Whole genome alignment
Applications of genome alignment

• Comparing different genome assemblies
• Locating genome duplications and conserved segments
• Gene finding through comparative genomics
• Analyzing pathogenic bacteria against their harmless close relatives
Homology map

We multiply align these blocks together
Overview/Goals

• Input:
  – Set of whole genomes, which may differ by substitutions, indels and rearrangements
  – Uses open reading frames or other gene predictions

• Output:
  – One alignment per region of genomes that has not been “shuffled”
    • Two genomes = global
    • > 2 genomes = multiple
Two different most parsimonious scenarios that transform the order of the 11 synteny blocks on the mouse X chromosome into the order on the human X chromosome.

Pevzner P., Tesler G. PNAS 2003;100:7672-7677
Whole-genome alignment

• Advanced data structures can also be used to efficiently speed up genomic alignments of closely-related organisms.

• We will introduce suffix trees and the MUMmer algorithm before going into detail next week.
Suffix trees

• Specialized form of keyword trees/tries

• Key idea:
  – preprocess text T, not pattern P
    • \( O(m) \) preprocess time
    • \( O(n+k) \) search time
      – \( k \) is number of occurrences of P in T
Keyword Tree

• $P = \{\text{poet, pope, popo, too}\}$
Suffix Tree

• Take any $m$ character string $S$ like xabxac
• Set of keywords is the set of suffixes of $S$
  – $\{xabxac, abxac, bxac, xac, ac, c\}$

• Changes relative to keyword trees:
  – Assumption: no suffix is a prefix of another suffix (can be a substring, but not a prefix)
    • Assure this by adding a character $\$\$ to end of $S$
  – Internal nodes except root must have at least 2 children
Example suffix tree

• \{xabxac, abxac, bxac, xac, ac, c\}
Notation to keep track of

• Label of a path from root r to a node v is simply the concatenation of labels on edges from r to v
• label of a node v is L(v)
  – path label from r to v
• string-depth of v
  – number of characters in v’s label L(v)
Using suffix trees in exact matching

- Build suffix tree for text $T$
- Match pattern $P$ against tree starting at root until
  - Case 1: $P$ is completely matched
    - Every leaf below this match point is the starting location of $P$ in $T$
  - Case 2: No match is possible
    - $P$ does not occur in $T$
• $T = xabxac$
  – suffixes = $\{xabxac, abxac, bxac, xac, ac, c\}$
• Pattern $P_1$: xa
• Pattern $P_2$: xb
In-class example

- \( S = xabxabdeabhixab\$
- \( xabxacdefghixab\$
- \( abxacdefghixab\$
- \( bxacdefghixab\$
- \( xacdefghixab\$
- \( \ldots \$
- \( \$

Building trees: $O(m^2)$ algorithm

- Initialize
  - One edge for the entire string $S[1..m]$.

- For $i = 2$ to $m$
  - Add suffix $S[i..m]$ to suffix tree
    - Find match point for string $S[i..m]$ in current tree
    - If in “middle” of edge, create new node $w$
    - Add remainder of $S[i..m]$ as edge label to suffix $i$ leaf

- Running Time
  - $O(m-i)$ time to add suffix $S[i..m]$
Running Time Analysis

• Build suffix tree:
  – Will show this is $O(m)$
  – This is preprocessing

• Search time:
  – $O(n+k)$ where $k$ is the number of occurrences of $P$ in $T$
  – $O(n)$ to find match point if it exists
  – $O(k)$ to find all leaves below match point
Why suffix trees are important in genome alignment

• Long unique matches have a high probability of being included in the final genomic alignment.

• We need to set the minimum length high-enough, however, to avoid random noise.
  – MUMs = maximal unique matches
  – MEMs = maximal exact matches
Overview

Genome A

<table>
<thead>
<tr>
<th>1</th>
<th>4</th>
<th>3</th>
<th>5</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Genome A’

We have 5 matches that can not be extended to left or right

We have 4 gaps to fill between these matches
MUM-based alignments

- MUMs are by definition unique maximal matches in both sequences
  - Originally required building a generalized suffix tree of both genomes
  - Internal nodes w/ only two leaves, one from each input, are unique and not right-extensible
  - Check for left-extensibility, then go!
Maximal Unique Matches

WINDOW$  INDIGO$
1234567  1234567

Left-extensible By “I”
Early whole genome alignment algorithms

• Arranged MUMs relative to one genome using Longest Increasing Subsequence (LIS) algorithm

• Filled in small gaps using dynamic programming
  – Space inefficient for large gaps
Banded Dynamic Programming

- Compute only lower and upper rectangles based on desired percent similarity
Suffix links are in green

From Delcher et al., 2002, Nucleic Acids Res 30(11):2478-83
Applications

• Comparing different genome assemblies
• Locating genome duplications and conserved segments
• Gene finding through comparative genomics
• Analyzing pathogenic bacteria against their harmless close relatives
BLASTZ

• Modification of BLAST for whole-genome alignment of close species (i.e. human-mouse)

• Optimized for intron-exon discovery.

• Two differences with gapped BLAST:
  – Matching regions can be restricted to occur in same order and orientation.
  – Uses a special scoring matrix that limits false positive alignments in low complexity regions.
Optimization

- Two changes to BLASTZ significantly improved its execution speed.

- If the software realizes that many regions of the mouse genome align to the same human segment, that segment is marked so that it will be ignored in later steps.

- Second, the idea of Ma et al. (2002) where for runs of 19 consecutive within which the 12 positions indicated by a 1 in the string 1110100110010101111 are identical.
Results

- **Data:**
  - human genome into ~3000 segments (1 MB each)
  - Divided mouse genome into 100 30MB segments

- **Run time:**
  - 481 CPU days
  - 0.5 days on a 1,024 processor cluster
  - 20 GB of output
MUMmer 2.0

• Improved space implementation of suffix tree using a few tricks (17 bytes/base)

• Introduced banded dynamic programming and advanced clustering to tackle larger gaps

• Used suffix tree “streaming” of multiple queries against a reference
MUMmer 2

- Three times faster
- One-third memory usage
- Support protein sequence and multiple sequences.
- Entire human chromosomes

- Can align millions of nucleotides in a few minutes on a desktop computer.
Linear time of suffix arrays

• There were three papers in 2002 that solved the old problem of constructing suffix arrays in linear time.

• These were:
  – Ko and Aluru – very interesting, but hard to understand
  – Kim et al. – was based on older parallel suffix tree algorithms
  – Karakkanen and Sanders is the simplest and most elegant.
FIGURE 1.1: Suffix tree, suffix array and Lcp array of the string mississippi. The suffix links in the tree are given by $x \rightarrow z \rightarrow y \rightarrow u \rightarrow r, v \rightarrow r,$ and $w \rightarrow r.$