
<th>ATG</th>
<th>CAA</th>
<th>TGG</th>
<th>AAT</th>
</tr>
</thead>
</table>

![Diagram of network with nodes and edges representing reads prefixed by distinct sequences]
Second Try: Assign Reads to *Edges*

Form a different network as follows:

- Create a node for each *distinct* prefix/suffix from reads.
- Connect node $v$ to node $w$ with a directed edge if there is a read whose prefix is $v$ and whose suffix is $w$.

<table>
<thead>
<tr>
<th>Reads</th>
<th>GTG</th>
<th>TGC</th>
<th>GCG</th>
<th>GGC</th>
<th>ATG</th>
<th>CAA</th>
<th>TGG</th>
<th>AAT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AT</td>
<td>TG</td>
<td>TGC</td>
<td>GG</td>
<td>TGG</td>
<td>GC</td>
<td>GCA</td>
<td>CA</td>
</tr>
</tbody>
</table>

Diagram:

- **Nodes:** AT, TG, GC, CG, CA, AA
- **Edges:** GT, GC, CA, ATG, TGC, TGG, GGC, GCG
Second Try: Assign Reads to Edges

Form a different network as follows:

- Create a node for each distinct prefix/suffix from reads.
- Connect node $v$ to node $w$ with a directed edge if there is a read whose prefix is $v$ and whose suffix is $w$.

**Reads**

<table>
<thead>
<tr>
<th>Reads</th>
<th>GTG</th>
<th>TGC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GCG</td>
<td>GGC</td>
</tr>
<tr>
<td></td>
<td>GCA</td>
<td>CGT</td>
</tr>
<tr>
<td></td>
<td>ATG</td>
<td>CAA</td>
</tr>
<tr>
<td></td>
<td>TGG</td>
<td>AAT</td>
</tr>
</tbody>
</table>
Second Try: Assign Reads to *Edges*

Form a different network as follows:

- Create a node for each *distinct* prefix/suffix from reads.
- Connect node $v$ to node $w$ with a directed edge if there is a read whose prefix is $v$ and whose suffix is $w$.

**Reads**

<table>
<thead>
<tr>
<th>GTG</th>
<th>TGC</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCG</td>
<td>GGC</td>
</tr>
<tr>
<td>GCA</td>
<td>CGT</td>
</tr>
<tr>
<td>ATG</td>
<td>CAA</td>
</tr>
<tr>
<td>TGG</td>
<td>AAT</td>
</tr>
</tbody>
</table>

**Graph**

- $ATG \rightarrow TG \rightarrow TGC \rightarrow GC \rightarrow GCA \rightarrow CA$
- $GT \rightarrow CG \rightarrow GG \rightarrow GCG$
- $GTG \rightarrow TGG \rightarrow GGC$
- $GG \rightarrow GTG$
- $CG \rightarrow CGT$

*Diagram showing the network with labeled nodes and edges.*
Second Try: Assign Reads to *Edges*

Form a different network as follows:

- Create a node for each *distinct* prefix/suffix from reads.
- Connect node $v$ to node $w$ with a directed edge if there is a read whose prefix is $v$ and whose suffix is $w$.

<table>
<thead>
<tr>
<th>Reads</th>
<th>GTG</th>
<th>TGC</th>
<th>GCG</th>
<th>GGC</th>
<th>GCA</th>
<th>CGT</th>
<th>ATG</th>
<th>CAA</th>
<th>TGG</th>
<th>AAT</th>
</tr>
</thead>
</table>

![Diagram of network with nodes and edges labeled with sequence pairs]
Second Try: Assign Reads to *Edges*

Form a different network as follows:

- Create a node for each *distinct* prefix/suffix from reads.
- Connect node $v$ to node $w$ with a directed edge if there is a read whose prefix is $v$ and whose suffix is $w$.

<table>
<thead>
<tr>
<th>Reads</th>
<th>GTG</th>
<th>TGC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GCG</td>
<td>GGC</td>
</tr>
<tr>
<td></td>
<td>GCA</td>
<td>CGT</td>
</tr>
<tr>
<td></td>
<td>ATG</td>
<td>CAA</td>
</tr>
<tr>
<td></td>
<td>TGG</td>
<td>AAT</td>
</tr>
</tbody>
</table>

Diagram:

- Node GT connects to CGT.
- Node CG connects to GT.
- Node GTG connects to TGG and GGC.
- Node GG connects to GGC and GCA.
- Node GC connects to GCA.
- Node CA connects to CAA.
- Node AT connects to ATG and TGC.
- Node TG connects to TGC and GC.
- Node AA connects to AAT.
An *Eulerian* Cycle Recovers the Genome

We have an Eulerian cycle in this network:
An Eulerian Cycle Recovers the Genome

We have an Eulerian cycle in this network:

ATG
An *Eulerian* Cycle Recovers the Genome

We have an Eulerian cycle in this network:

**ATG → TGG →**
An *Eulerian* Cycle Recovers the Genome

We have an Eulerian cycle in this network:

\[
\text{ATG} \rightarrow \text{TGG} \rightarrow \text{GGC}
\]
An *Eulerian* Cycle Recovers the Genome

We have an Eulerian cycle in this network:

\[
\text{ATG} \rightarrow \text{TGG} \rightarrow \text{GGC} \rightarrow \text{GCG}
\]
An *Eulerian* Cycle Recovers the Genome

We have an Eulerian cycle in this network:

\[
\text{ATG} \rightarrow \text{TGG} \rightarrow \text{GGC} \rightarrow \text{GCG} \rightarrow \text{CGT}
\]
An Eulerian Cycle Recovers the Genome

We have an Eulerian cycle in this network:

\[
\text{ATG} \rightarrow \text{TGG} \rightarrow \text{GGC} \rightarrow \text{GCG} \rightarrow \text{CGT} \rightarrow \text{GTG}
\]
An *Eulerian* Cycle Recovers the Genome

We have an Eulerian cycle in this network:

\[
\text{ATG} \rightarrow \text{TGG} \rightarrow \text{GGC} \rightarrow \text{GCG} \rightarrow \text{CGT} \rightarrow \text{GTG} \rightarrow \text{TGC}
\]
An *Eulerian* Cycle Recovers the Genome

We have an Eulerian cycle in this network:

\[
\begin{align*}
\text{ATG} & \rightarrow \text{TGG} & \rightarrow \text{GGC} & \rightarrow \text{GCG} & \rightarrow \text{CGT} & \rightarrow \text{GTG} & \rightarrow \\
\text{TGC} & \rightarrow \text{GCA}
\end{align*}
\]
An Eulerian Cycle Recovers the Genome

We have an Eulerian cycle in this network:

ATG → TGG → GGC → GCG → CGT → GTG → TGC → GCA → CAA
An Eulerian Cycle Recovers the Genome

We have an Eulerian cycle in this network:

ATG → TGG → GGC → GCG → CGT → GTG →
TGC → GCA → CAA → AAT
An *Eulerian* Cycle Recovers the Genome

We have an Eulerian cycle in this network:

- ATG → TGG → GGC → GCG → CGT → GTG → TGC → GCA → CAA → AAT
An *Eulerian* Cycle Recovers the Genome

We have an Eulerian cycle in this network:

- ATG $\rightarrow$ TGG $\rightarrow$ GGC $\rightarrow$ GCG $\rightarrow$ CGT $\rightarrow$ GTG $\rightarrow$ TGC $\rightarrow$ GCA $\rightarrow$ CAA $\rightarrow$ AAT

- This is the same sequence of reads that we found before!
An Eulerian Cycle Recovers the Genome

We have an Eulerian cycle in this network:

ATG → TGG → GGC → GCG → CGT → GTG →
TGC → GCA → CAA → AAT

• This is the same sequence of reads that we found before!
• Thus, we will obtain the same sequenced genome as before.

Genome: ATGGCGTGCAATG
An *Eulerian* Cycle Recovers the Genome

We have an Eulerian cycle in this network:

\[
\begin{align*}
ATG & \rightarrow \ TGG & \rightarrow \ GGC & \rightarrow \ GCG & \rightarrow \ CGT & \rightarrow \ GTG & \rightarrow \\
TGC & \rightarrow \ GCA & \rightarrow \ CAA & \rightarrow \ AAT
\end{align*}
\]

- This is the same sequence of reads that we found before!
- Thus, we will obtain the same sequenced genome as before.

Genome: \text{ATGGCGTGCA}
An *Eulerian* Cycle Recovers the Genome

We have an Eulerian cycle in this network:

\[
\begin{align*}
\text{ATG} & \rightarrow \text{TGG} \rightarrow \text{GGC} \rightarrow \text{GCG} \rightarrow \text{CGT} \rightarrow \text{GTG} \rightarrow \\
\text{TGC} & \rightarrow \text{GCA} \rightarrow \text{CAA} \rightarrow \text{AAT}
\end{align*}
\]

- This is the same sequence of reads that we found before!
- Thus we will obtain the same sequenced genome as before.
- The only difference: a computer can find an Eulerian cycle quickly.

Genome: \text{ATGGCGTGCA}
Dealing with Practical Complications
Complications in Genome Assembly

1. DNA is double-stranded (and may consist of multiple chromosomes).

2. Reads have imperfect “coverage” of the underlying genome.

3. Sequencing machines are error-prone.
Complication #1: Assembling Double-Stranded DNA

DNA strands run in opposite directions.

\[\ldots\text{ATGGCAATACGACAGTCAGCGGACAGACGTTAC}\ldots\]
\[\ldots\text{GTAACGTCTGTCCGCTGACTGTCGTATTGCCAT}\ldots\]
Complication #1: Assembling Double-Stranded DNA

DNA strands run in opposite directions.

\[
\ldots \text{ATG} \text{GCAATACGAC} \text{AGTCAGCGGACAGACGTTAC} \ldots
\]

\[
\ldots \text{GTAACGTCTGTCCGCTGACT} \text{GTGCTATTGC} \cdots \cdots \text{GTAACGTCTGTCCGCTGACT} \text{GTGCTATTGC}
\]

Given a read, we need to search for it and its reverse complement against the genome.

Read  \textbf{GTCGTATTGC}

Reverse Complement  \textbf{GCAATACGAC}
Complication #2: Imperfect Coverage

In practice, we can only reconstruct the genome in a number of pieces called **contigs**.
Complication #3: Errors in Reads Cause “Bubbles”
Thank You!

Questions?