CS681: Advanced Topics in Computational Biology

Can Alkan
EA224
calkan@cs.bilkent.edu.tr

http://www.cs.bilkent.edu.tr/~calkan/teaching/cs681/
Genome Assembly

Test genome

Random shearing and Size-selection

Sequencing

Contigs/scaffolds

Assemble
Graph problems in assembly

- Hamiltonian cycle/path
  - Typically used in overlap graphs
  - NP-hard
- Eulerian cycle/path
  - Typically used in de Bruijn graphs
The Bridge Obsession Problem

Find a tour crossing every bridge just once
Leonhard Euler, 1735

Bridges of Königsberg (Kaliningrad)
Eulerian Cycle Problem

- Find a cycle that visits every edge exactly once
- Linear time

More complicated Königsberg
Hamiltonian Cycle Problem

- Find a cycle that visits every vertex exactly once
- NP – complete

Game invented by Sir William Hamilton in 1857
Traveling salesman problem

- TSP: find the shortest path that visits every vertex once
  - Directed / undirected
  - NP-complete
  - Exact solutions:
    - Held-Karp: $O(n^2 2^n)$
  - Heuristic
    - Lin-Kernighan
Genome assembly problem is finding **shortest common superstring** of a set of sequences (reads):

- Given strings \( \{s_1, s_2, \ldots, s_n\} \); find the superstring \( T \) such that every \( s_i \) is a substring of \( T \)
- NP-hard problem
- Greedy approximation algorithm
  - Works for simple (low-repeat) genomes
The Shortest Superstring problem

Set of strings: \{000, 001, 010, 011, 100, 101, 110, 111\}

Concatenation

Superstring

000 001 010 011 100 101 110 111

Shortest superstring

0 0 0 1 1 1 0 1 0 0

0 0 0 1 0 1 0
Reducing SSP to TSP

- Define $overlap(s_i, s_j)$ as the length of the longest prefix of $s_j$ that matches a suffix of $s_i$.

```
aaaggcatcaaatctaaaggcatcaaa
   aaaggcatcaaaatctaaaggcatcaaa
overlap=12
```
Reducing SSP to TSP

- Define $overlap(s_i, s_j)$ as the length of the longest prefix of $s_j$ that matches a suffix of $s_i$.
  
  `aaaggcatcaaatctaaaggcatcaaa`
  `aaaggcatcaaaatctaaaggcatcaaa`

- Construct a graph with $n$ vertices representing the $n$ strings $s_1, s_2, \ldots, s_n$.

- Insert edges of length $overlap(s_i, s_j)$ between vertices $s_i$ and $s_j$.

- Find the shortest path which visits every vertex exactly once. This is the **Traveling Salesman Problem** (TSP), which is also NP – complete.
Reducing SSP to TSP (cont’d)
SSP to TSP: An Example

\[
S = \{ \text{ATC, CCA, CAG, TCC, AGT} \}
\]
Assembly paradigms

- Overlap-layout-consensus
  - greedy (TIGR Assembler, phrap, CAP3...)
  - graph-based (Celera Assembler, Arachne)
    - SGA for NGS platforms

- Eulerian path on de Bruijn graphs (especially useful for short read sequencing)
  - EULER, Velvet, ABySS, ALLPATHS-LG, Cortex, etc.
Overlap-Layout-Consensus

- Traditional assemblers: Phrap, Arachne, Celera etc.
- Short reads: Edena, SGA
- Generally more expensive computationally
  - Pairwise global alignments
- However, as reads get longer (>200bp ?) produce better results
  - They use the alignments of entire reads not isolated $k$-mer overlaps
Overlap-Layout-Consensus

**Assemblers:** ARACHNE, PHRAP, CAP, TIGR, CELERA

**Overlap:** find potentially overlapping reads

**Layout:** merge reads into contigs and contigs into scaffolds

**Consensus:** derive the DNA sequence and correct read errors

..ACGATTACAATAGGTT..
A quick example

TAGTCGAGGCTTTTAGATCCGATGAGGCTTTAGAGACAG

AGTCGAG CTTTATA CGATGAG CTTTTAGA
GTCGAGG TTAGATC ATGAGGC GAGACAG
GAGGCTC ATCCGAT AGGCTTT GAGACAG
AGTCGAG TAGATCC ATGAGGC TAGAGA

TAGTCGA CTTTTAGA CCGATGA TTAGAGA
CGAGGCT AGATCCG TGAGGCT AGAGAC

TAGTCGA GCTTTAG TCCGATG GCT CTA
TCGACGC GATCCGA GAGGCTT AGAGACA

TAGTCGA TTAGATC GATGAGG TTTAGAG

GTCGAGG TCTAGAT ATGAGGC TAGAGAC
AGGCTTT ATCCGAT AGGGCTT GAGACAG
AGTCGAG TTAGATT ATGAGGC AGAGACA

GGGTTTA TCCGATG TTTAGAG
GCAGGCT TAGATCC TGAGGCT GAGACAG
AGTCGAG TTTAGATC ATGAGGC TTAGAGA
GAGGCTT GATCCGA GAGGCTT GAGACAG
A quick example

AGTCGAG CTTTAGA CGATGAG CTTTAGA
GTCGAGG TTAGATC ATGAGGC GAGACAG
GAGGCTC ATCCGAT AGGCTTT GAGACAG
AGTCGAG TAGATCC ATGAGGC TAGAGAA
TAGTCGA CTTTAGA CCGATGA TTAGAGA
CGAGGCT AGATCCG TGAGGCT AGAGACA
TAGTCGA GCTTTTAG TCCGATG GCTCTAG
TCGACGC GATCCGA GAGGCTT AGAGACA
TAGTCGA TTAGATC GATGAGG TTTAGAG
GTCGAGG TCTAGAT ATGAGGC TAGAGAC
AGGCTTT ATCCGAT AGGCTTT GAGACAG
AGTCGAG TTAGATT ATGAGGC AGAGACA
GGCTTTA TCCGATG TTTAGAG
CGAGGCT TAGATCC TGAGGCT GAGACAG
AGTCGAG TTTAGATC ATGAGGC TTAGAGA
GAGGCTT GATCCGA GAGGCTT GAGACAG
A quick example
A quick example
A quick example
Overlap

- Find the best match between the suffix of one read and the prefix of another

- Due to sequencing errors, need to use dynamic programming to find the optimal *overlap alignment*

- Apply a filtration method to filter out pairs of fragments that do not share a significantly long common substring
Overlapping Reads

- Sort all k-mers in reads \((k \sim 24)\)
- Find pairs of reads sharing a k-mer
- Extend to full alignment – throw away if not >95% similar
Overlapping Reads and Repeats

- A $k$-mer that appears $N$ times, initiates $N^2$ comparisons

- For an Alu that appears $10^6$ times $\rightarrow 10^{12}$ comparisons – too much

- **Solution:**
  Discard all $k$-mers that appear more than $t \times \text{Coverage, } (t \sim 10)$
Finding Overlapping Reads

Create local multiple alignments from the overlapping reads
Finding Overlapping Reads (cont’d)

• Correct errors using multiple alignment

• Score alignments
• Accept alignments with good scores
Repeats are a major challenge

Do two aligned fragments really overlap, or are they from two copies of a repeat?

Solution: repeat masking – hide the repeats!!!

Masking results in high rate of misassembly (up to 20%)

Misassembly means alot more work at the finishing step
Merge Reads into Contigs

Merge reads up to potential repeat boundaries
Repeats, Errors, and Contig Lengths

- Repeats shorter than read length are OK
- Repeats with more base pair differences than sequencing error rate are OK
- To make a smaller portion of the genome appear repetitive, try to:
  - Increase read length
  - Decrease sequencing error rate
Error Correction

Role of error correction:

Discards ~90% of single-letter sequencing errors

- decreases error rate
  - decreases effective repeat content
  - increases contig length
Link Contigs into Scaffolds

- Normal density
- Too dense: Overcollapsed?
- Inconsistent links: Overcollapsed?
Find all links between unique contigs

Connect contigs incrementally, if ≥ 2 links
Fill gaps in scaffolds with paths of overcollapsed contigs
Define $T$: contigs linked to either $A$ or $B$

Fill gap between $A$ and $B$ if there is a path in $G$ passing only from contigs in $T$
Consensus

- A consensus sequence is derived from a profile of the assembled fragments
- A sufficient number of reads is required to ensure a statistically significant consensus
- Reading errors are corrected
Derive Consensus Sequence

Derive *multiple alignment* from pairwise read alignments

Derive each consensus base by weighted voting
Celera Assembler

Trim & Screen

Overlapper

Unitiger

Scaffold

Repeat Res I, II

Find all overlaps ≥ 40bp allowing 6% mismatch.

implies

TRUE

OR

REPEAT-INDUCED
Celera Assembler

1. **Trim & Screen**
2. **Overlapper**
3. **Unitiger**
4. **Scaffolder**
5. **Repeat Res I, II**

Compute all overlap consistent sub-assemblies:
- **Unitigs** (*Uniquely Assembled Contig*)
Celera Assembler

Edge Types:

- **Regular Dovetail**: A → B
- **Prefix Dovetail**: A ← B
- **Suffix Dovetail**: A → B

**E.G.:** Edges are annotated with deltas of overlaps

```
  A  B  A  B  A  B
  ▶ ▶ ◀ ◀ ▶ ◀
  E.G.:

  Edges are annotated with deltas of overlaps
```
The Unitig Reduction

1. Remove “Transitively Inferrable” Overlaps:

Diagram showing connections between nodes A, B, and C.
The Unitig Reduction

2. Collapse “Unique Connector” Overlaps:
Identifying Unique DNA Stretches

Discriminator Statistic is log-odds ratio of probability unitig is unique DNA versus 2-copy DNA.
Celera Assembler

Trim & Screen → Overlapper → Unitiger → Scaffolder → Repeat Res I, II

Scaffold U-unitigs with confirmed pairs

Mated reads

[Diagram showing the workflow and interactions between the tools and the data]

Query: SCAFFOLD ILLUSTRATION
Celera Assembler

Trim & Screen

Overlapper

Unitiger

Scaffolder

Repeat Res I, II

Fill repeat gaps with doubly anchored positive unitigs

Unitig>0
Overlap Graph: Hamiltonian Approach

Each vertex represents a read from the original sequence. Vertices from repeats are connected to many others.

Find a path visiting every VERTEX exactly once: Hamiltonian path problem
Overlap Graph: Eulerian Approach

Repeat

Repeat

Repeat

Find a path visiting every EDGE exactly once:
Eulerian path problem

Placing each repeat edge together gives a clear progression of the path through the entire sequence.
Multiple Repeats

Can be easily constructed with any number of repeats
Pre-assembly

NGS ERROR CORRECTION
Ideally

...ATGTTTTT...

...ACGTATT...

...ATGTTTTT...

...ACGTTTTT...

...ATGTTTTT...

...ATGTTTTCT...
Challenges

- Unknown reference genome
- Billions of reads
- Non-uniform error distribution
- Non-uniform genome sampling
- Polymorphisms
- Repeats
Approaches

- Spectrum alignment problem:
  - Chaisson et al., 2004, 2008; Chin et al., 2009; Quake (Kelley et al., 2010); Reptile (Yang et al., 2010)

- Suffix tree:
  - SHREC (Schroder et al., 2009)
  - SHREC (Salmela and Schroder, 2010)

- Alignment based:
  - CORAL (Salmela, 2011)

- Most incorporate the base quality values
COUNTING KMERS
Counting k-mers for assembly

- **Error correction**
  - Erroneous reads will have low-frequency k-mers

- **Contamination detection**
  - Sequence from DNA contamination will be represented at a very low coverage

- **Repeat detection**
  - Very high frequency k-mers: repeat/duplication
  - Handle accordingly

- k-mers in NGS data sets can easily overwhelm memory capacity
Counting k-mers

- Given sequencing reads count how many times each k-mer occurs

- **De Bruijn graph assemblers**
  - Euler (Pevzner et al. 2001)
  - Velvet (Zerbino et al. 2008)
  - Allpaths (Butler et al. 2008)
  - ABySS (Simpson et al. 2009)
  - SOAPDenovo (Li et al. 2010)

- **Error Correction: Quake** (Kelley et al. 2010)

- **k-mer counters:** Jellyfish (Marçais et al. 2011), BFCounter (Melsted et al., 2011)
Memory usage

- Simple method
  
  Store each k-mer in a hash table with a counter

- Memory needed
  
  - store canonical k-mers
  - 2 bits for each of A,C,G,T
  - k/4 bytes per k-mer (k=31, 8 bytes)
  - 1-2 bytes per counter
  - +10% hash table overhead

- For a genome of size G, expect to see up to G distinct k-mers (2.5-3 billion for Human)

- ~ 36 Gb of memory
Number of k-mers

- This ignores the effect of sequencing errors
- 31-mers in reads aligned to Chr21
- Illumina 100x100 32-fold coverage
- Mapped 31-mers to reference
- 99.9% of unique k-mers are errors
Removing unique k-mers

Number of 31-mers

Fold coverage
Bloom filter

- Bloom filter encodes a set of k-mers
- Uses a bit array $B$ of length $m$ and $d$ hash functions
  - to insert $x$, we set $B[h_i(x)] = 1$, for $i=1,\ldots,d$
  - to query $y$, we check if $B[h_i(y)]$ all equal 1, for $i=1,\ldots,d$
- Need an estimate for $n$, the number of k-mers to insert
Bloom filter example

- a and b are inserted into a Bloom filter with m = 10, n=2, d=3
- c is not in the set, since some bits are 0
- d has not been inserted, but is still reported in the set, a false positive
- Bloom filters have no false negatives
Bloom filter

- Storing \( n \) k-mers in \( m \) bit array with \( d \) hash functions has a false positive rate of
  \[
  \approx (1 - e^{-d \frac{n}{m}})^d
  \]
- Given \( n \) and \( m \), the optimal \( d \) is \( \approx \frac{m}{n} \ln(2) \)
- Example: \( m = 8n, d=5 \) gives 2.16% fpr
  \( m = 6n, d=4 \) gives 5.6% fpr
  \( m = 4n, d=3 \) gives 14.6% fpr
- \( m=8n \), corresponds to storing 1 byte per k-mer
Algorithm

- Use a Bloom filter and a hash table

**ATGAAGTG GG**

**k-mers**
- ATGA
- TGAA
- GAAG
- AAGT
- AGTG
- GTGG
- TGGG

**AGTGGGGTGAA**

**k-mers**
- AGTG
- GTGG
- TGGG
- GGGT
- GTGA
- GTGA
- TGAA

**Bloom filter**

**Hash table**

<table>
<thead>
<tr>
<th></th>
<th>TGGG</th>
<th>AGTG</th>
<th>GTGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Pass</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Second Pass</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
This scheme guarantees

- k-mers seen twice will be in the hash table
- some unique k-mers will slip through
- second pass gives accurate counts and allows to discard false positives

Memory usage

- full for k-mers in hash table (~ 9 bytes)
- minimal for k-mers in bloom filter (~ 0.5-1 bytes)
Results whole genome

- 25-mers in 36 bp reads
- 2.37 billion distinct 25-mers in hg18
- 12.18 billion 25-mers in the sequencing data
  - 9.35 billion unique
  - 2.83 billion with coverage 2 or greater

<table>
<thead>
<tr>
<th>Program</th>
<th>Time (hrs)</th>
<th>Memory (G)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFCounter</td>
<td>23.82</td>
<td>42</td>
</tr>
<tr>
<td>Naïve</td>
<td>&gt; 26.83</td>
<td>&gt;128</td>
</tr>
</tbody>
</table>
NEXT: DE BRUIJN GRAPHS