CS4330: Combinatorial Methods in Bioinformatics

Practical genome assembly based on de Bruijn graphs

Wong Limsoon

Acknowledgement: This set of slides were adapted from Ken Sung’s
Practical issues in genome assembly based on de Bruijn graph

- Read errors
- Heterozygosity
- Repeats
- Incomplete coverage

De Bruijn graph becomes big, complicated, & contain many erroneous edges
De Bruijn graph fragments into many connected components

Choice of K for constructing the de Bruijn graph
How successful practical genome assemblers deal with these issues

Velvet

Efficient in assembling short-read sequencing data


SOAPdenovo

Suitable for large-scale genome assembly


SPAdes (aka St. Petersburg genome assembler)

Can handle diverse sequencing data types, including short reads, long reads, and mate-pair reads

Velvet

https://en.wikipedia.org/wiki/Velvet_assembler

Simplification

Remove tips

Merge bubbles

Remove low-coverage edges
Step 1: Simplify the graph $DB_K(\mathcal{R})$

Whenever node $x$ has only one outgoing edge, and it goes to a node $y$ which has only one incoming edge, these two nodes are merged

$Akin\ to\ x \oplus_K y$

Do this until no further merging is possible
Exercise

Is this step “safe”?
If not, how to make it safer?

Step 1: Simplify the graph $DB_K(\mathcal{R})$

Whenever node $x$ has only one outgoing edge, and it goes to a node $y$ which has only one incoming edge, these two nodes are merged

*Akin to $x \in^\mathcal{R} y$

Do this until no further merging is possible
Step 2: Remove “tips”

“Tips” correspond to edges due to erroneous reads

Do tip removal until no further removal is possible

A “tip” is a chain of nodes such that

*Length is at most 2K*
*Disconnected at one end*
*Each node has low # of occurrences in the reads*
Exercise

Why is the length of a tip defined to be at most 2K?
What might be other good choices?

Step 2: Remove “tips”

“Tips” correspond to edges due to erroneous reads
Do tip removal until no further removal is possible

A “tip” is a chain of nodes such that
Length is at most 2K
Disconnected at one end
Each node has low # of occurrences in the reads
Step 3: Merge “bubbles”

“Bubbles” correspond to errors or SNPs

Use “Tour bus” algo to merge bubbles

“Bubbles” are two paths having the same starting node and the same ending node where these two paths represent two strings differing by very few nucleotides (e.g., 1 nucleotide)
Merge a bubble

- The top path represents GACTCCGAG.
- The bottom path represents GACTTCTGAG.

- There is only one nucleotide different. We merge them.
Tour bus algorithm

```python
Algorithm Tour_Bus(H, s)
Require: H is the de Bruijn graph and s is an arbitrary node in H
Ensure: A graph formed after merging the bubbles
1: Set Q be a queue with one node s;
2: while Q ≠ ∅ do
3:   u = dequeue(Q);
4:   for each child v of u do
5:     if visited[v] = false then
6:       Set π(v) = u; /* set u as v’s parent in the BFS tree */
7:       Set visited[v] = true;
8:       enqueue(Q, v);
9:     else
10:        Find the lowest common ancestor c of u and v by π();
11:        if the paths c → u and c → v are similar enough then
12:          Merge the two paths and keep the path with the highest path weight;
13:        end if
14:     end if
15:   end for
16: end while
```
Example
Step 4: Remove erroneous connections

Remove edges with low coverage (a cutoff set by users)
Exercise

How to derive a reasonable threshold for this step?
Assembly quality

Simulations of Tour Bus. The genome of *E. coli* and 5-Mb samples of DNA from three other species (*S. cerevisiae, C. elegans, and H. sapiens*, respectively) were used to generate 35-bp read sets of varying read depths (X-axis of each plot). We measured the contig length N50 (Y-axis, log scale) after tip-clipping (black curve) then after the subsequent bubble smoothing (red curve). In the first column are the results for perfect, error-free reads. In the second column, we inserted errors in the reads at a rate of 1%. In the third column, we generated a slightly variant genome from the original by inserting random SNPs at a rate of 1 in 500. The reads were then generated with errors from both variants, thus simulating a diploid assembly.

![Graph showing assembly quality](image)

## Efficiency

### Table 1. Efficiency of the Velvet error-correction pipeline on the BAC data set

<table>
<thead>
<tr>
<th>Step</th>
<th>No. of nodes</th>
<th>N50 (bp)</th>
<th>Maximum length (bp)</th>
<th>Coverage (percent &gt;50 bp)</th>
<th>Coverage (percent &gt;100 bp)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>1,353,791</td>
<td>5</td>
<td>7</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Simplified</td>
<td>945,377</td>
<td>5</td>
<td>80</td>
<td>4.3</td>
<td>0.2</td>
</tr>
<tr>
<td>Tips clipped</td>
<td>4898</td>
<td>714</td>
<td>5037</td>
<td>93.5</td>
<td>78.7</td>
</tr>
<tr>
<td>Tour Bus</td>
<td>1147</td>
<td>1784</td>
<td>7038</td>
<td>93.4</td>
<td>90.1</td>
</tr>
<tr>
<td>Coverage cutoff</td>
<td>685</td>
<td>1958</td>
<td>7038</td>
<td>92.0</td>
<td>90.0</td>
</tr>
<tr>
<td>Ideal</td>
<td>620</td>
<td>2130</td>
<td>9045</td>
<td>93.7</td>
<td>91.9</td>
</tr>
</tbody>
</table>

Each line in this table represents a different stage in Velvet. The initial graph was built directly from the BAC reads. The second was the result of node concatenation. The next three graphs were the result of the three consecutive steps of error correction: tip clipping, Tour Bus, and coverage cutoff. The last graph was obtained by building the graph of the reference sequence then submitting it to Tour Bus, to simulate an error-free and gap-free assembly.

### Table 2. Efficiency of the Velvet error-correction pipeline on the *Streptococcus* data set

<table>
<thead>
<tr>
<th>Step</th>
<th>No. of nodes</th>
<th>N50 (bp)</th>
<th>Maximum length (bp)</th>
<th>Coverage (percent &gt;50 bp)</th>
<th>Coverage (percent &gt;100 bp)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>3,621,167</td>
<td>16</td>
<td>16</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Simplified</td>
<td>2,222,845</td>
<td>16</td>
<td>44</td>
<td>0.1</td>
<td>0</td>
</tr>
<tr>
<td>Tips clipped</td>
<td>15,267</td>
<td>2195</td>
<td>7949</td>
<td>96.2</td>
<td>95.4</td>
</tr>
<tr>
<td>Tour Bus</td>
<td>3303</td>
<td>4334</td>
<td>17,811</td>
<td>96.8</td>
<td>96.4</td>
</tr>
<tr>
<td>Coverage cutoff</td>
<td>1496</td>
<td>8564</td>
<td>29,856</td>
<td>96.9</td>
<td>96.5</td>
</tr>
<tr>
<td>Ideal</td>
<td>1305</td>
<td>9609</td>
<td>29,856</td>
<td>97.0</td>
<td>96.8</td>
</tr>
</tbody>
</table>

Recall this example, where $K = 3$

Genome = AAGATCGATGATTT

$\mathcal{R} = \{ \text{AAGATC}, \text{GATCGAT}, \text{CGATGA}, \text{ATGATT}}, \text{GATTT} \}$

$DB_3(\mathcal{R})$:

Two possible Eulerian paths but can’t tell which is real

AAGATCGATGATTT

AAGATGATCGATTT
Consider $K = 4$

Genome = AAGATCGATGATTT

$\mathcal{R} = \{\text{AAGATC, GATCGAT, CGATGA, ATGATT, GATTT}\}$

$\text{DB}_4(\mathcal{R})$:

A unique Eulerian path:

AAGATCGATGATTT
Consider $K = 5$

Genome = AAGATCGATGATTT

$\mathcal{R} = \{\text{AAGATC, GATCGAT, CGATGA, ATGATT, GATTT}\}$

$\text{DB}_5(\mathcal{R})$:

No Eulerian paths; fragmented graphs give these strings:

AAGATC, GATCGAT, CGATGA, ATGATT, GATTT
How to choose suitable K

Large K

*K-mers more likely to have errors*

# of correct K-mers is reduced

More genome coverage gaps

Small K

*More likely to be repeated*

*Short repeats create loops and branches in de Bruijn graph*
Instead of a fixed $K$, try different values

*When $K$ is small, we get short but high-quality contigs*

*Use them to correct errors in reads*

*Then, increment $K$ and try again*

Algorithm IDBA($\mathcal{R}, k_{min}, k_{max}$)

Require: $\mathcal{R}$ is a set of reads and $k_{min}$ and $k_{max}$ are de Bruijn graph parameter

Ensure: A set of contigs

1: for $k = k_{min}$ to $k_{max}$ do
2:   Generate the de Bruijn graph $H_k$ for $\mathcal{R}$;
3:   Remove tips;
4:   Merge bubbles;
5:   Remove nodes with multiplicity $\leq m$;
6:   Extract all maximal simple paths in $H_k$ as contigs;
7:   All reads in $\mathcal{R}$ are aligned to the computed contigs;
8:   The mismatch in the read is corrected if 80\% of reads aligned to the same position has the correct base;
9: end for
10: Extract all maximal simple paths in $H_{k_{max}}$ as contigs;
Low-coverage regions remain problematic

A region is covered by two reads ...AACT and ACTG...

At \( K = 3 \), \( \text{DB}_K(\mathcal{R}) \) shows a path \( \text{AAC} \rightarrow \text{ACT} \rightarrow \text{CTG} \) but cannot output the contig ... AACTG ...

\text{AACTG} \text{ is not in any read }

\text{ACT has 2 incoming & 2 outgoing edges due to TACTT}

Similar problems exist at \( K = 4, 5, \ldots \)
IDBA-UD $\approx$ IDBA + local assembly

Using information from paired-end read mapping, the two copies of ACT can be separated

Then, …AACTG … can be produced
## Performance


| k  | Contigs | | | | | | Scaffolds | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
|   | No. | N50 | Max | Cov | Sub.err | Err no. | No. | N50 | Max | Cov | Sub.err | Err no. | Time (s) | Mem (M) |
|   | len | (%) | (len) |   | (len) | | len | (%) | (len) |   | (len) | |
| IDBA-UD | 20–100 | 210 | 36 513 | 201 860 | 99.56 | 0.0225 | 104 437 | 83 | 194 322 | 406 269 | 99.55 | 0.0218 | 53 784 | 63 | 432 |
| SOAPdenovo | 31 | 3346 | 1584 | 8691 | 98.36 | 0.0572 | 1 079 112k | 147 | 121 214 | 246 514 | 92.50 | 0.0483 | 1 087 283k | 31 | 852 |
| Velvet | 21 | 473 | 13 761 | 48 489 | 98.09 | 0.0323 | 515k | 111 | 111 871 | 225 438 | 96.81 | 0.0291 | 667k | 43 | 526 |
| IDBA | 20–40 | 672 | 8350 | 37 391 | 98.52 | 0.0164 | 33 301 | 60 | 119 931 | 308 798 | 97.55 | 0.0161 | 39 420 | 24 | 414 |

Table 2. The assembly results on simulated 10× length-100 reads of *L. plantarum* (~3.3 Mb) with 1% error rate.
Limitations

De Bruijn graph is big

*Need lots of memory to run*

Cannot use connectivity of paired-end reads before scaffolding

*SPAdes uses paired de Bruijn graph to capture this info*

If there are long repeats, this approach may fail to get long contigs
Exercise

IDBA tries increasingly bigger values for $K$

*i.e., it must assemble the genome multiple times*

Would it be possible to guess a good value for $K$ by just taking a quick look at the read set?

How?

---

**Hypothesis a la KmerGenie**

Empirical evidence from KmerGenie

When a value $K^*$ of $K$ maximizes the set of genomic K-mer species observed in a read set $\mathcal{R}$ for a genome $G$, then $K^*$ is the best value for producing the genome assembly from $\mathcal{R}$ for $G$.

A genomic K-mer species is a K-mer species that indeed appears in the genome $G$. 
Must read

Velvet


https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2336801/

IDBA-UD


Good to read

**SOAPdenovo**

**SPAdes**

**KmerGenie**