P&S Mobile Genomics
Lecture 9: Genome Assembly

Can Firtina
ETH Zürich
Spring 2022
10 May 2022
Agenda for Today

- Genome Assembly
  - Basics
  - Overlap-Layout-Consensus
Recall: Caveats of Sequencing Technologies

Small pieces of a puzzle
short reads (Illumina)

Large pieces of a puzzle
long reads (ONT & PacBio)

Which sequencing technology is the best?

- 100-300 bp
  - low error rate (~0.1%)
- 500-2M bp
  - high error rate (~15%)

Looking forward,
Will we be able to read
the entire genome sequence?
Genome Assembly Basics

- There is no sequencing technology that can read an entire chromosome from start to end
  - Rather we have short fragments of genome: **Reads**

- Reconstruct the actual genome from its pieces to
  - Compare two genomes to reveal large structural variations as well as small mutations to **pinpoint diseases** and **study certain phenotypes** (e.g., eye color, hair color)
  - Map known genes
  - Use it as a reference to map reads from the same species
  - ...

- Two major approaches to reconstruct a genome
  - Hierarchical sequencing
    - Human Genome Project
    - **Slow, expensive**, but highly accurate and contiguous assembly
  - Whole genome shotgun (WGS) sequencing
    - **Fast, cheaper**, but less accurate and less contiguous
Genome Assembly from WGS Sequencing

Genome (Non-human-readable)

Reads (Human-readable)

Overlap

Find the ordering (i.e., Layout)

Consensus (i.e., assembly)
A Common Assembly Pipeline

Input: Reads → Overlapping Reads → Layout → Consensus → Analysis

Output: Consensus → Analysis

Optional: Error Correction and Scaffolding (Ordered Contigs)
Overlapping Reads

- **Goal:** Solve the genome assembly puzzle by filling the gaps with *overlapping reads*
- **Overlaps:** Matching blocks between *pairs of reads* using
  - Exact matching short subsequences between reads
  - Suffix Tree
  - Alignment
- **Condition:** Suffix of a read overlaps prefix of another read

---

<table>
<thead>
<tr>
<th>Overlapping Reads:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTGAAGCAGCTATGACTA</td>
</tr>
<tr>
<td>AAGCAGCTATGACTATTACT</td>
</tr>
<tr>
<td>GCACGTGGACTATTACTAA</td>
</tr>
<tr>
<td>TACCGATTGGACTATTACTTA</td>
</tr>
<tr>
<td>GGACTATCCCATTCACCTGGAT</td>
</tr>
<tr>
<td>CATTTACACCTGGATGACTAC</td>
</tr>
<tr>
<td>ACGGATACCATCCTACTTACT</td>
</tr>
<tr>
<td>GGATCTTTACTTTACTGACTAC</td>
</tr>
<tr>
<td>AGCGTTCACGTCCTAGGC</td>
</tr>
<tr>
<td>GGTACCCTGAGCCTAGAAACT</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Overlapping Reads:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTGAAGCAGCTATGACTA</td>
</tr>
<tr>
<td>AAGCAGCTATGACTATTACT</td>
</tr>
<tr>
<td>GCACGTGGACTATTACTAA</td>
</tr>
<tr>
<td>TACCGATTGGACTATTACTTA</td>
</tr>
<tr>
<td>GGACTATCCCATTCACCTGGAT</td>
</tr>
<tr>
<td>CATTTACACCTGGATGACTAC</td>
</tr>
<tr>
<td>ACGGATACCATCCTACTTACT</td>
</tr>
<tr>
<td>GGATCTTTACTTTACTGACTAC</td>
</tr>
<tr>
<td>AGCGTTCACGTCCTAGGC</td>
</tr>
<tr>
<td>GGTACCCTGAGCCTAGAAACT</td>
</tr>
</tbody>
</table>
Storing Overlaps in Graphs

- **Graphs** are useful to 1) avoid storing redundant reads and 2) identify ordering of overlaps
- **Nodes**: Reads/Chunks of reads
- **Directed Edges**: When suffix of one read overlaps prefix of another read
  - **Label**: Number of matches between overlapping reads

```
ATTGAAGCACGTATACTA  ATTTGAAGCACGTATACTA  AAGCAGTATACCTATTACT
 ||||||||||||||   |||||           |||||           ||||| 14
 AAGCACGTATATATTACT  GCACGTGACTACCT  GCACGTGACTACCT
 10                          10

ATTGAAGCAGCTATACCTA  AAGCAGTATACCTATTACT
 14
 GCACGTGACTACCTATTACTA
 15
 GCACGTGACTACCTATTACT
```

- Edges can get quite messy
Let’s construct the following string from its pieces:
- `to_every_thing_turn_turn_turn_turn_there_is_a_season`

**Pieces**: Every substrings of length 7 (7-mers)

A part of such an overlap graph:

---

**Goal**: Find assembly by **ordering overlaps** correctly

How to find a **simpler** ordering of overlaps relative to each other from the overlap graph?

Image source: [http://www.cs.jhu.edu/~langmea/resources/lecture_notes/16_assembly_scs_v2.pdf](http://www.cs.jhu.edu/~langmea/resources/lecture_notes/16_assembly_scs_v2.pdf)
A Common Assembly Pipeline

Input:
- Reads
- Overlapping Reads
- Layout

Output:
- Consensus

Optional:
- Error Correction and Scaffolding (Ordered Contigs)
- Analysis
Layout – Graph Cleaning

- Overlap graphs may contain **redundant information**
  - **Transitive (redundant) edges**: An edge from node $v$ to node $w$ ($v \rightarrow w$) is transitive if:
    - There exists $v \rightarrow u$ and $u \rightarrow w$
    - We can remove the edge $v \rightarrow w$ without losing the ability to visit $w$ starting from $v$

- **Bubbles**: A directed acyclic graph with sink and source nodes $v$ and $w$ such that
  - There exist at least two *isolated* paths from $v$ to $w$
  - We want to collapse bubbles to simplify the overlap graph

- **Tips**: Short branches in the graph that terminate very early
Overlap graphs may contain **redundant edges**

- **Transitive edges** can be removed without losing the connectivity information of the graph
- The green edges are **transitive edges** because blue edges provide the connectivity information that green edges provide.

Image source: [http://www.cs.jhu.edu/~langmea/resources/lecture_notes/16_assembly_scs_v2.pdf](http://www.cs.jhu.edu/~langmea/resources/lecture_notes/16_assembly_scs_v2.pdf)
Layout – Transitive Reduction Example

- Overlap graphs may contain **redundant edges**
  - Transitive edges that can be removed without losing the connectivity information of the graph
  - Let’s remove the transitive edges that skip one or two nodes:

    ![Diagram of transitive reduction example](http://www.cs.jhu.edu/~langmea/resources/lecture_notes/16_assembly_scs_v2.pdf)

  - Remember the messy overlap graph?
Overlap graphs may contain **redundant edges**

- Transitive edges that can be removed without losing the connectivity information of the graph
- Let’s remove the transitive edges that skip one or two nodes:

After the transitive reduction:

- It is now much easier to identify **ordering of overlaps** from this graph

Image source: http://www.cs.jhu.edu/~langmea/resources/lecture_notes/16_assembly_scs_v2.pdf
Bubbles: Different multiple paths with the same source and sink
- May remain undetected after transitive edge removal
- One of the paths are collapsed (e.g., the shorter one)
  - Shorter paths may be due to **repeats** after transitive reduction

We can collapse bubbles to
- Reduce the complexity of the overlap graph
- Improve the contiguity of the assembly inferred from the graph

Why do we have bubbles?
- Sequencing errors (missing overlaps)
- Variants between parent genomes (diploid and polyploid genomes)
Read the following paper if you are curious about

- How the transitive reduction works:

  **BIOINFORMATICS**
  Vol. 21 Suppl. 2 2005, pages i79–i85
doi:10.1093/bioinformatics/bti1114

  *Genes and Genomes*

  **The fragment assembly string graph**
  Eugene W. Myers
  Department of Computer Science, University of California, Berkeley, CA, USA

- How to collapse bubbles in overlap graphs:

  **Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences**
  Heng Li
  Author Notes

  *Bioinformatics*, Volume 32, Issue 14, 15 July 2016, Pages 2103–2110,
  [https://doi.org/10.1093/bioinformatics/btw152](https://doi.org/10.1093/bioinformatics/btw152)

  **Published:** 19 March 2016  **Article history**
Overlap graphs may contain **redundant edges**

- Transitive edges that can be removed without losing the connectivity information of the graph
- After the transitive reduction:

- **Bubble Collapsing**

Image source: [http://www.cs.jhu.edu/~langmea/resources/lecture_notes/16_assembly_scs_v2.pdf](http://www.cs.jhu.edu/~langmea/resources/lecture_notes/16_assembly_scs_v2.pdf)
Spelling out the Contigs

- Take all nodes with unambiguous branches (e.g., single branch, leading no cycles)
- “Spell out” the contig by following the unambiguous branches
A Common Assembly Pipeline

Input:
- Reads
- Overlapping Reads
- Layout

Output:
- Consensus

Optional:
- Error Correction and Scaffolding (Ordered Contigs)
- Analysis
Consensus of Overlapping Reads

- Layout the overlaps of reads from the overlap graph
- Take the consensus at each base to generate contigs

Contig:

```
ATTGACCTAAACTTTACCT
  | | | | | | | | | | | |
TGACCTAATTTTACCT
  | | | | | | | | | | | |
CCTAATTTTAGCTTTAGC
  | | | | | | | | | | | |
TTTTACCTTTTAGATTGAGGACGACG
  | | | | | | | | | | | |
TACCTTTTAGATTGAGGACGACG
  | | | | | | | | | | | |
```

```
ATTGACCTAAATTTTACCTTTAGATTGAGGACGACG
  | | | | | | | | | | | |
TAGTTTTGAGGACGACGACGACGACGCACAGGAC
  | | | | | | | | | | | |
ATTGACCTAAATTTTACCTTTTAGATTGAGGACGACGACGCACAGGAC
  | | | | | | | | | | | |
```
A Common Assembly Pipeline

Input:
- Reads
- Overlapping Reads
- Layout

Output:
- Consensus

Optional:
- Error Correction and Scaffolding (Ordered Contigs)
- Analysis
Consensus of Overlapping Reads

- Take the consensus at each base to generate contigs

Contig: ATTGACCTAAACCTTTACCT

Sequencing Errors?
Assembly Polishing (Error Correction)

- Sequencing errors on reads may propagate to contigs
  - Leading to inaccurate analysis on the assembly we just generated
- Idea: Align reads back to contigs again to generate a stronger consensus

Errenous Contig:

Aligned Reads:

Corrected Contig:

Sequencing Errors

Align Reads to Contigs

Correct Errors
A Reading on Assembly Polishing


Apollo: a sequencing-technology-independent, scalable and accurate assembly polishing algorithm

Can Firtina, Jeremie S Kim, Mohammed Alser, Damla Senol Cali, A Ercument Cicek, Can Alkan ☑️, Onur Mutlu ☑️


Published: 13 March 2020   Article history ▼
Contigs are usually **not ordered**

A *gapless* chromosome may potentially be represented by several *gapped contigs*

- What is the relative order of contigs to represent the genome correctly?

**Unordered Contigs:**

---

[Diagram of unordered contigs]
Scaffolding – Ordering the Contigs (cont’d)

- Overlap parts of reads to contigs to find the **pairwise ordering of contigs**

![Diagram showing the process of scaffolding]

- Ultra long reads, paired-end reads, optical mapping usually help scaffolding
  - These are good keywords to check if you are curious
A Common Assembly Pipeline

Input:

1. Reads
2. Overlapping Reads
3. Layout

Output:

Consensus (Gapped Contigs)

Optional:

Error Correction and Scaffolding (Ordered Contigs) → Analysis
What Makes a Good Assembly?

- **Accurate**
  - Should be resolved from errors as much as possible
  - Solutions:
    - Long and accurate reads (e.g., PacBio HiFi reads)
    - Error correction tools
    - Accurate assemblers

- **Contiguous**
  - **Gaps**: Missing information on assembly
  - Solutions:
    - Long and accurate reads
    - Accurate assemblers
    - We need better tools to resolve repeats in overlap graphs

- Tools to generate overlaps: Minimap2, Canu
- Tools for assembly: Miniasm, mdbg, Canu, Hifiasm, Flye
- Tools to assess the assembly quality: QUAST and the MUMmer package
P&S Mobile Genomics
Lecture 9: Genome Assembly

Can Firtina
ETH Zürich
Spring 2022
10 May 2022