A new strategy for genome sequencing

J. Craig Venter, Hamilton O. Smith and Leroy Hood

(Translating the cloning jargon)

<table>
<thead>
<tr>
<th>Vector</th>
<th>Human-DNA insert size range</th>
<th>Number of clones required to cover the human genome</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yeast artificial chromosome (YAC)</td>
<td>100–2,000 kb</td>
<td>3,000 (1,000 kb)</td>
</tr>
<tr>
<td>Bacterial artificial chromosome (BAC)</td>
<td>80–350 kb</td>
<td>20,000 (150 kb)</td>
</tr>
<tr>
<td>Cosmid</td>
<td>30–45 kb</td>
<td>75,000 (40 kb)</td>
</tr>
<tr>
<td>Plasmid</td>
<td>3–10 kb</td>
<td>600,000 (5 kb)</td>
</tr>
<tr>
<td>M13 phage</td>
<td>1 kb</td>
<td>3,000,000 (1 kb)</td>
</tr>
</tbody>
</table>
**Thinking about the basic shotgun concept**

- Start with a very large set of random sequencing reads
- How might we match up the overlapping sequences?
- How can we assemble the overlapping reads together in order to derive the genome?

**Thinking about the basic shotgun concept**

- At a high level, the first genomes were sequenced by comparing pairs of reads to find overlapping reads
- Then, building a graph (i.e., a network) to represent those relationships
- The genome sequence is a “walk” across that graph
The “Overlap-Layout-Consensus” method

**Overlap:** Compare all pairs of reads (allow some low level of mismatches)

**Layout:** Construct a graph describing the overlaps

- Simplify the graph
- Find the simplest path through the graph

**Consensus:** Reconcile errors among reads along that path to find the consensus sequence

---

Building an overlap graph

---

**Building an overlap graph**

Reads

A → B → C → D → E → F → G → H → I

Overlap graph

1. Remove all contained nodes & edges going to them

**Simplifying an overlap graph**

1. Remove all contained nodes & edges going to them
Simplifying an overlap graph

2. Transitive edge removal:
   Given A – B – C and A – C, remove A – C

3. If un-branched, calculate consensus sequence
   If branched, assemble un-branched bits and then decide how they fit together
Simplifying an overlap graph

This basic strategy was used for most of the early genomes. Also useful: “mate pairs”
GigAssembler (used to assemble the public human genome project sequence)

Jim Kent  David Haussler

Whole genome Assembly: big picture

http://www.nature.com/scitable/content/anatomy-of-whole-genome-assembly-20429
GigAssembler – Preprocessing

1. Decontaminating & Repeat Masking.
2. Aligning of mRNAs, ESTs, BAC ends & paired reads against initial sequence contigs.
   - psLayout → BLAT
3. Creating an input directory (folder) structure.

RepBase + RepeatMasker
GigAssembler: Build merged sequence contigs ("rafts")

Figure 1 Two sequences overlapping end to end. The sequences are represented as dashes. The aligning regions are joined by vertical bars. End-to-end overlap is an extremely strong indication that two sequences should be joined into a contig.

Sequencing quality (Phred Score)
Sequencing quality (Phred Score)

\[ Q = -10 \log_{10} P \]

or

\[ P = 10^{-Q/10} \]

Phred quality scores are logarithmically linked to error probabilities

<table>
<thead>
<tr>
<th>Phred Quality Score</th>
<th>Probability of incorrect base call</th>
<th>Base call accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1 in 10</td>
<td>90 %</td>
</tr>
<tr>
<td>20</td>
<td>1 in 100</td>
<td>99 %</td>
</tr>
<tr>
<td>30</td>
<td>1 in 1000</td>
<td>99.9 %</td>
</tr>
<tr>
<td>40</td>
<td>1 in 10000</td>
<td>99.99 %</td>
</tr>
<tr>
<td>50</td>
<td>1 in 100000</td>
<td>99.999 %</td>
</tr>
</tbody>
</table>

http://en.wikipedia.org/wiki/Phred_quality_score

GigAssembler: Build merged sequence contigs ("rafts")

Figure 2: Two sequences with tails. The nonaligning regions on either side can be classified into 'extensions' and 'tails.' Short tails are fairly common even when two sequences should be joined into a contig because of poor quality sequence near the ends and occasional chimeric reads. Long tails, however, are generally a sign that the alignment is merely due to the sequences sharing a repeating element.
GigAssembler: Build merged sequence contigs ("rafts")

Figure 3  Merging into a raft. A contig ("raft") of three sequences: A, B, and C has already been constructed by GigAssembler. The program now examines an alignment between sequence C and a new sequence, D, to see whether D should also be added to the raft. The parts of D marked with +\(\)s are compatible with the raft because of the C/D alignment. The program must also check that the parts of D marked with ?\(\)s are compatible with the raft by examining other alignments.

GigAssembler: Build sequenced clone contigs ("barges")

Figure 4  Three overlapping draft clones: A, B, and C. Each clone has two initial sequence contigs. Note that initial sequence contigs a1, b1, and a2 overlap as do b2 and c1.
GigAssembler: Build a “raft-ordering” graph

- Add information from mRNAs, ESTs, paired plasmid reads, BAC end pairs: building a “bridge”
  - Different weight to different data type: (mRNA ~ highest)
  - Conflicts with the graph as constructed so far are rejected.
- Build a sequence path through each raft.
- Fill the gap with N’s.
  - 100: between rafts
  - 50,000: between bridged barges
Finding the shortest path across the ordering graph using the Bellman-Ford algorithm


Find the shortest path to all nodes.

Take every edge and try to relax it (N – 1 times where N is the count of nodes)
Find the shortest path to all nodes.

Take every edge and try to relax it \((N - 1)\) times where \(N\) is the count of nodes.

Edward Marcotte/Univ. of Texas/BIO337/Spring 2014
Find the shortest path to all nodes.

Take every edge and try to relax it \((N - 1)\) times where \(N\) is the count of nodes.
Find the shortest path to all nodes.

Take every edge and try to relax it (N – 1 times where N is the count of nodes)
Modern assemblers now work a bit differently, using so-called DeBruijn graphs:

Here’s what we saw before:

In Overlap-Layout-Consensus:
- Nodes are reads
- Edges are overlaps

Answer: A-D-C-B-E
Modern assemblers now work a bit differently, using so-called **DeBruijn graphs:**

In a DeBruijn graph:
- Vertices are \((k-1)\)-mers
- Edges are \(k\)-mers

When a reference genome is assembled, new sequencing data can ‘simply’ be mapped to the reference.
Mapping reads to assembled genomes

Table 1 A selection of short-read analysis software

<table>
<thead>
<tr>
<th>Program</th>
<th>Website</th>
<th>Open source?</th>
<th>Handles ABI color space?</th>
<th>Maximum read length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bowtie</td>
<td><a href="http://bowtie.cbcb.umd.edu">http://bowtie.cbcb.umd.edu</a></td>
<td>Yes</td>
<td>No</td>
<td>None</td>
</tr>
<tr>
<td>BWA</td>
<td><a href="http://maq.sourceforge.net/bwa-man.shtml">http://maq.sourceforge.net/bwa-man.shtml</a></td>
<td>Yes</td>
<td>Yes</td>
<td>None</td>
</tr>
<tr>
<td>Maq</td>
<td><a href="http://maq.sourceforge.net">http://maq.sourceforge.net</a></td>
<td>Yes</td>
<td>Yes</td>
<td>127</td>
</tr>
<tr>
<td>Mosaic</td>
<td><a href="http://bioinformatics.bc.edu/marshlab/Mosaic">http://bioinformatics.bc.edu/marshlab/Mosaic</a></td>
<td>No</td>
<td>Yes</td>
<td>None</td>
</tr>
<tr>
<td>Novoalign</td>
<td><a href="http://www.novocraft.com">http://www.novocraft.com</a></td>
<td>No</td>
<td>No</td>
<td>None</td>
</tr>
<tr>
<td>SOAP2</td>
<td><a href="http://soap.genomics.org.cn">http://soap.genomics.org.cn</a></td>
<td>No</td>
<td>No</td>
<td>60</td>
</tr>
<tr>
<td>ZOOM</td>
<td><a href="http://www.bioinfor.com">http://www.bioinfor.com</a></td>
<td>No</td>
<td>Yes</td>
<td>240</td>
</tr>
</tbody>
</table>

Edward Marcotte/Univ. of Texas/BIO337/Spring 2014

Mapping strategies

Edward Marcotte/Univ. of Texas/BIO337/Spring 2014

Trapnell C, Salzberg SL, Nat. Biotech., 2009
Burroughs-Wheeler transform indexing

BWT is often used for file compression (like bzip2), here used to make a fast ‘lookup’ index in a genome

BWT = ‘reversible block-sorting’

Input  SIX.MIXED.PIXIES.SIFT.SIXTY.PIXIE.DUST.BOXES

Forward BWT

Output   TĘXYDST.E.IXIXIXXSSMPPS.B..E.S.EUSFXDIIIOIIIT

Reverse BWT

Recovered input   SIX.MIXED.PIXIES.SIFT.SIXTY.PIXIE.DUST.BOXES

This sequence is more compressible

Burroughs-Wheeler transform indexing


Edward Marcotte / Univ. of Texas / BIO337 / Spring 2014
### Burroughs-Wheeler transform indexing

#### Sorting All Rows in Alphabetical Order

<table>
<thead>
<tr>
<th>ANANA</th>
<th>^B</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANA</td>
<td>^BAN</td>
</tr>
<tr>
<td>A</td>
<td>^BANAN</td>
</tr>
<tr>
<td>BANANA</td>
<td>^</td>
</tr>
<tr>
<td>NANA</td>
<td>^BA</td>
</tr>
<tr>
<td>NA</td>
<td>^BANA</td>
</tr>
<tr>
<td>^BANANA</td>
<td></td>
</tr>
<tr>
<td>^BANANA</td>
<td></td>
</tr>
</tbody>
</table>


### Burroughs-Wheeler transform indexing

#### Taking Last Column

<table>
<thead>
<tr>
<th>ANANA</th>
<th>^B</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANA</td>
<td>^BAN</td>
</tr>
<tr>
<td>A</td>
<td>^BANAN</td>
</tr>
<tr>
<td>BANANA</td>
<td>^</td>
</tr>
<tr>
<td>NANA</td>
<td>^BA</td>
</tr>
<tr>
<td>NA</td>
<td>^BANA</td>
</tr>
<tr>
<td>^BANANA</td>
<td></td>
</tr>
<tr>
<td>^BANANA</td>
<td></td>
</tr>
</tbody>
</table>

Burroughs-Wheeler transform indexing

BWT is remarkable because it is reversible.

Any ideas as how you might reverse it?

Burroughs-Wheeler transform indexing

<table>
<thead>
<tr>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>BNN^AA</td>
</tr>
</tbody>
</table>
### Burroughs-Wheeler transform indexing

<table>
<thead>
<tr>
<th>Add 1</th>
<th>Sort 1</th>
<th>Add 2</th>
<th>Sort 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>A</td>
<td>BA</td>
<td>AN</td>
</tr>
<tr>
<td>N</td>
<td>A</td>
<td>NA</td>
<td>AN</td>
</tr>
<tr>
<td>N</td>
<td>A</td>
<td>NA</td>
<td>A</td>
</tr>
<tr>
<td>^</td>
<td>B</td>
<td>^B</td>
<td>BA</td>
</tr>
<tr>
<td>A</td>
<td>N</td>
<td>AN</td>
<td>NA</td>
</tr>
<tr>
<td>A</td>
<td>^</td>
<td>^</td>
<td>^B</td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td>^</td>
</tr>
</tbody>
</table>

- Write the sequence as the last column
- Sort it...
- Add the columns...
- Sort those...

---

### Burroughs-Wheeler transform indexing

<table>
<thead>
<tr>
<th>Add 3</th>
<th>Sort 3</th>
<th>Add 4</th>
<th>Sort 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAN</td>
<td>ANA</td>
<td>BANA</td>
<td>ANAN</td>
</tr>
<tr>
<td>NAN</td>
<td>ANA</td>
<td>NANA</td>
<td>ANA</td>
</tr>
<tr>
<td>NA</td>
<td>^</td>
<td>NA</td>
<td>^</td>
</tr>
<tr>
<td>^BA</td>
<td>BAN</td>
<td>^BAN</td>
<td>BANA</td>
</tr>
<tr>
<td>ANA</td>
<td>NAN</td>
<td>ANAN</td>
<td>NANA</td>
</tr>
<tr>
<td>ANA</td>
<td>NA</td>
<td>ANA</td>
<td>NA</td>
</tr>
<tr>
<td>^B</td>
<td>^BA</td>
<td>^BA</td>
<td>^B</td>
</tr>
<tr>
<td>A</td>
<td>^B</td>
<td>A</td>
<td>^B</td>
</tr>
</tbody>
</table>

- Add the columns...
- Sort those...
- Add the columns...
- Sort those...

---

### Burroughs-Wheeler transform indexing

<table>
<thead>
<tr>
<th>Add 5</th>
<th>Sort 5</th>
<th>Add 6</th>
<th>Sort 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>BANAN</td>
<td>ANANA</td>
<td>BANANA</td>
<td>ANANA</td>
</tr>
<tr>
<td>NANA</td>
<td>ANA^</td>
<td>NANA</td>
<td>ANA^</td>
</tr>
<tr>
<td>NA^B</td>
<td>A^BA</td>
<td>NA^B</td>
<td>A^BAN</td>
</tr>
<tr>
<td>^BANA</td>
<td>BANAN</td>
<td>^BANAN</td>
<td>BANANA</td>
</tr>
<tr>
<td>ANANA</td>
<td>NANA</td>
<td>ANANA</td>
<td>NANA</td>
</tr>
<tr>
<td>ANA^</td>
<td>NA^B</td>
<td>ANA^</td>
<td>NA^B</td>
</tr>
<tr>
<td>^BANAN</td>
<td>^BANA</td>
<td>^BANAN</td>
<td>^BANA</td>
</tr>
<tr>
<td>A^BA</td>
<td>^BANAN</td>
<td>A^BA</td>
<td>^BANAN</td>
</tr>
</tbody>
</table>

Add the columns... Sort those... Add the columns... Sort those...

---

### Burroughs-Wheeler transform indexing

<table>
<thead>
<tr>
<th>Add 7</th>
<th>Sort 7</th>
<th>Add 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>BANANA</td>
<td>ANANA^</td>
<td>BANANA</td>
</tr>
<tr>
<td>NANA^B</td>
<td>ANA^BA</td>
<td>NANA^B</td>
</tr>
<tr>
<td>NA^BAN</td>
<td>A^BANA</td>
<td>NA^BAN</td>
</tr>
<tr>
<td>^BANANA</td>
<td>BANANA</td>
<td>^BANANA</td>
</tr>
<tr>
<td>ANANA^</td>
<td>NANA^B</td>
<td>ANANA^B</td>
</tr>
<tr>
<td>ANA^BA</td>
<td>NA^BAN</td>
<td>ANA^BAN</td>
</tr>
<tr>
<td>^BANAN</td>
<td>^BANANA</td>
<td>^BANANA</td>
</tr>
<tr>
<td>A^BANA</td>
<td>^BANAN</td>
<td>A^BAN</td>
</tr>
</tbody>
</table>

Add the columns... Sort those... Add the columns...

The row with the "end of file" character at the end is the original text

---

Burroughs-Wheeler transform indexing

<table>
<thead>
<tr>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>^BANANA</td>
</tr>
</tbody>
</table>