De novo meta-assembly of ultra-deep sequencing data

Hamid Mirebrahim¹, Timothy J. Close² and Stefano Lonardi¹

¹Department of Computer Science and Engineering
²Department of Botany and Plant Sciences
Ultra-deep sequencing (>1,000x coverage) is possible and feasible, expected to become more common.
Expectation: more data → ‘better’ assemblies

**Expectation:** more data $\rightarrow$ ‘better’ assemblies

More data are not necessarily better: why?

- Possible “suspects”
  - Sequencing errors
  - Highly uneven coverage
  - Read duplication / PCR amplification bias
  - Chimeric reads
  - “Imperfections” in the assembly algorithms

Possible solutions

• “Classic” error correction
  – based on rare k-mers
  – ineffective for ultra-deep sequencing data

• Down-sampling
  – disregard a fraction of the input reads, according to some predetermined strategy
  – it may remove “critical” reads (i.e., rare error-free reads that can help bridge or fill assembly gaps)
  – not very effective

• SLICEMLBLER (next)

1: “Slice” the input
- The set of input reads is partitioned into $n$ distinct slices, where $n = \text{the depth of coverage for the whole input read set / the desired depth of coverage for each slice}$
- Each slice contains approximately the same number of reads
2: Assemble the reads in each slice

- Each of the $n$ slices is assembled independently using a standard assembler (Velvet, SPAdes, IDBA, etc.)
- Each assembly is expected to contain a mix of high-quality and low-quality contigs
3: Find frequently occurring substrings (FOS)
   - Identify high-quality contigs (or fraction thereof) or FOS
   - Use a generalized suffix-tree for efficiency
   - Remove tandem repeats at the end of FOS
Finding FOS efficiently

**Definition:** Given integers $k$ and $l$, a FOS is a maximal substring $r$ such that $|r| \geq l$ and it appears in at least $k$ assemblies.

- Build a *generalized suffix tree* on the contigs of the $n$ assemblies (and their reverse complement).
- Each input assembly is assigned a distinct “color” (Hui, CPM’92).
- Annotate each internal node $u$ with the number of distinct colors in the subtree rooted at $u$.
- In order to find FOS, determine all the deepest internal nodes (deeper than $l$) which have a color count of least $k$.
- Building and annotating the suffix tree can be done in linear time.

![Suffix Tree Diagram]
SLICEMBLER algorithm

4: Merge FOS
- When detected FOS are overlapping they can be merged to obtain longer FOS
- Merge based on exact suffix-prefix overlap and bridge reads (similar to scaffolding)
5: Filter reads

- Input reads are mapped to FOS (e.g., BWA)
- Any read that maps to a contig in the current assembly is removed from input (unless it maps close to the end)
- Only the remaining reads are re-assembled in the next iteration
Experimental results

• Real ultra-deep sequencing data
  • Sequenced 16 barley BACs with Illumina HiSeq
  • Depth of coverage: 8,000x-15,000x
  • Paired-end reads (avg length ~88bp after trimming)
  • Selected 8,000x paired-end reads
  • High-quality references are available for five BACs

• Synthetic ultra-deep reads (wgsim)
  • Generated from the reference barley BACs
  • Paired-end reads (2x100 bp)
  • Various levels of coverage and error rates
Experimental results (real barley BACs)

Number of contigs

Longest contig (Kbps)

n50 (Kbps)

Reference covered (%)

Duplication ratio (%)

# of misassemblies

Statistics collected with QUAST for contigs longer than 500 bp

Velvet (8000x)
Racer+Velvet (8000x)
Velvet (800x)
Slicembler+Velvet (8000x)
## Assembly quality vs. base assembler (real barley BAC)

<table>
<thead>
<tr>
<th>Method</th>
<th>Number of contigs</th>
<th>% ref covered</th>
<th>Duplication ratio</th>
<th>Mismatches per 100Kbp</th>
<th>N50</th>
<th>Longest contig</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDBA (8,000x)</td>
<td>34</td>
<td>97.0%</td>
<td>1.010</td>
<td>0.93</td>
<td>7,335</td>
<td>13,889</td>
</tr>
<tr>
<td>SLICEMBLER + IDBA (10 slices of 800x)</td>
<td>13</td>
<td>97.0%</td>
<td>1.010</td>
<td>1.1</td>
<td>16,121</td>
<td>31,161</td>
</tr>
<tr>
<td>Velvet (8,000x)</td>
<td>39</td>
<td>94.7%</td>
<td>1.027</td>
<td>20.0</td>
<td>3,649</td>
<td>16,048</td>
</tr>
<tr>
<td>SLICEMBLER + Velvet (10 slices of 800x)</td>
<td>14</td>
<td>95.1%</td>
<td>1.001</td>
<td>0</td>
<td>12,178</td>
<td>16,128</td>
</tr>
<tr>
<td>SPAdes (8,000x)</td>
<td>49</td>
<td>95.7%</td>
<td>1.006</td>
<td>0.94</td>
<td>9,129</td>
<td>21,872</td>
</tr>
<tr>
<td>SLICEMBLER + SPAdes (10 slices of 800x)</td>
<td>11</td>
<td>96.9%</td>
<td>1.024</td>
<td>1.2</td>
<td>27,685</td>
<td>31,158</td>
</tr>
<tr>
<td>Ray (8,000x)</td>
<td>35</td>
<td>80.0%</td>
<td>1.003</td>
<td>0</td>
<td>3,996</td>
<td>7,186</td>
</tr>
<tr>
<td>SLICEMBLER + Ray (10 slices of 800x)</td>
<td>24</td>
<td>88.0%</td>
<td>1.000</td>
<td>0</td>
<td>7,192</td>
<td>12,842</td>
</tr>
</tbody>
</table>

Statistics collected with QUAST for contigs longer than 500 bp
Assembly quality vs. sequencing error rate (simulated barley BACs)

Statistics collected with QUAST for contigs longer than 500 bp
Conclusions

- Modern *de novo* genome assemblers seem unable to take advantage of ultra-deep coverage

- **SLICEMBLER** is an iterative meta-assembler that takes advantage of the whole dataset and due to its “majority voting” scheme
  - Is more resilient to sequencing errors than its base assemblers
  - Almost never incorporates misassemblies in the consensus assembly

- **SLICEMBLER** is available at [www.slicembler.cs.ucr.edu](http://www.slicembler.cs.ucr.edu)

- **SLICEMBLER** is slow, but a C++ implementation called **SLICEMBLER++**, will be available soon
Thank you

DBI-1062301 and IIS-1302134

NIFA 2009-65300-05645

AID-OAA-A-13-00070

www.slicembler.cs.ucr.edu
Will sequencing cost continue to decrease?

“It’s difficult to make predictions, especially about the future”
Experimental results
(real barley BACs)
Frequently occurring substrings (FOS)
Varying sequencing error rate (Velvet)

Varying sequencing error rate (IDBA)

Sketch of the algorithm

- **partition** the input read set to $n$ slices, each of which has $D_s$ coverage
- **while** the cumulative length of FOS is less than the length of genome
  - **assemble** the reads in all $n$ slices individually
  - **create** a suffix-tree from the $n$ assemblies and their reverse compl
  - **assign** $k = n$, $l = \frac{l_{target}}{5}$
  - **while** ($l > l_{min}$)
    - **find** FOS longer than $l$, appearing in at least $k$ assemblies
    - **if** FOS found **then merge** them with the previous FOS, **break**
    - **else if** $k > n/2$ **then assign** $k = k-1$
    - **else assign** $l = l/2$, $k = n$
    - **if** ($l \leq l_{min}$) **and** (no FOS were found) **then break**
  - **map** reads to FOS and **eliminate** mapped reads from the input
- **report** FOS
Percentage of reads that map exactly to the reference, iteration by iteration
Assembly quality vs. slice coverage

<table>
<thead>
<tr>
<th></th>
<th>500x</th>
<th>1,000x</th>
<th>2,500x</th>
<th>5,000x</th>
<th>7,500x</th>
<th>10,000x</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of contigs</td>
<td>20</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>18</td>
<td>38</td>
</tr>
<tr>
<td>Longest contig</td>
<td>27,364</td>
<td>31,823</td>
<td>31,946</td>
<td><strong>31,950</strong></td>
<td>21,865</td>
<td>9,425</td>
</tr>
<tr>
<td>N50</td>
<td>6,707</td>
<td>26,275</td>
<td><strong>26,288</strong></td>
<td>26,267</td>
<td>12,428</td>
<td>3,643</td>
</tr>
<tr>
<td>Percent Refer. Covered</td>
<td>90.6%</td>
<td>88.7%</td>
<td><strong>94%</strong></td>
<td>93.9%</td>
<td>92.9%</td>
<td>84.7%</td>
</tr>
<tr>
<td>Duplication ratio</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Mismatches per 100kbp</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Another application for SLICEMLBLER: co-assembly of single cell sequencing data