CMSC423: Bioinformatic Algorithms, Databases and Tools

Genome assembly
Reading assignment

- http://www.cbcb.umd.edu/research/assembly_primer.shtml
- Chapter 4.5 – coverage statistics
- Chapter 8 – genome assembly
Shotgun sequencing

original DNA (hopefully)

assembly

shearing

sequencing
Overview of terms

Original DNA

fragments

sequenced ends

Assembly

Scaffolding

CMSC423 Fall 2009
Assembly Glossary

- **Read** – small (50-2000bp) segment of DNA "read" by a sequencing instrument
- **Mate-pair, paired ends** – pair of reads whose distance from each other within the genome is approximately known
- **Contig** – contiguous segment of DNA reconstructed (unambiguously) from a set of reads
- **Scaffold** – group of contigs that can be ordered and oriented with respect to each other (usually with the help of mate-pair data)
So...

- Sequencing technologies only "read" small chunks of DNA, yet genomes are substantially larger.
- The shotgun sequencing approach generates many random fragments from the original DNA.
- The task of the assembly program is to stitch together the many small pieces into a reconstruction of the genome.
- Essentially..... a huge jigsaw puzzle.
- Think: shred a collection of Harry Potter books at random then try to rebuild the original without any additional information.
Shortest common superstring problem

Given a set of strings, \( \Sigma = (s_1, \ldots, s_n) \), determine the shortest string \( S \) such that every \( s_i \) is a sub-string of \( S \).

NP-hard approximations: 4, 3, 2.89, ...

Greedy algorithm (4-approximation)

phrap, TIGR Assembler, CAP
Greedy algorithm details

Compute all pairwise overlaps
*Pick best (e.g. in terms of alignment score) overlap
Join corresponding reads
Repeat from * until no more joins possible

• How do you compute an overlap alignment?
• Hint: modify Smith-Waterman dynamic programming algorithm
Repeats (where greedy fails)

\[
\text{AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA}
\]
\[
\text{AAAAAAAA AAAA AAAA AA AAAA}
\]
\[
\text{AAAAAA AAAA AAAA AA AAAA}
\]
\[
\text{AAAAAA AAAA AAAA AA AAAA}
\]

CMSC423 Fall 2009
Impact of randomness – non-uniform coverage

Imagine raindrops on a sidewalk
Lander-Waterman statistics

L = read length  
T = minimum overlap  
G = genome size  
N = number of reads  
c = coverage (NL / G)  
σ = 1 – T/L

E(#islands) = Ne^{-cσ}  
E(island size) = L(e^{cσ} – 1) / c + 1 – σ  
contig = island with 2 or more reads

See chapter 4.5
All pairs alignment

- Needed by the assembler
- Try all pairs – must consider $\sim n^2$ pairs
- Smarter solution: only $n \times$ coverage (e.g. 8) pairs are possible
  - Build a table of k-mers contained in sequences (single pass through the genome)
  - Generate the pairs from k-mer table (single pass through k-mer table)
Additional pairwise-alignment details

- 4 types of overlaps
- Often – assume first read is “forward”
  - Normal
  - Innie
  - Outie
  - Anti-normal

- Representing the alignment

- Why not store length of overlap?
Brief aside (assembly paradigms)

- Greedy algorithm
  - easy to implement
  - relatively efficient
  - but... can make mistakes because it is greedy (only takes into account local information)

- How can you "reason" about repeats?

- Graph theory can help: 2 paradigms
  - Overlap-Layout-Consensus: nodes=reads, edges=reads overlap
  - deBruijn/repeat graph: nodes = k-mers, edges = k+1-mers (extracted from the reads).

- Both translate into: find a constrained path within a graph
Overlap-layout-consensus

Main entity: read
Relationship between reads: overlap

3 Stages: overlap (btwn reads) + layout (find placement of reads wrt each other) + consensus (multiple alignment of reads)
Paths through graphs and assembly

- Hamiltonian circuit: visit each node (city) exactly once, returning to the start
- I.e. use every read in the genome exactly once

A

B

C

D

E

F

G

H

I

Genome
Aside: graph traversals

- Hamiltonian path: visit every single node of a graph EXACTLY once (NP-hard)
- Eulerian path: visit every edge of a graph EXACTLY once (polynomial time)
- Chinese Postman: find the shortest path in a graph that visits all the edges (i.e. Eulerian path where you allow a minimum number of edges to be reused)

- Note: a Hamiltonian path or an Eulerian path are not guaranteed to exist. A Chinese postman path can always be constructed
Sequencing by hybridization

probes - all possible k-mers
Assembling SBH data

Main entity: oligomer (overlap)
Relationship between oligomers: adjacency

ACCTGATGCCAATTGCACT...

CTGAT follows CCTGA (they share 4 nucleotides: CTGA)

Problem: given all the k-mers, find the original string

In assembly: fake the SBH experiment - break the reads into k-mers
Eulerian circuit

- Eulerian circuit: visit each edge (bridge) exactly once and come back to the start
- an edge (roughly) corresponds to a read

ACCTAGATTGAGGGTC

ACCTAGATTGAGGGTC → CCTAGATTGAGGGTCG
deBruijn graph

- Nodes – set of k-mers obtained from the reads
- Edges – link k-mers that overlap by k-1 letters
  
  ACCAGTGCA
  
  CCAGTGCAT

- This formulation particularly useful for very short reads
- Solution – Eulerian path (actually Chinese postman) through the graph
- Note – multiple Eulerian paths possible (exponential number) due to repeats
deBruijn graph of *Mycoplasma genitalium*
Assembly...parting thoughts

- The basic idea of both OLC and deBruijn approaches: identify sections of DNA that MUST be present in the actual genome:
  - OLC – each read must be used because it is a piece of the original genome
  - deBruijn – each edge must be used because the DNA string corresponding to it is a piece of the original genome
Assembly... recap

- Greedy algorithm... pretty good but gets stuck at repeats
- Overlap layout consensus – equivalent to Hamiltonian path (NP-hard)
- deBruijn graph – equivalent to Eulerian path (polynomial time)
- ... BUT – exponential # of Eulerian paths consistent with reads (because of repeats)
- Ultimately... still NP-hard
Read-length vs. genome complexity

![Graph showing read-length vs. genome complexity for various organisms.](image)
In practice: graph simplifications

Thm: a graph has a unique Eulerian path if and only if its cycle graph is a tree.

Defn: cycle graph – each node is a cycle in the original graph, nodes are connected by an edge if the corresponding cycles intersect.
AMOS quick tour

- amos.sourceforge.net

Basic workflow:
- sequences are converted into the AMOS format (.afg)
- an .afg file is loaded into a flat-file database (the "bank")
- all programs interact through the bank

![Diagram showing AMOS components and workflow](image-url)
An AMOS pipeline

```bash
#!runAmos -C

#--------------------------------------- USER DEFINED VALUES ------------------#
# allow input to be either <file>.afg or just <file>
REF = $(PREFIX).1con
TGT = $(strip .afg PREFIX).afg
#------------------------------------------------------------------------------#
BINDIR   = /usr/local/bin
NUCMER   = $(shell which nucmer)
SEQS     = $(PREFIX).seq
BANK     = $(PREFIX).bank
ALIGN    = $(PREFIX).delta
LAYOUT   = $(PREFIX).layout
CONFLICT = $(PREFIX).conflict
CONTIG   = $(PREFIX).contig
FASTA   = $(PREFIX).fasta

INPUTS   = $(TGT) $(REF)
OUTPUTS  = $(CONTIG) $(FASTA)

## Building AMOS bank
10: $(BINDIR)/bank-transact -c -z -b $(BANK) -m $(TGT)

## Collecting clear range sequences
20: $(BINDIR)/dumpreads $(BANK) > $(SEQS)

## Running nucmer
30: $(NUCMER) --maxmatch --prefix=$(PREFIX) $(REF) $(SEQS)

## Running layout
40: $(BINDIR)/layout-align -U $(LAYOUT) -C $(CONFLICT) -b $(BANK) $(ALIGN)

## Running consensus
50: $(BINDIR)/make-consensus -B -b $(BANK)

## Outputting contigs
60: $(BINDIR)/bank2contig $(BANK) > $(CONTIG)

## Converting to FastA file
70: $(BINDIR)/ctg2fasta < $(CONTIG) > $(FASTA)
```
Project

• You will need to modify the Minimus pipeline to use your own overlapper program (replacing the hash-overlap command with your own)
• Part of the project is figuring out how to do this (using the AMOS documentation)
AMOS interchange format

Based on Celera message format

3-letter object tag (RED= read)

- single-line attribute (action: ADD)
- internal identifier (int32)
- external identifier

multi-line attribute
Basic flow...

- Start with an AMOS .afg file (I will provide one)
- Load it in the bank
  - `bank-transact -cf -b mybank.bnk -m myfile.afg`
- Dump the reads back out in a multi-fasta file
  - `dumpreads mybank.bnk > myfile.fa`
  - why? the IDs are now the internal IDs within the bank
- Use your program to compute overlaps (output an afg file)
  - `myoverlapper myfile.fa > myoverlaps.afg`
- Load the new overlaps in the bank
  - `bank-transact -b mybank.bnk -m myoverlaps.afg`
- Continue with standard Minimus pipeline
Overlap format

\{OVL
adj: N
rds: 159, 161
scr: 0
ahg: -32
bhg: 0
\}
\{OVL
adj: N
rds: 159, 162
scr: 0
ahg: -32
bhg: 43
\}
\{OVL
adj: l
rds: 159, 163
scr: 0
ahg: 362
bhg: 560
\}

adj – "Normal", "Innie"
rds: iid1, iid2
ahg, bhg – ahang, bhang

Note: output is "redundant" both A ovl B and B ovl A are reported