

Algorithms in Bioinformatics:

Lecture 10-11: Genome Alignment

Lucia Moura

Fall 2011

Genome Alignment

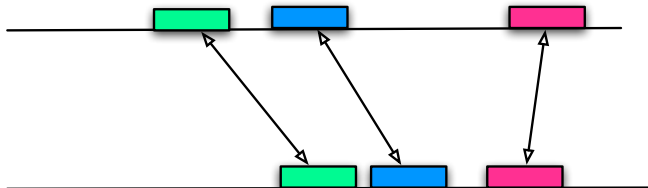
- Complete genomes have now been sequenced. Biologists want to know the similarities and differences between two or more related organisms.
- This leads to whole genome alignment problem.
- Gene or protein comparisons (small DNA sequences) can be done using algorithms for sequence alignment such as Smith-Waterman, Needleman-Wunsch (seen in chapter 2/sequence similarity)
- Special tools are needed for whole genome, since $O(nm)$ time complexity is prohibitive.
- We will study two methods: MUMmer and Mutation Sensitive Alignment.

Genome Alignment Methods

Phases:

- ① Identify potential anchors:
 - ▶ short regions between two genomes which are highly similar.
 - ▶ possible conserved regions between two genomes.

Example: Maximal Unique Matches (MUM) can be used as anchors.
- ② Identify a set of co-linear anchors:
 - ▶ these are likely to be a conserved region.
- ③ Close the gaps between the anchors to get the final alignment.

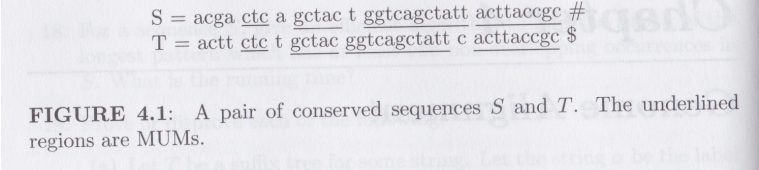


Maximal Unique Match (MUM)

Definition

Given two genomes A and B , a Maximal Unique Match (MUM) substring is a common substring of A and B of length longer than a specified minimum length d such that

- it is maximal; that is, it cannot be extended on either end without causing a mismatch;
- it is unique in both sequences.



S = acga ctc a gctac t ggtcagctatt acttacgc #
T = actt ctc t gctac ggtcagctatt c acttacgc \$

FIGURE 4.1: A pair of conserved sequences S and T . The underlined regions are MUMs.

How to find MUMs

Algorithm Suffix-tree-MUM

Require: Two genome sequences $S[1..m_1]$ and $T[1..m_2]$, a threshold d

Ensure: The set of MUMs of S and T

- 1: Build a generalized suffix tree for S and T .
- 2: Mark all the internal nodes that have exactly two leaf children, which represent both suffixes of S and T .
- 3: For each marked node of depth at least d , suppose it represents the i -th suffix S_i of S and the j -th suffix T_j of T . We check whether $S[i-1] = T[j-1]$. If not, the path label of this marked node is a MUM.

FIGURE 4.4: A suffix tree-based algorithm for computing the MUMs of two genomes S and T .

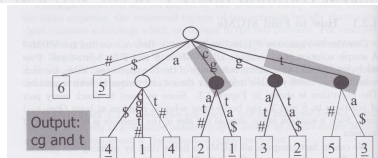


FIGURE 4.5: Consider two genomes $S = acgat\#$ and $T = cgtas$. Step 1 constructs the generalized suffix tree for genomes S and T . Step 2 identifies internal nodes with exactly one descendant leaf representing S and exactly one descendant leaf representing T . Those nodes are filled with solid color. Step 3 identifies MUMs, which correspond to those highlighted paths.

MUMs and conserved regions

Note that MUMs can cover conserved regions, but a lot of the MUMs may be noise. Phase 2 of genome alignment tries to filter out the noisy MUMs.

Mouse Chr.	Human Chr.	# of Published Gene Pairs	# of MUMs
2	15	51	96,473
7	19	192	52,394
14	3	23	58,708
14	8	38	38,818
15	12	80	88,305
15	22	72	71,613
16	16	31	66,536
16	21	64	51,009
16	22	30	61,200
17	6	150	94,095
17	16	46	29,001
17	19	30	56,536
18	5	64	131,850
19	9	22	62,296
19	11	93	29,814

FIGURE 4.6: Mice and humans share a lot of gene pairs. The set of known conserved genes was obtained from *GenBank* [217]. The number of MUMs was computed using the method in Section 4.2.1.

MUMmer1 (LCS)

Here we present the first version of the MUMmer method:

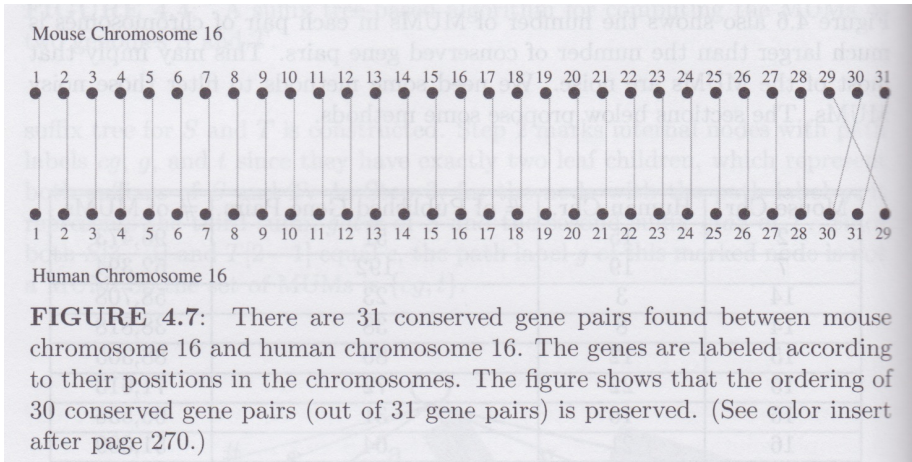
DELCHER, KASIF, FLEISCHMANN, PETERSON, WHITE, SALSBERG,
Alignment of two genomes, *Nucleid Acid Research* 27, 2369–2376, 1999.

Method MUMmer1 to align $S[1..m_1]$ and $T[1..m_2]$

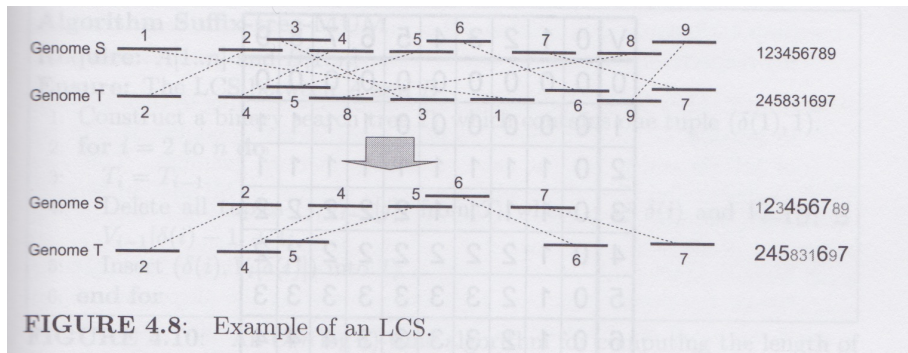
- Step 1: Compute all MUMs between sequences S and T .
This can be done in $O(m_1 + m_2)$.
- Step 2: Label MUMs according to position in S and find Longest Common Subsequence (LCS) of MUMs in S and T .
This can be done in $O(n \log n)$, where n is the number of MUMs;
 $n \ll m_1, m_2$.
- Step 3: Complete the alignment. Employ one of several algorithms for closing the gaps/interruptions in the MUM alignment.
Time complexity depends on the algorithm used.

Step 2- Longest Common Subsequence problem

Motivation: “It has been observed that two closely related species preserve the ordering of most of the conserved genes.” (p 92)



Step 2- Longest Common Subsequence (LCS) Algorithms



Two algorithms for finding LCS of two permutations (since MUMs are unique):

- dynamic programming: $O(n^2)$
- sparsified dynamic programming using binary search trees: $O(n \log n)$

LCS of permutations by dynamic programming

Let $A[1..n]$ and $B[1..n]$ be two strings which are permutations of $\{1, 2, \dots, n\}$.

Define $\delta(i)$ to be the index of B such that $A[i] = B[\delta(i)]$.

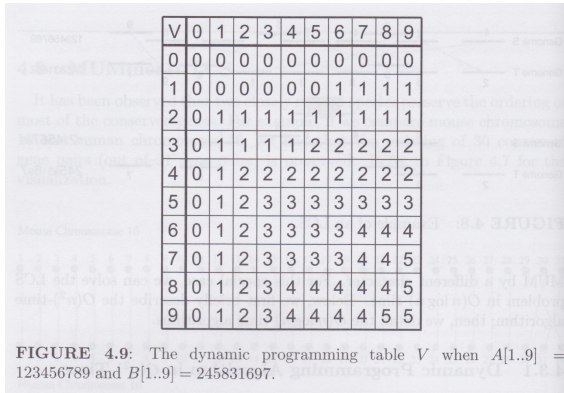
Let $V_i[j]$ be the length of the longest common subsequence of $A[1..i]$ and $B[1..j]$.

$$V_i[0] = 0$$

$$V_0[j] = 0$$

$$\begin{aligned} V_i[j] &= V_{i-1}[j], && \text{if } j < \delta(i) \\ &= \max\{V_{i-1}[j], V_{i-1}[\delta(i) - 1] + 1\}, && \text{if } j \geq \delta(i) \end{aligned}$$

What is special about this dynamic programming table?



Note that $V_i[0] \leq V_i[1] \leq \dots \leq V_i[n]$.

We can store $V_7[0..9] = 0123333445$ more compactly as:

$(1, 1), (2, 2), (3, 3), (7, 4), (9, 5)$, where (j, v) corresponds to an index j where there is a change to value v in the row.

LCS of permutations by $O(n \log n)$ sparsified dynamic programming

The idea is to update the rows of the matrix, given that each row of the matrix is stored in an optimized way.

Lemma

Let j' be the smallest integer greater than $\delta(i)$ such that $V_{i-1}[\delta(i) - 1] + 1 < V_{i-1}[j']$. So, for $1 \leq j \leq n$, we have

$$\begin{aligned} V_i[j] &= V_{i-1}[\delta(i) - 1] + 1, & \text{if } \delta(i) \leq j \leq j' - 1 \\ &= V_{i-1}[j], & \text{otherwise} \end{aligned}$$

This suggests the following update to create row V_i from V_{i-1} :

- 1 Delete all tuples $(j, V_{i-1}[j])$ in V_{i-1} where $j \geq \delta(i)$ and $V_{i-1}[j] \leq V_{i-1}[\delta(i) - 1] + 1$.
- 2 Insert $(\delta(i), V_{i-1}[\delta(i) - 1] + 1)$.

Algorithm

Algorithm ~~Suffix tree-MUM~~ LCS

Require: $A[1..n]$ and $B[1..n]$

Ensure: The LCS between A and B

- 1: Construct a binary search tree T_1 which contains one tuple $(\delta(1), 1)$.
- 2: **for** $i = 2$ to n **do**
- 3: $T_i = T_{i-1}$
- 4: Delete all tuples $(j, V_{i-1}[j])$ from T_i where $j \geq \delta(i)$ and $V_{i-1}[j] \leq V_{i-1}[\delta(i) - 1] + 1$;
- 5: Insert $(\delta(i), V_i[\delta(i)])$ into T_i .
- 6: **end for** $(\delta(i), V_{i-1}[\delta(i) - 1] + 1)$

FIGURE 4.10: An $O(n \log n)$ -time algorithm for computing the length of the longest common subsequence (LCS) between A and B .

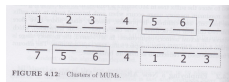
Analysis: The tuples are kept in a balanced binary search tree, so that insertions, deletions and searches take $O(\log n)$ time. At each iteration i we delete α_i tuples and insert one. Thus we can construct V_n in $O((\alpha_1 + \alpha_2 + \dots + \alpha_n + n) \log n)$. Since we insert n -tuples and each tuple can be deleted at most once, we have $\alpha_1 + \alpha_2 + \dots + \alpha_n \leq n$.

So the running time is $O(n \log n)$.

MUMmer2 and MUMmer3

These new improvements appeared in Delcher, Phillippy, Carlton, Salzberg (2002) and Kurtz, Phillippy, Delcher, Smoot, Shumway, Antonescu, Salzberg (2004).

- Reducing memory usage: reduction in suffix tree memory usage (2: 20 bytes/base; 3: 16 bytes/base).
- Alternative algorithm for finding MUMs: only the reference genome is stored in the suffix tree and a second is used as query. This reduces memory requirement and reduces time for using several query sequences.



- Clustering MUMs: [\[improve coverage\]](#)

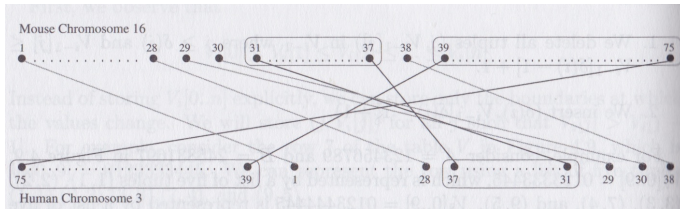
MUMmer2 has a module that takes into account possible rearrangements between genomes. The system outputs a series of separate, independent alignment regions, corresponding to clusters of MUMs.

- Extending the definition of MUMs: [\[improve coverage\]](#)

MUMmer3 allows 3 variations of MUMs (unique in both strings, unique in neither string, unique in the reference string only).

Mutation Sensitive Alignment (MSA)

Main motivation:



When genomes are closely related, they can be transformed via a few reversals or transpositions.

MSA looks for subsequences of the two MUM sequences that differ by at most k reversals, transpositions, or reversed transpositions.

This algorithm appeared in:

H. L. CHAN, T. W. LAM, W. K. SUNG, PRUDENCE W. H. WONG, S. M. YIU AND X. FAN, The mutated subsequence problem and locating conserved genes, *Bioinformatics* **21** (2005): 2271–2278.

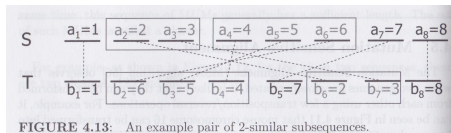
Maximum weight common subsequence (MWCS) of MUMs

- Let A and B be permutations of the n MUMs on two genomes.
- Each MUM is assigned a weight, normally equal to its length.
- The weight of a common subsequence of A and B is the sum of the weights of the MUMs that form the subsequence.
- The maximum weight common subsequence (MWCS) is a common subsequence of A and B with maximum weight.

k -similar subsequence problem

Definition

Let k be a non-negative integer. Let X be a subsequence of A and Y be a subsequence of B . (X, Y) is said to be a *pair of k -similar subsequences* if X can be transformed into Y by performing k transposition/reversal/transposed-reversal operations on k disjoint sub-sequences in X .



The k -similar subsequence problem: find a k -similar subsequence of A and B which maximizes the total weight.

Chan et al. 2005 say the problem is believed to be NP-complete (the textbook says it is NP-complete, but the reference is incorrect).

Idea of the Heuristic Algorithm for k -similar subsequence problem

- Step 1: Find the MWCS between A and B (the backbone).
Adapt algorithm for LCS (page 13 of this notes) to account for weights. This takes $O(n \log n)$ time.
- Step 2: For every interval $A[i..j]$ compute the score $Score(i, j)$ of inserting such a subsequence into the backbone.
- Step 3: Find k intervals $A[i_1..j_1], \dots, A[i_k..j_k]$ that are mutually disjoint and maximize the total score.
- Step 4: Refine the k intervals so that $B[\delta(i_1).. \delta(j_1)], \dots, B[\delta(i_k).. \delta(j_k)]$ are also mutually disjoint.

Step 2 - For every interval $A[i..j]$ compute the score $Score(i, j)$ of inserting such a subsequence into the backbone:

$Score(i, j)$ = weight of the MWCS($A[i..j]$, $B[\delta(i)..\delta(j)]$)

– total weight of characters in the backbone that fall into $A[i..j]$ or $B[\delta(i)..\delta(j)]$

- This step is dominated by computing each MWCS for all $1 \leq i < j \leq n$.
- Doing $O(n^2)$ MWCSs, each taking $O(n \log n)$ would take $O(n^3 \log n)$.
- This can be done in $O(n^2 \log n)$ using the following algorithm.

```
1) for  $i = 1$  to  $n$  do
2)   Find the MWCS for  $A[i..n]$  and  $B[\delta(i)..\delta(n)]$  in  $O(n \log n)$  time.
3)   Find the MWCS for  $A[i..n]$  and reverse of  $B[1..\delta(i)]$  in  $O(n \log n)$  time.
4)   Retrieve MWCS( $A[i..j]$ ,  $B[\delta(i)..\delta(j)]$ ) in  $O(\log n)$  time for every  $j \geq i$ 
5) end for
```

Step 3 - Find k intervals $A[i_1..j_1], \dots, A[i_k..j_k]$ that are mutually disjoint and maximize the total score.

Solve it by dynamic programming.

For $0 \leq c \leq k$ and $j \leq n$, let $Opt(c, j) =$ maximum sum of scores of c intervals in $A[1..j]$.

$$Opt(c, j) = \max \left\{ \begin{array}{l} Opt(c, j-1), \\ \max_{1 \leq i \leq j} [Opt(c-1, i-1) + Score(i, j)] \end{array} \right\}$$

This can be computed using a table with kn entries, each computed in $O(n)$. So the running time of step 3 is $O(kn^2)$.

Step 4 - Refine the k intervals so that $B[\delta(i_1).. \delta(j_1)], \dots, B[\delta(i_k).. \delta(j_k)]$ are also mutually disjoint.

This is done via a greedy heuristic:

while there are no overlapping intervals:

- Pick arbitrary overlapping intervals $\delta(i).. \delta(j)$ and $\delta(i').. \delta(j')$.
- Consider all possible ways of reducing $i..j$ and $i'..j'$ so that above intervals do not overlap and score is maximized.

The original paper claims to do this step in $O(kn^2)$, while the textbook says $O(k^2)$.

All steps together yield running time $O(n^2(\log n + k))$.

Summary of MSA

Given two genomes $S[1..m_1]$ and $T[1..m_2]$ and a non-negative k , the MSA algorithm gives an approximate/heuristic solution to the maximum weight k -similar subsequence problem.

- 1 Find all MUMs of S and T , and let A and B be the ordering of the MUMs on the two genomes. [time $O(m_1 + m_2)$]
- 2 Run the heuristic algorithm to approximate the maximum weight k -similar subsequences of A and B . [time $O(n^2(\log n + k))$]

Total running time: $O(m_1 + m_2 + n^2(\log n + k))$.

Experimental results

Exp. No.	Coverage		Preciseness	
	MUMmer	MSA	MUMmer	MSA
1	76.50%	92.20%	21.70%	22.70%
2	71.40%	91.70%	21.30%	25.10%
3	87.00%	100.00%	24.80%	25.50%
4	76.30%	94.70%	27.40%	26.70%
5	92.50%	96.30%	32.50%	32.00%
6	72.20%	95.80%	31.20%	32.90%
7	67.70%	87.10%	13.50%	17.80%
8	78.10%	90.60%	37.20%	36.70%
9	80.00%	86.70%	40.70%	49.70%
10	82.00%	92.00%	30.90%	32.10%
11	65.20%	89.10%	30.50%	36.00%
12	60.00%	80.00%	27.50%	41.90%
13	89.10%	95.30%	18.20%	18.40%
14	72.70%	86.40%	10.40%	12.60%
15	78.50%	91.40%	30.00%	29.70%
average	76.60%	91.30%	26.50%	29.30%

FIGURE 4.15: Performance comparison between MUMmer3 and MSA.

human/mouse genome:

coverage=percentage of published genes discovered - is good

preciseness= percentage of results that match some published gene - only 30%!

Two possibilities: lots of conserved regions that are not genes, or lots of unknown genes.

Exp. No.	Coverage		Preciseness	
	MUMmer	MSA	MUMmer	MSA
1	100%	100%	44%	91%
2	58%	80%	80%	85%
3	58%	69%	64%	80%
4	83%	95%	91%	94%
5	61%	68%	70%	85%
6	59%	73%	78%	87%
7	58%	69%	47%	79%
8	83%	94%	86%	94%
9	63%	69%	65%	86%
10	75%	85%	75%	85%
11	53%	68%	77%	83%
12	75%	87%	71%	93%
13	60%	67%	52%	89%
14	74%	75%	65%	90%
15	52%	63%	66%	82%
16	61%	85%	83%	89%
17	58%	74%	83%	84%
18	61%	76%	76%	86%
average	66%	78%	71%	87%

FIGURE 4.17: Experimental result on Baculoviridae.

Genome Alignment Summary

We have seen MUMmer algorithms and MSA algorithm.

For other methods for genome alignment, see survey:

P. CHAIN, S. KURTZ, E. OHLEBUSCH, T. SLEZAK,
An Applications-focused Review of Comparative Genomics Tools:
Capabilities, Limitations and Future Challenges, *Briefings in Bioinformatics*
4(2): 105-123 (2003)