De novo genome assembly

Andrew Tritt / Alicia Clum
CGRL Workshop
October 22, 2014
JGI Programs & Infrastructure

DOE Mission Areas

Bioenergy  Carbon Cycling  Biogeochemistry
Why Assemble?

### Metagenomes (600/yr)
Assembly supports understanding community structure and metabolic capabilities.

### Fungi (200/yr)
Reference genomes for understanding taxonomic relationships, and identification of genes involved in carbon cycling.

### Microbes (>1000/yr)
Reference genomes for clarifying and expanding taxonomic understanding of microbial life.
I. Background
  • Graphs

II. Genome assembly
  • Old → Assembly in the Sanger days
    • Overlap-layout-consensus
  • New → Assembly in the era of big data
    • De Bruijn graphs
  • Scaffolding → utilizing paired-end data

III. Hands On
  • Assembly of short-read data
Graphs

Not like that of a function or a chart

• a representation of a set of objects where pairs of objects are connected by links

• consists of a set of nodes (aka vertices) that represent the objects and a set of edges that represent the links or relationships between objects

• used to represent many real-life problems and systems
Examples:

- Biology: gene regulatory networks, neuron networks, species migration pattern
- Chemistry/Physics: three dimensional molecules
- Computer science: websites, data structures, electrical circuits and many many more
- Sociology: social networks

Image from: //socialnetworking.lovetoknow.com/What_is_Social_Network_Theory
The visual representation:

ABCDEF are “nodes”

A and B have some relationship, but A and C do not.
Example: Airports and flights

What is the easiest way to get to New York from San Francisco?
In addition to connecting nodes with edges, we can *weight* edges and *direct* them.
What is the easiest way to get to New York from San Francisco, *round-trip*?
What is the *quickest* way to get to New York from San Francisco?
So, graphs are great if you want to plan a trip, but they are also great if you want to assemble a genome!
Graphs and genome assembly

- **Nodes** represent some part of a sequence
  - In Overlap-Layout-Consensus (OLC), represent “reads”
  - In de Bruijn graphs, represent “k-1 mers”

- **Edges** represent some overlap between two sequences
  - In OLC, represents overlap between “reads”
  - In de Bruijn graphs, represent “k mers”
Assembly done by

1. Overlapping
   - Align each read to all other reads

2. Layout
   - Construct a graph to get an approximate read layout

3. Consensus
   - Traverse the graph to compute a consensus sequence

* Sanger data assembled using this algorithm, new long read technology is making these popular again
Example:

Genome: ATGCTAGTGCGACAGCTTTGATCGATCGAT

Reads:

ATGCTAG
TAGTGGCAGA
AGAGCTT
CTTGATCGAT
CGATCGAT
Visit each node exactly once, and require the maximum overlap between nodes
Overlap-Layout-Consensus

Genome: \texttt{ATGCT\underline{TAGTGGCAGAGCTTGTGATCGATCGAT}}

What happens if we don’t require the maximum overlap between nodes?

Mis-assembly! \texttt{ATGCT\underline{TAGA}G} instead of \texttt{ATGCT\underline{TAGT}}
Problems with OLC algorithm

1. Computationally expensive
   - Aligning every read to each other read is expensive
     - \( O(n^2) \): for \( n \) reads, \((n^2 - n)/2\) alignments

2. No straightforward solution
   - Visiting each node exactly once is called a Hamiltonian path
     - NP-complete: A solution may or may not exist, but there is no way of knowing, and finding the solution is not deterministic
For a given sequence of characters, a de Bruijn graph represents all k-mers in the sequence

- Nodes represent \((k-1)\)-mers
- Edges represent overlap between two \(k\)-mers
  - An edge exists between two nodes if the \(k\)-mer representing the edge is found in the data set
    - \((k-2)\)-length suffix of one node is the \((k-2)\)-length prefix of the other node
  - Weighted according to \(k\)-mer frequency
  - Directed from suffix node to prefix node
Building a De Bruijn graph with $k=3$

**Genome:** ATGGCGTGCA

**k-mers:** \{ATG, TGG, GGC, GCG, CGT, GTG, TGC, GCA\}

Once graph is built, traverse it to extract contigs

Repetitive sequence. What do we do here?

Contig gets broken 😞
De Bruijn Graphs

Advantages

1. Computationally tractable
   • want every $k$-mer in graph to be in assembly $\rightarrow$ traverse every edge exactly once $\rightarrow$ Eulerian path (solvable problem)

2. Less expensive computation
   • $k$-mers are hashed, no aligning reads.
     • Hash = efficient storage and comparison of strings
     • $O(n)$
       • for every read of length $L$, $L - k + 1$ $k$-mers to hash.
       • memory requirements scale linearly with genome size
Thus far, we have talked about how to make *contigs*.

→ utilizes individual reads

Generally, we sequence a DNA molecule from both ends

→ two *paired* reads

• use this pairing information to connect contigs → *scaffolding*
1. Build contigs

2. Map (align) reads to contigs

3. Connect with pairing information
Okay, so now we know which contigs are linked… what next?

Connect these contigs with a sequence of N’s

Estimate the number of N’s using pairs that map to the same contigs
Scaffolding

An example assembly:

> Scaffold1

```
GGGATAAAATATGTTGCAAGTTTTTCTAGGATACGTAATTTTCTTCATT
TACAAAAAGTATAGTAAAGGTATAGCAATAGGAGGAAGAAATAGAAG
CAAAAGAGAATCAAACCACTCCCTATAATATCTTTTTGATTTTTCTTTGAAT
ATCTCGCTTTTTTTGAGAATTTCTTTACCACAAAAACACTGCTC
```

NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN

```
ATTGTAACGTGAGGTCTTCTTTTTGACAAAGATCATTTTTGACTACTACA
GGAATGCTCCTTAAAGCTTTTAAGCCTTATGACATTTTATTAGACAATAA
GTATTGGTAGTATTATGTCCCAACATTAGGATTATAGCGGAACAGCAGCAG
```

...
Our biggest problem

**REPEATS!**

Repeats present ambiguity in genome sequence

→ E.g. a read is generated from a duplicated region

→ Which specific region did it come from?

“Without repeats, it would just be simple math”

Figure is a *dot-plot* of inaZ gene. Provided by Dr. Wolber

dots represent matching sequence
A fungal de Bruijn graph
Questions?
Sequencing capacity

How do we assemble this data back into a genome?
Platform Summary

- Massively parallel short reads
- limited by phasing errors
- read length up to 300bp
- 0.3% error, mostly substitutions and at ends
- Gigabases / run collected in days or weeks

CGRL

- Single molecule long reads
- read length limited by polymerase damage
- reads lengths up to 15 kbp (or longer)
- 15% error, mostly randomly distributed indels
- Megabases / run collected in hours