

Large-Scale Global Alignments

Multiple Alignments



Lecture 10, Thursday May 1, 2003

ARACHNE: Steps to Assemble a Genome

1. Find overlapping reads



2. Merge good pairs of reads into longer contigs



3. Link contigs to form supercontigs



4. Derive consensus sequence

..ACGATTACAATAGGTT..

Lecture 10, Thursday May 1, 2003

3. Link Contigs into Supercontigs



Normal density

Too dense:
Overcollapsed?
(Myers et al. 2000)



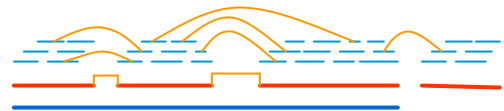
Inconsistent links:
Overcollapsed?

Lecture 10, Thursday May 1, 2003

3. Link Contigs into Supercontigs (cont'd)

Find all links between unique contigs

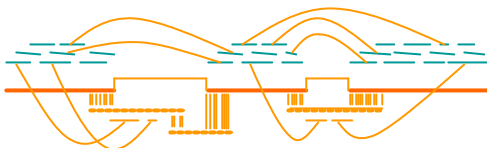
Connect contigs incrementally, if ≥ 2 links



Lecture 10, Thursday May 1, 2003

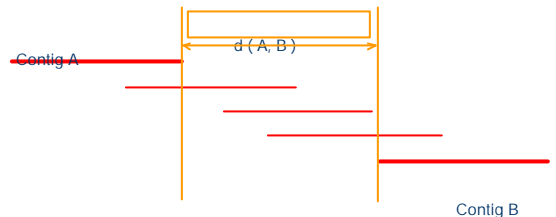
3. Link Contigs into Supercontigs (cont'd)

Fill gaps in supercontigs with paths of overcollapsed contigs



Lecture 10, Thursday May 1, 2003

3. Link Contigs into Supercontigs (cont'd)



Define $G = (V, E)$

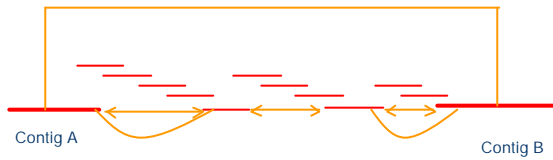
$V :=$ contigs

$E := (A, B)$ such that $d(A, B) < C$

Reason to do so: Efficiency; full shortest paths cannot be computed

Lecture 10, Thursday May 1, 2003

3. Link Contigs into Supercontigs (cont'd)



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

Lecture 10, Thursday May 1, 2003

4. Derive Consensus Sequence

```

TAGATTACACAGATTACTGA TTGATGGCGTAA CTA
TAGATTACACAGATTACTGACTTGATGGCGTAAACTA
TAG TTACACAGATTATGACTT CATGGCGTAA CTA
TAGATTACACAGATTACTGACTTGATGGCGTAA CTA
TAGATTACACAGATTACTGACTTGATGGCGTAA CTA
    
```

↓ ↓ ↓ ↓ ↓

TAGATTACACAGATTACTGACTTGATGGCGTAA CTA

Derive **multiple alignment** from pairwise read alignments

Derive each consensus base by weighted voting

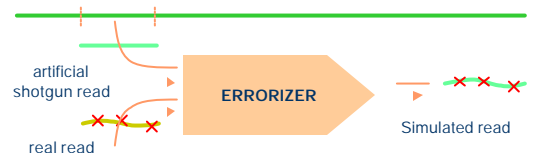
Lecture 10, Thursday May 1, 2003

Simulated Whole Genome Shotgun

- Known genomes
Flu, yeast, fly, Human chromosomes 21, 22
- Make "realistic" shotgun reads
- Run ARACHNE
- Align output with genome and compare

Lecture 10, Thursday May 1, 2003

Making a Simulated Read



Simulated reads have error patterns taken from random real reads

Lecture 10, Thursday May 1, 2003

Human 22, Results of Simulations

Plasmid/ Cosmid	10 X / 0.5 X	15 X / 0.5 X	3 X / 0 X
	100.0%	100.0%	97.5%
	1.40 Mb	10.0 Mb	0.0 Mb
	3.0%	3.0%	0.1%
	41	32	26
	97.3	91.1	67

Lecture 10, Thursday May 1, 2003

Neurospora crassa Genome (Real Data)

- 40 Mb genome, shotgun sequencing complete (WI-CGR)
- Evaluated assembly using 1.5Mb of finished BACs
- 1% uncovered (of finished BACs)

Coverage:
1705 contigs
368 supercontigs

Efficiency:
Time: 20 hr
Memory: 9 Gb

Accuracy:
< 3 misassemblies
compared with 1 Gb of
finished sequence
Errors/10⁶ letters:
Subst. 260



Mouse Genome

Improved version of ARACHNE assembled the mouse genome

Several heuristics of iteratively:

- Breaking supercontigs that are suspicious
- Rejoining supercontigs

Size of problem: 32,000,000 reads

Time: 15 days, 1 processor

Memory: 28 Gb

N50 Contigsize: 16.3 Kb \rightarrow 24.8 Kb

N50 Supercontig size: .265 Mb \rightarrow 16.9 Mb

Lecture 10, Thursday May 1, 2003

Next few lectures

More on alignments

- Large-scale global alignment – Comparing entire genomes
 - Suffix trees, sparse dynamic programming
 - MumMer, Avid, LAGAN, Shuffle-LAGAN

- Multiple alignment – Comparing proteins, many genomes
 - Scoring, Multidimensional-DP, Center-Star, Progressive alignment
 - CLUSTALW, TCOFFEE, MLAGAN

Gene recognition

- Gene recognition on a single genome
 - GENSCAN – A HMM for gene recognition

- Cross-species comparison-based gene recognition
 - TWINSKAN – A HMM
 - SLAM – A pair-HMM

Lecture 10, Thursday May 1, 2003

Rapid Global Alignments

How to align genomic sequences in (more or less!)
linear time

Motivation

- Genomic sequences are very long:
 - Human genome = 3×10^9 -long
 - Mouse genome = 2.7×10^9 -long
- Aligning genomic regions is useful for revealing common gene structure
 - Useful to compare regions $> 1,000,000$ -long

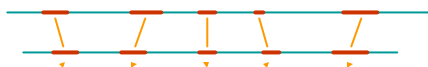
Lecture 10, Thursday May 1, 2003

Main Idea

Genomic regions of interest contain ordered islands of similarity

- E.g. genes

- Find local alignments
- Chain an optimal subset of them



Lecture 10, Thursday May 1, 2003

Outline

- Methods to **FIND** Local Alignments
 - Sorting k-long words
 - Suffix Trees
- Methods to **CHAIN** Local Alignments
 - Dynamic Programming
 - Sparse Dynamic Programming

Lecture 10, Thursday May 1, 2003

Methods to FIND Local Alignments

1. Sorting K-long words
BLAST, BLAT, and the like
2. Suffix Trees

Finding Local Alignments: Sorting k-long words

Given sequences x, y :

1. Write down all
($w, 0, i$): $w = x_{i+1} \dots x_{i+k}$
($z, 1, j$): $z = y_{j+1} \dots y_{j+k}$
2. Sort them lexicographically
3. Deduce all k-long matches between x and y
4. Extend to local alignments

Lecture 10, Thursday May 1, 2003

Sorting k-long words: example

Let x, y be matched with 3-long words:

$x = \text{caggc:}$ (cag,0,0), (agg,0,1), (ggc,0,2)
 $y = \text{ggcag:}$ (ggc,1,0), (gca,1,1), (cag,1,2)

Sorted: (agg,0,1), (cag,0,0), (cag,1,2), (ggc,0,2), (ggc,1,0), (gca,1,1)

Matches:

1. cag: $x_1x_2x_3 = y_3y_4y_5$
2. ggc: $x_3x_4x_5 = y_1y_2y_3$

Lecture 10, Thursday May 1, 2003

Running time

- Worst case: $O(NxM)$
- In practice: a large value of k results in a short list of matches

Tradeoff:

Low k : worse running time

High k : significant alignments missed

PatternHunter:

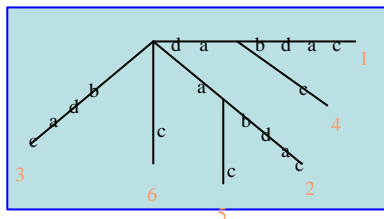
Sampling non-consecutive positions increases the likelihood to detect a conserved region, for a fixed value of k – refer to Lecture 3

Lecture 10, Thursday May 1, 2003

Suffix Trees

- Suffix trees are a method to find all maximal matches between two strings (and much more)

Example:
 $x = \text{dabdac}$



Lecture 10, Thursday May 1, 2003

Definition of a Suffix Tree

Definition:

For string $x = x_1 \dots x_m$, a **suffix tree** is:

- A rooted tree with m leaves
- Leaf: $x_1 \dots x_m$
- Each edge is a substring
- No two edges out of a node, start with same letter

It follows, every substring corresponds to an initial part of a path from root to a leaf

Lecture 10, Thursday May 1, 2003

Constructing a Suffix Tree

- Naïve algorithm: $O(N^2)$ time
- Better algorithms: $O(N)$ time
(outside the scope of this class – too technical and not so interesting)

Memory: $O(N)$ but with a sizeable constant

Lecture 10, Thursday May 1, 2003

Naïve Algorithm to Construct a Suffix Tree

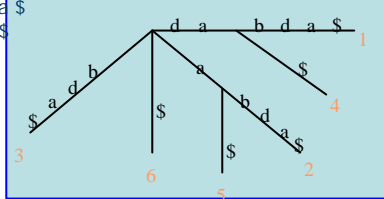
1. Initialize tree T : a single root node r
2. Insert special symbol $\$$ at end of x
3. For $j = 1$ to m
 - Find longest match of $x_i \dots x_m$ to T , starting from r
 - Split edge where match stops: new node w
 - Create edge (w, j) , and label with unmatched portion of $x_i \dots x_m$

Lecture 10, Thursday May 1, 2003

Example of Suffix Tree Construction

$x = d a b d a \$$

1. Insert $d a b d a \$$
2. Insert $a b d a \$$
3. Insert $b d a \$$
4. Insert $d a \$$
5. Insert $a \$$
6. Insert $\$$



Lecture 10, Thursday May 1, 2003

Faster Construction

Several algorithms

$O(N)$ time,

$O(N)$ memory with a big constant

Technical but not deep, outside the scope of this course

Optional: Gusfield chapter 6

Lecture 10, Thursday May 1, 2003

Memory to Store Suffix Tree

- Can store in $O(N)$ memory!
- Every edge is labeled with (i, j) :
 (i, j) denotes $x_i \dots x_j$
- Tree has $O(N)$ nodes

Proof:

1. # leaves \geq # nodes $- 1$
2. # leaves = $|x|$

Lecture 10, Thursday May 1, 2003

Application: Find all Matches Between x and y

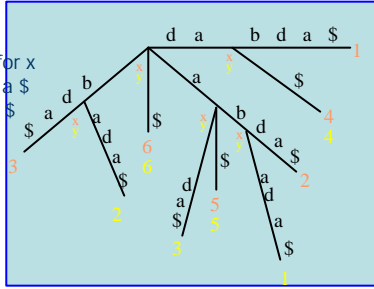
1. Build suffix tree for x , mark nodes with x
2. Insert y in suffix tree, mark all nodes y "passes from" with y
 - The path label of every node marked both 0 and 1, is a common substring

Lecture 10, Thursday May 1, 2003

Example of Suffix Tree Construction for x, y

x = d a b d a \$
y = a b a d a \$

1. Construct tree for x
2. Insert a b a d a \$
3. Insert b a d a \$
4. Insert a d a \$
5. Insert d a \$
6. Insert a \$
6. Insert \$



Lecture 10, Thursday May 1, 2003

Application: Online Search of Strings on a Database

Say a database $D = \{s_1, s_2, \dots, s_n\}$
(eg. proteins)

Question: given new string x, find all matches of x to database

1. Build suffix tree for $\{s_1, \dots, s_n\}$
2. All new queries x take $O(|x|)$ time (somewhat like BLAST)

Lecture 10, Thursday May 1, 2003

Application: Common Substrings of k Strings

- Say we want to find the longest common substring of s_1, s_2, \dots, s_n
1. Build suffix tree for s_1, \dots, s_n
 2. All nodes labeled $\{s_{i_1} \dots s_{i_k}\}$ represent a match between s_{i_1}, \dots, s_{i_k}

Lecture 10, Thursday May 1, 2003