Final Notes & Review: Genome Assembly

Part 1: The "Big Picture" - The Exploding Newspaper Problem

This is the central analogy for the entire lecture. Understanding it is key.

- **The Analogy:** Imagine you have 100 identical copies of today's newspaper. You blow them up with dynamite. All you have left is a massive pile of tiny, overlapping confetti snippets. Your task is to reconstruct the original newspaper. You can't just find the pieces from *one* copy; you must use the **overlaps** from all the different pieces to figure out the correct sequence of words.
- **The Biological Problem:** This is *exactly* the problem of **Genome Sequencing**. For technical reasons, we cannot read a long DNA strand (a chromosome) from start to finish. We can only read short, random fragments called **reads**. The process is:
    1. Take millions of identical copies of a cell's DNA.
    2. "Shatter" them into millions of short, random pieces.
    3. Sequence these short pieces to generate a massive list of reads.
    4. Use a computer to solve the "exploding newspaper" problem: computationally reassemble the original genome based on the overlaps between the reads. This reassembly process is called **Genome Assembly**.

Part 2: The Core Challenge & Initial Assumptions

The Number One Villain of Genome Assembly: Repeats

- **The Problem:** If a sequence (like ATG) appears multiple times in the genome, it creates ambiguity. When your assembly reaches ATG, you now have multiple possible k-mers that could come next. Which path do you follow? Taking the wrong path will lead to an incorrect assembly. The human genome is over 50% repetitive sequence, which makes this an enormous challenge.

Initial (Unrealistic) Assumptions for Learning:
To make the problem solvable at first, the lecture starts with a set of ideal conditions:

1. All reads are the same length k (we call them k-mers).
2. All reads come from the same strand (we ignore reverse complements for now).
3. There are no sequencing errors.
4. We have "perfect coverage" (every single k-mer from the genome is present in our list of reads).

This simplifies the problem to its core: **The String Reconstruction Problem**.

- **The Problem:** Given a collection of all k-mers from a genome, reconstruct the original genome string.

Part 3: The First Solution - The Overlap Graph (An Intuitive but Flawed Idea)

This is the first and most intuitive way to model the problem.

- **How it Works:**
    - Let every **read (k-mer)** be a **node** in a graph.
    - Draw a directed **edge** from k-mer A to k-mer B if they overlap. Specifically, if the suffix of A is the same as the prefix of B (e.g., AGTC overlaps with GTCA).
- **The Goal:** To reconstruct the genome, we need to find a path through this graph that visits **every single node (read) exactly once**.
- **The Computational Name:** This is the famous **Hamiltonian Path Problem**.

The Fatal Flaw: The Hamiltonian Path Problem is **NP-hard**. This means there is no known efficient algorithm to solve it. For a real genome with millions of reads (nodes), trying to find a Hamiltonian path is computationally impossible. Our intuitive first attempt has led us to a dead end.

Part 4: The Second Solution - The de Bruijn Graph (A Clever and Powerful Idea)

This is the breakthrough concept that makes modern genome assembly possible. It's less intuitive but computationally brilliant.

- **How it Works (The Flip):**
    - Let every **overlap** be a **node**. The nodes are all the unique (k-1)-mers (the prefixes/suffixes of our reads).
    - Let every **read (k-mer)** be an **edge**. A k-mer like ATGC becomes an edge connecting the node ATG (its prefix) to the node TGC (its suffix).
- **The Goal:** To reconstruct the genome, we now need to find a path that travels along **every single edge (read) exactly once**.
- **The Computational Name:** This is the famous **Eulerian Path Problem**.

The "Eureka!" Moment - Euler's Gift to Genomics:

- **Why this is a breakthrough:** Unlike the Hamiltonian Path Problem, the **Eulerian Path Problem is efficiently solvable!** The great mathematician Leonhard Euler solved the "Seven Bridges of Königsberg" problem in the 1700s, which laid the foundation for solving this exact problem.
- **Euler's Theorem:** A graph has an Eulerian cycle (a path that visits every edge once and returns to the start) if and only if it is **strongly connected** and every node is **balanced** (meaning its in-degree equals its out-degree). A path exists if at most two nodes are unbalanced.
- **The Bottom Line:** By changing our graph representation from reads-as-nodes to reads-as-edges, we transformed an **intractable problem (Hamiltonian)** into a **tractable, efficiently solvable one (Eulerian)**. This is the single most important takeaway of the entire lecture.

Part 5: Dealing with Reality - Complications & Modern Solutions

The simple Eulerian path model is great, but real data is messy. The end of the lecture addresses how we handle real-world problems.

1. **Issue: Imperfect Coverage.** We don't get every k-mer from the genome. This creates gaps in our de Bruijn graph.
    - **Solution: Read Breaking.** We can take our longer (but imperfect) reads and break them down into shorter, overlapping k-mers. For example, break a 100-mer into all of its constituent 30-mers. This increases the chances of having perfect coverage for the shorter k, but it also makes the graph more tangled with repeats. It's a trade-off.
    - **Result: Contigs.** Because of these gaps, we often can't find a single Eulerian path. The algorithm instead finds all the continuous, non-branching paths in the graph. These are called **contigs**. The output of a real assembly is often a set of contigs, not a complete chromosome.
2. **Issue: Sequencing Errors.** Reads often contain mistakes.
    - **Result: Bubbles.** An error in a read creates a small detour in the de Bruijn graph. You get two very similar paths that start and end at the same nodes, creating a "bubble."
    - **Solution: Bubble Popping.** Assembler software includes algorithms to detect these bubbles and "pop" them by removing the path that is supported by fewer reads, assuming it's the erroneous one.
3. **Issue: Repeats are still a problem.** Even with a de Bruijn graph, a long repeat can create a tangled mess that is hard to resolve.
    - **Solution: Read Pairs.** Modern sequencing can generate **read-pairs**. We get two short reads, read1 and read2, and we know that they are separated by a known distance d in the genome (e.g., d=500 bases). This is like having a "gapped" read. This long-range information is incredibly powerful for resolving repeats and finding the correct path through tangled regions of the graph. This leads to a more complex but more powerful **Paired de Bruijn Graph**.

Potential Exam Questions & How to Answer Them

- **Q: What is the primary computational challenge in genome assembly, and how does it manifest in an overlap graph?**
    - **A:** The primary challenge is the presence of **repeats** in the genome. In an overlap graph, a k-mer that represents a repeat will have multiple outgoing edges, creating ambiguity about which path to follow. Choosing the wrong path can lead to an incorrect assembly.
- **Q: Explain the fundamental difference between an Overlap Graph and a de Bruijn Graph.**
    - **A:** In an **Overlap Graph**, the reads (k-mers) are the **nodes**, and the overlaps are the **edges**. In a **de Bruijn Graph**, this is flipped: the overlaps ((k-1)-mers) are the **nodes**, and the reads (k-mers) are the **edges**.
- **Q: Why is the de Bruijn graph approach considered superior to the overlap graph approach for genome assembly?**
    - **A:** Because it transforms the problem. Reconstructing a genome in an overlap graph requires finding a **Hamiltonian Path** (visit every node once), which is computationally intractable (NP-hard). Reconstructing it in a de Bruijn graph requires finding an **Eulerian Path** (visit every edge once), which is computationally efficient and can be solved in linear time thanks to Euler's theorem.
- **Q: In a real sequencing project, you rarely get a single, complete chromosome. What do you get instead, and why?**
    - **A:** You get a set of **contigs**. This happens for two main reasons:
    - 1) **Imperfect Coverage** means some reads are missing, creating gaps and breaking the de Bruijn graph's Eulerian path.
    - 2) **Long, complex repeats** create tangled regions in the graph that the algorithm cannot uniquely resolve, forcing it to stop the path and start a new contig on the other side.
- **Q: What is a "bubble" in a de Bruijn graph, and what is the most common cause?**
    - **A:** A bubble is a structure where two short, similar paths share the same start and end nodes. The most common cause is a **sequencing error** in a read, which creates a short, erroneous detour away from the correct path.

Genome Assembly: Practice Solutions and Explanations

Question 1: Why is de Bruijn graphs preferred over overlap graphs in genome assembly problems?

This is the most important high-level concept of the chapter.

- **The Short Answer:** Because de Bruijn graphs transform the assembly problem from a computationally "impossible" one into a "possible" one.
- **The Detailed Explanation:**
    1. **Overlap Graph:** In this approach, each read (k-mer) is a **node**. An edge connects two nodes if they overlap. To reconstruct the genome, you must find a path that visits **every node exactly once**. This is the famous **Hamiltonian Path Problem**, which is NP-hard. This means there is no known efficient algorithm to solve it for large graphs, making it computationally intractable for millions of reads.
    2. **De Bruijn Graph:** This approach cleverly flips the representation. Each overlap ((k-1)-mer) is a **node**, and each read (k-mer) is an **edge**. To reconstruct the genome, you must find a path that uses **every edge exactly once**. This is the **Eulerian Path Problem**.
    3. **The Breakthrough:** Thanks to Leonhard Euler's work from the 1700s, we have a very efficient (linear-time) algorithm to solve the Eulerian Path Problem, even for graphs with millions of edges.
- **Conclusion:** De Bruijn graphs are preferred because they represent the same biological problem in a way that is computationally easy to solve, whereas the more intuitive overlap graph leads to a problem that is currently impossible to solve for real-world genomes.

Question 2: Construct Overlap graph and de-bruijn graphs for the following DNA string: CGATCGATGTGTCGAT (Assume k=3). Also try to draw the de-bruijn graph for paired reads considering (k=3, d=1) reads.

Part A: Standard de Bruijn Graph (k=3)

1. **Enumerate all 3-mers:**
   CGA, GAT, ATC, TCG, CGA, GAT, ATG, TGT, GTG, TGT, GTC, TCG, CGA, GAT
2. **Construct Overlap Graph:**
    - **Nodes (Unique 3-mers):** CGA, GAT, ATC, TCG, ATG, TGT, GTG, GTC
    - **Edges (Connect suffix -> prefix):** This graph is complex. For example, CGA has the suffix GA. The only k-mer with prefix GA is GAT, so you have an edge CGA -> GAT. TCG has the suffix CG. The k-mer CGA has prefix CG, so you have an edge TCG -> CGA. This creates a cycle. The repeat of TGT and GAT will create multiple branching paths, making the Hamiltonian Path hard to find.
3. **Construct de Bruijn Graph:**
    - **Nodes (Unique 2-mers):** CG, GA, AT, TC, TG, GT, GT
    - **Edges (from the 3-mer list):**
        - CGA: Edge from CG to GA

        - GAT: Edge from GA to AT
        - ATC: Edge from AT to TC
        - TCG: Edge from TC to CG
        - ...and so on for all 14 k-mers.
    - The resulting graph is much simpler to analyze for an Eulerian path.

Part B: Paired de Bruijn Graph (k=3, d=1)

1. **Enumerate all (3,1)-paired reads:** A paired read is (k-mer_i | k-mer_{i+1+d}). Here, d=1, so we look at (k-mer_i | k-mer_{i+2}).
    - (CGA | ATC), (GAT | TCG), (ATC | CGA), (TCG | GAT), (CGA | ATG), etc.
2. **Construct Paired de Bruijn Graph:**
    - **Nodes (Paired (k-1)-mers):** The prefix of (CGA | ATC) is (CG | AT). The suffix is (GA | TC). These are the nodes.
    - **Edges (Paired k-mers):** The paired read (CGA | ATC) becomes a directed edge from node (CG | AT) to node (GA | TC).
    - **Benefit:** This graph adds more constraints. It tells you not only that CGA is followed by GAT, but that it is also linked to ATC two positions down the line, which is extremely helpful for resolving repeats.

Question 3: Assume you have the following reads (k=10): ...

This question tests the concept of **read breaking**.

1. **Create an overlap graph (i) and a de-bruijn graph (ii) considering shorter k-mers (k=3) from these three reads. Construct the superstring.**
    - **Step 1: Break the reads.** First, you must generate all 3-mers from the three 10-mer reads.
        - From GTGTGGGGCA: GTG, TGT, GTG, TGG, TGG, GGG, GGC, GCA
        - From GGGCAGCGAG: GGG, GGC, GCA, CAG, AGC, GCG, CGA, GAG
        - From GCGAGTTTT: GCG, CGA, GAG, AGT, GTT, TTT, TTT
    - **Step 2: Build the de Bruijn Graph.**
        - **Nodes (unique 2-mers):** GT, TG, GG, GC, CA, AG, GC, CG, GA, TT.
        - **Edges (all 3-mers listed above):** GTG is an edge GT -> TG, TGT is TG -> GT, etc.
    - **Step 3: Find the Eulerian Path and construct the superstring.** By tracing a path that uses every edge once (e.g., starting at GT, the only node with more outs than ins), you can reconstruct the original sequence. The path will spell out: **GTGTGGGCAGCGAGTTTT**.
    - The overlap graph would be too complex and is not the practical way to solve this.

Question 4: What happens to the de-bruijn graph if read length increases? Why we can not increase read length?

- **Effect of Increase:** As read length k increases, the de Bruijn graph becomes **simpler and less tangled**.
- **Explanation:** Repeats are what cause ambiguity (nodes with multiple incoming/outgoing edges). If k is longer than the longest repeat sequence in the genome, no (k-1)-mer node will be part of a repeat, and the graph will resolve into a single, simple path.
- **Why we can't just increase it:** There is a **technological trade-off**. Current DNA sequencing technologies cannot produce very long reads (e.g., >30,000 bases) that are also highly accurate. We can get short, accurate reads, or long, error-prone reads. There is a constant push to improve this, but for now, we are limited by technology.

Question 5: Does de-bruijn graph guarantee unique euler paths? How does that effect genome assembly? What are the solutions to this problem?

- **Guarantee Unique Paths?:** Absolutely not.
- **Effect on Assembly:** If there are **repeats** in the genome that are longer than k, the de Bruijn graph will have nodes with multiple incoming and outgoing edges. This creates branch points in the graph, leading to **multiple possible Eulerian paths**. Each path corresponds to a different potential genome assembly, and the algorithm doesn't know which one is correct. This is the primary reason assembly is hard.
- **Solutions:**
    1. **Increase k:** If possible, use a k that is longer than the repeat.
    2. **Use Read Pairs:** This is the most powerful solution. Read pairs provide long-range information that acts as a "scaffold," telling the algorithm which path to take through a repetitive, tangled region of the graph.

Question 6: How does creating shorter reads from original reads help in genome assembly?

This is the concept of **Read Breaking**.

- **How it Helps:** It solves the problem of **imperfect coverage**. We may not have sequenced every single 100-mer in a genome. But if we break all the 100-mer reads we *do* have into their constituent 30-mers, we have a much, much higher chance of having a complete set (perfect coverage) of all 30-mers.
- **The Trade-off:** The downside is that using a smaller k makes the de Bruijn graph more tangled and susceptible to being confused by shorter repeats.

Question 7: Explain contigs and bubbles and reason behind them.

- **Contigs: What they are:** Long, unambiguous, contiguous segments of an assembled genome. **Reason:** They are the result of **gaps in coverage**. When a read is missing from our data, it creates a break in the de Bruijn graph where an edge should be. The algorithm can't cross this gap, so it stops and reports the path it has built so far as a contig. A typical assembly results in hundreds of contigs instead of one complete chromosome.
- **Bubbles: What they are:** A structure in the de Bruijn graph where two short, similar paths diverge from a start node and converge back at an end node. **Reason:** They are most often caused by **sequencing errors** or slight variations in repeats (polymorphisms). An error in a single read creates an alternative, erroneous path that briefly detours from the correct path. Assemblers "pop" these bubbles by choosing the path that is supported by a higher number of reads.

Question 8: For each of the following graphs, find out if they have a euler path/cycle and hamiltonian path/cycle and if yes, find that.

Let's analyze them from left to right, top to bottom.

- **Graph 1 (House with X):**
  - **Eulerian:** Node degrees are: 4, 2, 4, 2, 3, 3. There are two nodes with odd degrees. **An Eulerian path exists.** It must start at one odd-degree node and end at the other.
  - **Hamiltonian:** By inspection, **a Hamiltonian cycle exists** (e.g., top -> middle-left -> bottom-left -> middle-right -> bottom-right -> top).
- **Graph 2 (House with extra arms):**
  - **Eulerian:** Node degrees are: 3, 3, 2, 2, 3, 1, 1, 1. There are six nodes with odd degrees. **No Eulerian path or cycle.**
  - **Hamiltonian:** By inspection, **no Hamiltonian path or cycle** can exist because of the nodes with degree 1. If you start on an "arm," you must immediately go into the main body, and you can never return to the arm.
- **Graph 3 (3x3 grid):**
  - **Eulerian:** All corner nodes have degree 2, edge nodes have degree 3, center node has degree 4. There are four nodes with odd degrees. **No Eulerian path or cycle.**
  - **Hamiltonian:** By inspection, **a Hamiltonian path exists** (it can be traced like a snake), but **no Hamiltonian cycle** exists. Once you visit the center node, you cannot easily get back to the start without revisiting a node.
- **Graph 4 (Diamond shape):**
  - **Eulerian:** Node degrees are: 4, 4, 6, 4, 4, 3, 3. There are two nodes with odd degrees (top and bottom points). **An Eulerian path exists.**
  - **Hamiltonian:** By inspection, **a Hamiltonian cycle exists**.
- **Graph 5 (Square with two diagonals):**
  - **Eulerian:** Node degrees are: 3, 3, 3, 3, 3, 3, 3, 3. All eight nodes have an odd degree. **No Eulerian path or cycle.**
  - **Hamiltonian:** By inspection, **a Hamiltonian cycle exists** (e.g., trace the outer perimeter, then cut across the middle).

Here are the one-liner key points for your last-minute exam review, covering the entire Genome Assembly topic.

Final Review: Genome Assembly - The One-Liner Key Points

- **The Core Problem:** Genome Assembly is the puzzle of reconstructing a full genome from millions of short, overlapping DNA fragments ("reads").
- **The Main Villain: Repeats** are the biggest challenge in assembly, as they create ambiguity and multiple possible paths in any graph model.
- **The Intuitive-but-Flawed Approach: Overlap Graphs** (where reads are nodes) fail because they frame the problem as finding a **Hamiltonian Path** (visit every node once), which is computationally impossible (NP-hard) for large genomes.
- **The Genius Solution: De Bruijn Graphs** (where overlaps are nodes and reads are edges) succeed by transforming the problem into finding an **Eulerian Path** (visit every edge once), which is efficiently solvable.
- **The Key Theorem: Euler's Theorem** provides the "rules of the road," stating that an Eulerian Path exists if a graph is connected and has at most two nodes with an unequal number of incoming and outgoing edges.
- **Problem 1: Missing Reads (Imperfect Coverage):** Gaps in read data break the de Bruijn graph's path, forcing assemblers to output partial sequences called **Contigs**.
- **Problem 2: Sequencing Errors:** Mistakes in reads create short, divergent paths in the graph called **Bubbles**, which algorithms must detect and "pop."
- **Solution for Gaps (Trade-off): Read Breaking** (using shorter k-mers from longer reads) improves coverage to help form contigs but can make the graph more tangled with short repeats.
- **Solution for Repeats & Gaps: Read Pairs** are the most powerful tool, providing long-range information that acts as a scaffold to jump over repeats and correctly order the final contigs.

–Fahad Nadim Ziad, 24341216