CS 68: BIOINFORMATICS

Prof. Sara Mathieson
Swarthmore College
Spring 2018
Outline: Jan 29

- Continue: de Bruijn graph (DBG) assembly theory
- DBG assembly in practice
- Evaluation of assemblies

Notes:
- Lab 1 due Wednesday
- Assembly reading posted: spend 1.5 hours max
- Office hours today 3-5pm
- Fill out partner form for Lab 2 (if you know who you want to work with)

Candidate job talks:
Monday 11:30-12:20
Wednesday 11:30-12:20
Recap:
building the de Bruijn graph (DBG)
Take each length-3 input string and split it into two overlapping substrings of length 2. Call these the *left* and *right* 2-mers.

```
AAABBBA
```

take all 3-mers:  
AAA, AAB, ABB, BBB, BBA

form L/R 2-mers:  
AA, AA, AA, AB, AB, BB, BB, BB, BB, BB, BA

L  R  L  R  L  R  L  R  L  R

Let 2-mers be nodes in a new graph. Draw a directed edge from each left 2-mer to corresponding right 2-mer:

Each edge in this graph corresponds to a length-3 input string
DBG:
- Nodes: (k-1)-mers
- Edges: k-mers of the genome or reads
DBGs can have multi-edges, making them multi-graphs

If we add one more B to our input string: AAABBBBA, and rebuild the De Bruijn graph accordingly, we get a multiedge.
Graph terminology

Directed **multigraph** $G(V, E)$ consists of set of vertices, $V$ and **multiset** of directed edges, $E$

Otherwise, like a directed graph

Node’s **indegree** = # incoming edges

Node’s **outdegree** = # outgoing edges

De Bruijn graph is a directed multigraph

$V = \{ a, b, c, d \}$

$E = \{ (a, b), (a, b), (a, b), (a, c), (c, b) \}$
Graph terminology (cont.)

Node is **balanced** if indegree equals outdegree

Node is **semi-balanced** if indegree differs from outdegree by 1

Graph is **connected** if each node can be reached by some other node

**Eulerian walk** visits each edge exactly once

Not all graphs have Eulerian walks. Graphs that do are **Eulerian**.

A directed, connected graph is Eulerian if and only if it has at most 2 semi-balanced nodes and all other nodes are balanced

Jones and Pevzner section 8.8
Graph terminology (cont.)

- Node is **balanced** if indegree equals outdegree.
- Node is **semi-balanced** if indegree differs from outdegree by 1.
- Graph is **connected** if each node can be reached by some other node.
- **Eulerian walk** visits each edge exactly once.
- Not all graphs have Eulerian walks. Graphs that do are **Eulerian**.

A directed, connected graph is Eulerian if and only if it has at most 2 semi-balanced nodes and all other nodes are balanced.

Jones and Pevzner section 8.8

Slide: adapted from Ben Langmead, John Hopkins
Back to our De Bruijn graph

Is it Eulerian?  Yes

Argument 1: $AA \rightarrow AA \rightarrow AB \rightarrow BB \rightarrow BB \rightarrow BA$

Argument 2: $AA$ and $BA$ are semi-balanced, $AB$ and $BB$ are balanced
How to get the sequence from an Eulerian path?

Back to our De Bruijn graph

Is it Eulerian? Yes

Argument 1: **AA → AA → AB → BB → BB → BA**

Argument 2: **AA** and **BA** are semi-balanced, **AB** and **BB** are balanced
How to get the sequence from an Eulerian path?

Back to our De Bruijn graph

Is it Eulerian? Yes

Argument 1: $AA \rightarrow AA \rightarrow AB \rightarrow BB \rightarrow BB \rightarrow BA$

Argument 2: $AA$ and $BA$ are semi-balanced, $AB$ and $BB$ are balanced

Start with the sequence in the first node. Follow the path, adding on one base each time.

Slide: adapted from Ben Langmead, John Hopkins
Handout 1: work with a partner

Goals:
1) Practice the mechanics of constructing a de Bruijn graph
2) See issues that affect both OLC and DBG assembly
3) Think about how we would ask a computer to find an Eulerian path
Creating Eulerian paths
1. $4^k$
2. ACGTAG
3. $2^k$
   - ZAB, ABC, BCD,
   - CDA, DAB, ABE,
   - ECA, FAB, ABY

$G = 12$
$k = 3$

$O(G)$

$6 - k + 1$

10 k-mers
Fleury’s Algorithm $O(|E|^2)$

- start at node with more outgoing edges
- while there are still edges:
  - traverse & delete edge that does not disconnect graph (if possible)

$E = \{ (a, b), (a, b), \ldots \}$
Recursive Algorithm

- start at any node u
- cycle # type stack
  - find-cycle(u):
    - for each edge e=(u,v):
      - remove e
      - find-cycle(v)
      - push u onto cycle