

Combinatorial Methods in Bioinformatics

Finding Genes by Whole Genome Alignment

Wing-Kin Sung, Ken 宋永健

ksung@comp.nus.edu.sg

Mouse vs Human

- Mouse and human are closely related species. They share a lot of gene pairs.

Mouse Chr No.	Human Chr No.	# of Published Gene Pairs
2	15	51
7	19	192
14	3	23
14	8	38
15	12	80
15	22	72
16	16	31
16	21	64
16	22	30
17	6	150
17	16	46
17	19	30
18	5	64
19	9	22
19	11	93

Our aim

- Suppose we are given human and mouse genomes!
- Is it possible to extract all conserved gene pairs between human and mouse?
- One possible solution --- **Whole genome alignment!**

Agenda

	Coverage	Preciseness
MUM	≈100%	Many false positives
LCS (MUMmer1)	Very less	Not many false positives
Clustering (MUMmer3)	76.6%	26.5%
Mutation-Sensitive Alignment	91.3%	29.3%

Observation 1

- Though a pair of conserved genes rarely have exactly the same sequence, they share a lot of common substrings and some of them are indeed unique to this pair of genes!
- For example,

Genome1: ...ACGA**CTCA**GCTACTGGTCAGCTATTACTT...

Genome2: ...ACTT**CTCTGCTAC**GGTCAGCTATTCCCGC...

Solution!

- For a pair of genomes, we try to find all common substrings which are unique!
- Such kind of common substring is known as **Maximal Unique Match (MUM)**.
- For almost every conserved gene pair, there exists a MUM which is unique to them.

Formal definition of MUM

- MUM stands for **Maximal Unique Match**
- It is a common substring of the two genomes such that
 1. The common substring is unique in both genomes
 2. The common substring is maximal, that is, the substring cannot be extended on either end without incurring a mismatch
- We usually aim to find MUM of length at least d (d is a user defined threshold).

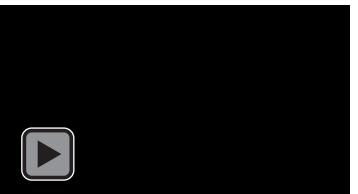
In our experiment for aligning human and mouse, we set $d=20$.

Example of finding MUMs

- Consider $S=a\text{c}\text{g}\text{a}\text{t}\#$, $T=\text{c}\text{g}\text{t}\text{a}\$$
- There are two MUMs: cg and t

Question

- S=acgactcagctactggtcagctattacttaccgc#
- T=acttctctgctacggtcagctattacttaccgc\$
- Can you find all MUMs between S and T?



Answer

- S=acgactcagctactggtcagctattacttaccgc#
- T=acttctctgctacggtcagctattacttaccgc\$
- There are four MUMs: ctc, gctac, ggtcagctatt, acttaccgc.

How to find MUMs?

- Brute-force approach:

Input: Two genome sequences $S[1..m_1]$ and $T[1..m_2]$

- For every position i in S
 - For every position j in T
 - Find the longest common prefix P of $S[i..m_1]$ and $T[j..m_2]$
 - Check whether $|P| \geq d$ and whether P is unique in both genomes. If yes, report it as a MUM.

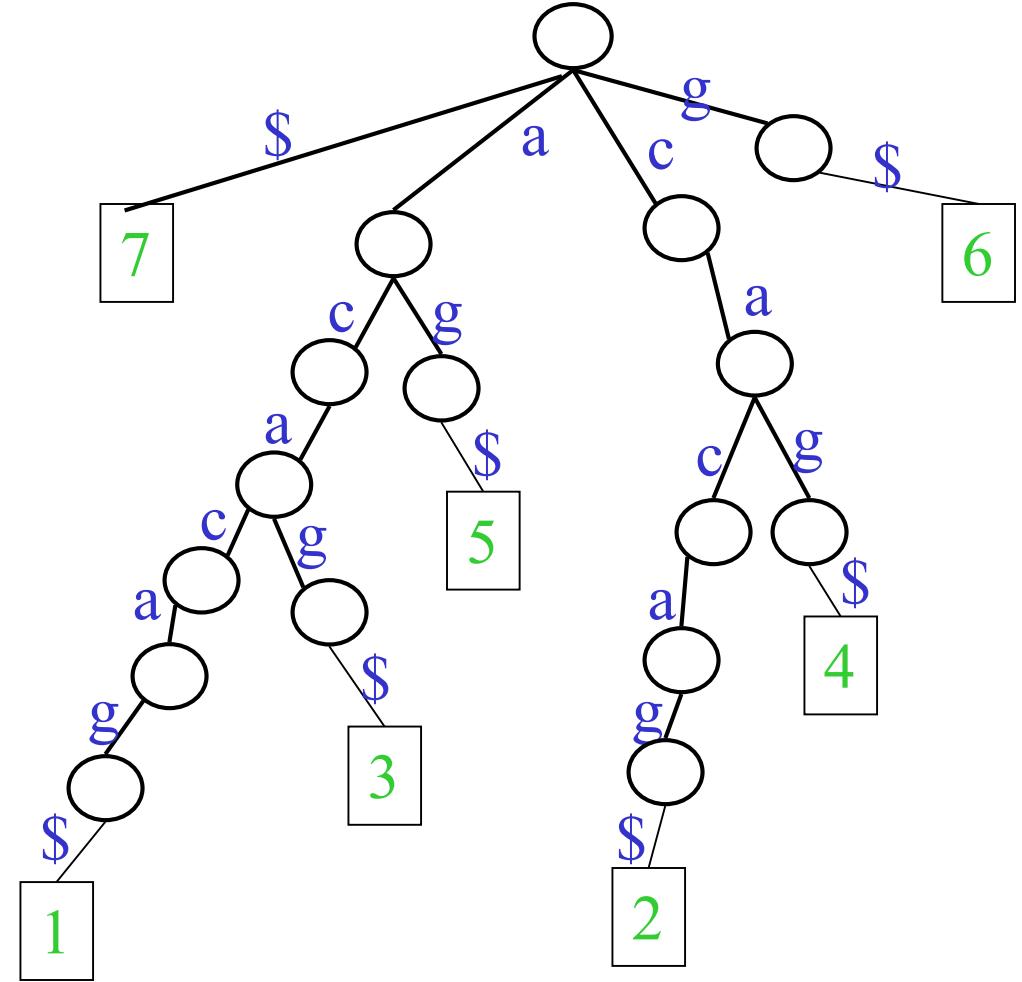
- This solution requires at least $O(m_1 m_2)$ time. **Too slow!**

Suffix Trie

E.g. consider the string $S = \text{acacag\$}$

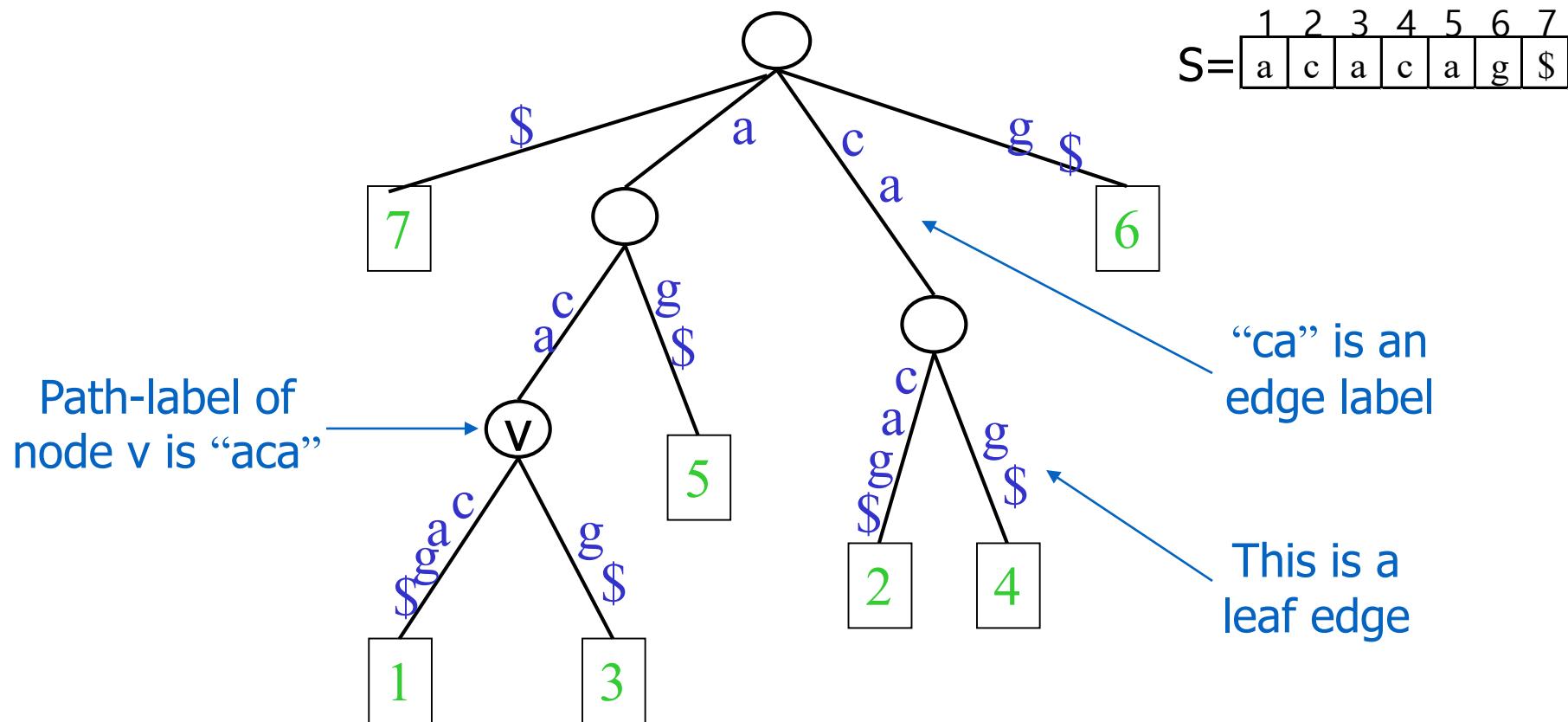
Suffix Trie: a tries of
all possible suffixes of S

	Suffix
1	acacag\$
2	cacag\$
3	acag\$
4	cag\$
5	ag\$
6	g\$
7	\$



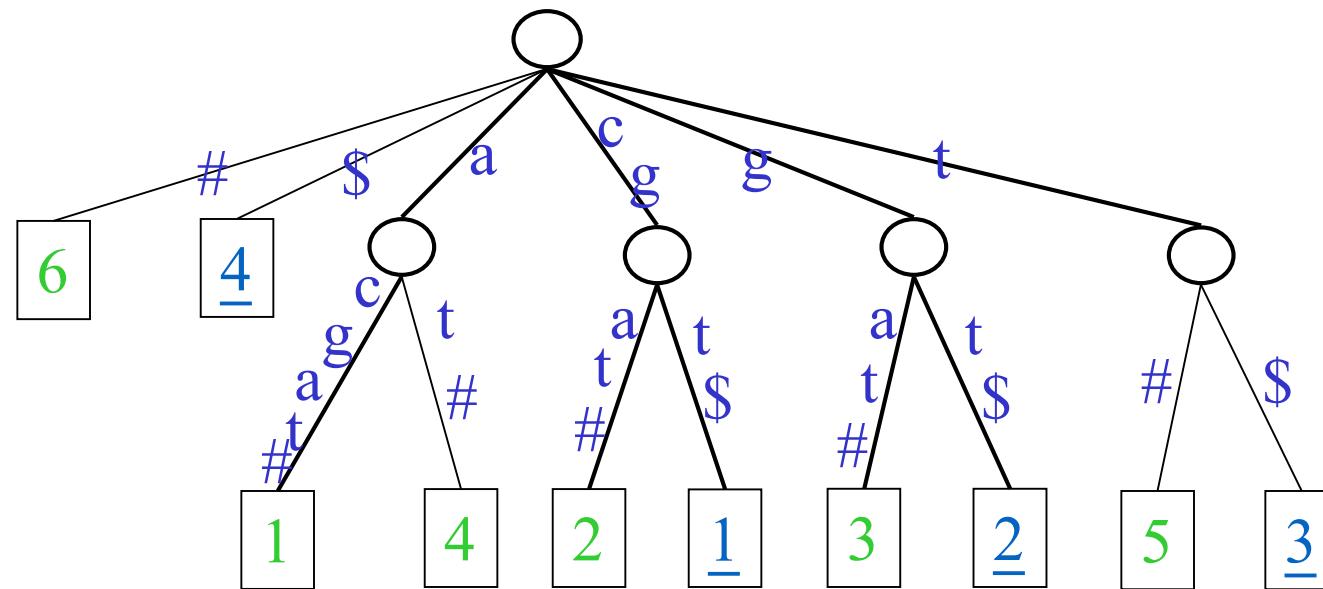
Suffix Tree

Suffix tree for $S=acacag\$$: merge nodes with only one child



Generalized suffix tree

- Build a suffix tree for two or more strings
- E.g. $S_1 = \text{acgat}\#$, $S_2 = \text{cgt}\$$



Suffix tree can be built in linear time

- Let n be the length of the text S with alphabet Σ . The suffix tree can be constructed in $O(n)$ time.
- **Weiner's algorithm** [IEEE Symposium on Switching and Automata Theory 1973]
 - Knuth called it the algorithm of 1973
 - $O(n)$ time and $O(n|\Sigma|)$ word space or $O(n \log |\Sigma|)$ time and $O(n)$ word space
 - Insert suffixes into the tree from right to left
- **McGreight's algorithm** [JACM 1976]
 - $O(n)$ time and $O(n|\Sigma|)$ word space or $O(n \log |\Sigma|)$ time and $O(n)$ word space
 - use 25% less space than Weiner's algorithm
- **Ukkonen's algorithm** [Algorithmica 1995]
 - $O(n)$ time and $O(n|\Sigma|)$ word space or $O(n \log |\Sigma|)$ time and $O(n)$ word space
 - It is an online algorithm (that is, as the string is getting longer, we can incrementally update the suffix tree)
- **Farach's algorithm** [FOCS 1997]
 - $O(n)$ time and $O(n)$ word space, independent of alphabet size
- **Hon, Sadakane, and Sung's algorithm** [FOCS 2003]
 - $O(n)$ bit space $O(n \log^e n)$ time for $0 < e < 1$
 - $O(n)$ bit space $O(n)$ time for suffix array construction

Generalized suffix tree can be built in linear time

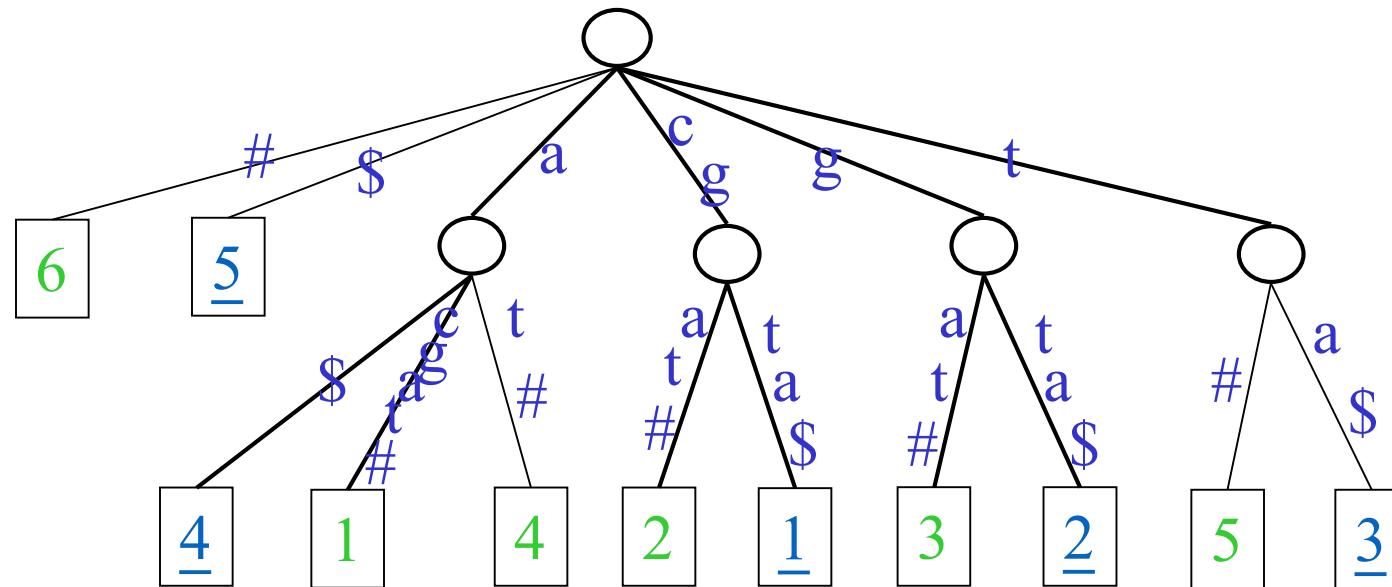
- Let the total length of S_1, S_2, \dots, S_k be m .
- The generalized suffix tree for S_1, S_2, \dots, S_k can be built in $O(m)$ time.

Finding MUMs by suffix tree!

- MUMs can be found in $O(m_1+m_2)$ time by suffix tree!
 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, WLOG, 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 an MUM.

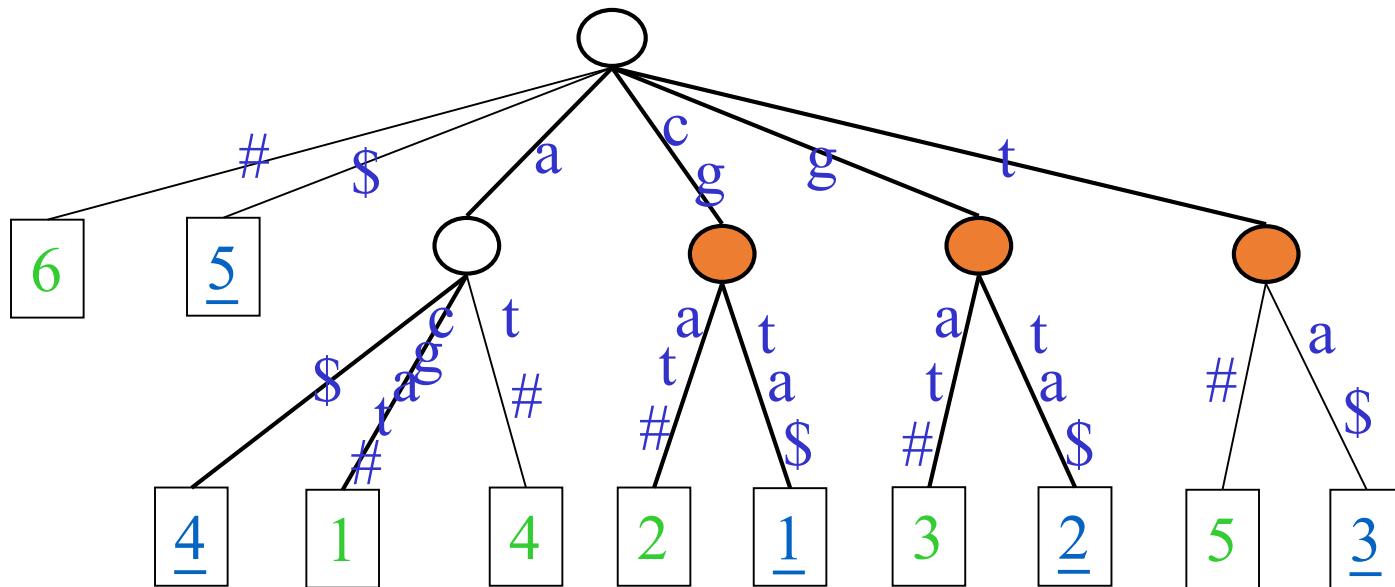
Example of finding MUMs (I)

- Consider $S=acgat\#$, $T=cgtat\$$
- Step 1: Build the generalized suffix tree for S and T .



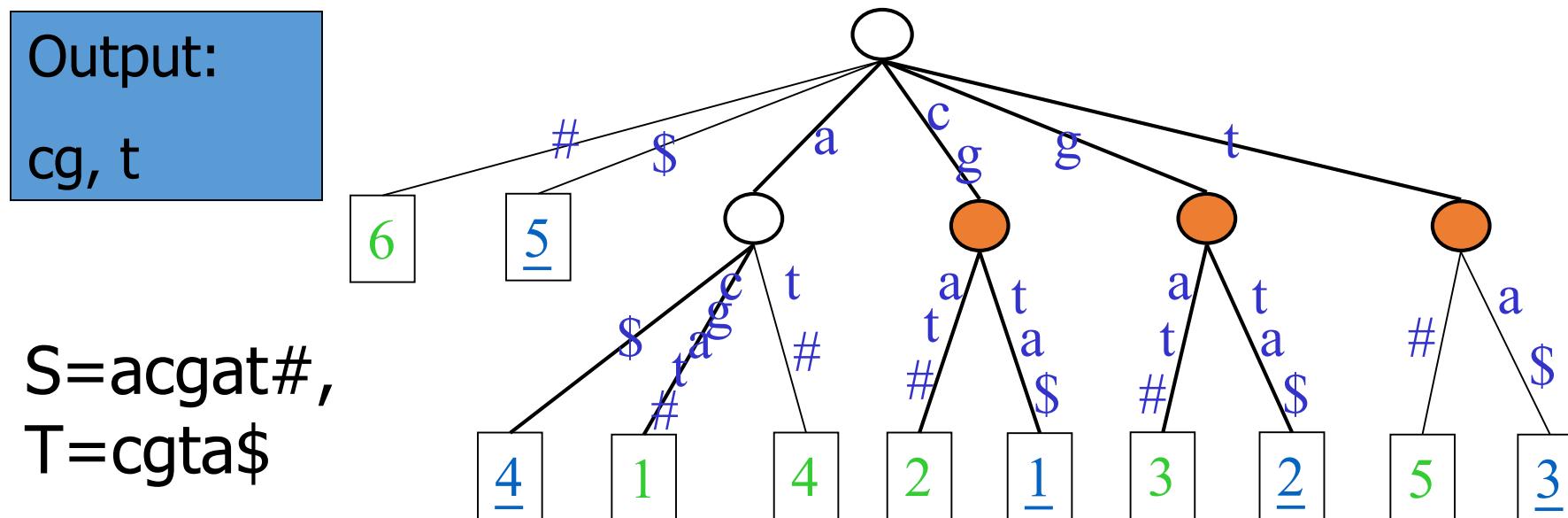
Example of finding MUMs (II)

- Step 2: Mark all the internal nodes that have exactly two leaf children, which represent both suffixes of S and T.



Example of finding MUMs (III)

- Step 3: For each marked node, WLOG, 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 an MUM.



Time analysis

- Step 1: Building suffix tree takes $O(m_1+m_2)$ time.
- Step 2: Mark internal nodes takes $O(m_1+m_2)$ time.
- Step 3: Extracting MUM also takes $O(m_1+m_2)$ time.
- In total, $O(m_1+m_2)$ time.

Good news!

- When we do experiment, we found that MUMs can cover nearly 100% of the know conserved gene pairs.

The problem solved?

- Now, we can find MUMs in linear time! Is the problem solved?

Ans: No!

Mouse Chr No.	Human Chr No.	# of Published Gene Pairs	# of MUMs
2	15	51	96,473
7	19	192	52,394
14	3	23	58,708
14	9	28	29,814

No. of MUMs >> no. of gene pairs!

There are many noise!

How can we extract the right MUMs?

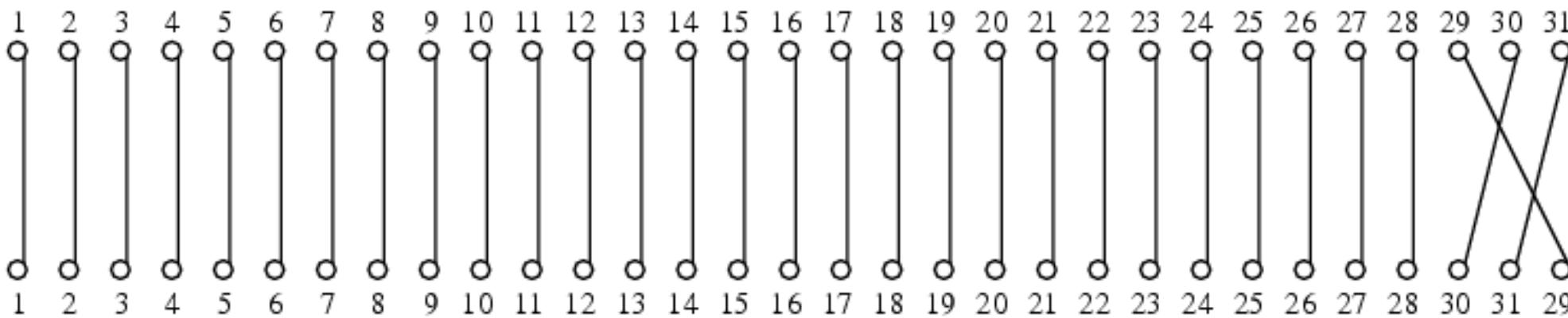
17	19	30	56,536
18	5	64	131,850
19	9	22	62,296
19	11	93	29,814

Observation 2

- Two related species should preserve the ordering of most conserved genes!

Conserved genes in Mouse Chromosome 16 and Human Chromosome 16

Mouse Chromosome 16



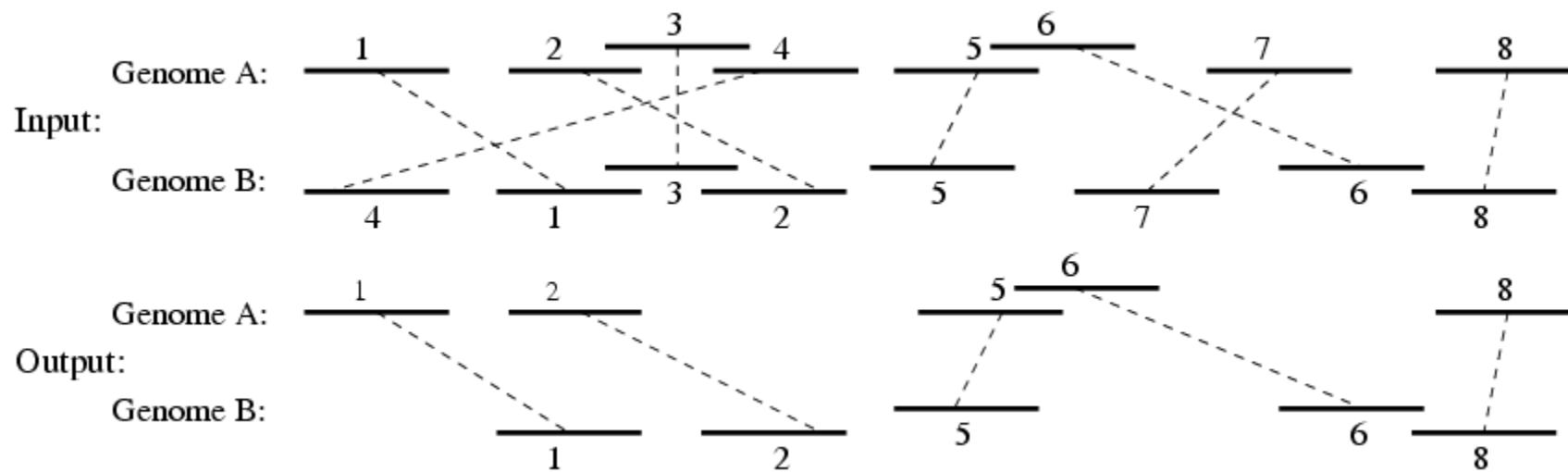
Human Chromosome 16

- (a) There are 31 conserved gene pairs found in Mouse chromosome 16 and Human chromosome 16. The genes are labeled according to the positions of the genes in the mouse chromosome from the 5' end. The corresponding genes in human are drawn according to their relative positions from the 5' end of human chromosome.

Solution 2

- Instead of reporting all MUMs to the user.
- We compute the longest common subsequence (LCS) of all MUMs.
- We report only the MUMs on the LCS.

Example of LCS



12345678

41325768



12345678

41325768

The longest common subsequence (LCS) problem

- Suppose there are n MUMs.
- Input: two sequences $A[1..n]=a_1a_2\dots a_n$ and $B[1..n]=b_1b_2\dots b_n$ of n distinct characters.
- Output: the longest common subsequence.

How to find LCS in $O(n^2)$ time?

- Let $C_i[j]$ be the length of the longest common subsequence of $A[1..i]$ and $B[1..j]$.
- Let $\delta(i)$ be the index of the character in B such that $a_i = b_{\delta(i)}$
- Note that $C_i[0]=C_0[j]=0$ for all $0 \leq j \leq n$; and

$$C_i[j] = \max \begin{cases} C_{i-1}[j] \\ 1 + C_{i-1}[\delta(i) - 1] & \text{if } j \geq \delta(i) \end{cases}$$

- Note that the length of the $\text{LCS}(A, B) = C_n[n]$.
- By dynamic programming, the problem is solved in $O(n^2)$ time.

Example (I)

- A[1..8]=12345678
 - B[1..8]=41325768

Example (II)

- $A[1..8]=12345678$
 - $B[1..8]=41325768$
 - Base cases:
 - $C_i[0]=0$ and $C_0[j]=0$

Example (III)

- $A[1..8]=12345678$
 - $B[1..8]=41325768$
 - Fill the table row by row with the recursive formula:

$$C_i[j] = \max \begin{cases} C_{i-1}[j] \\ 1 + C_{i-1}[\delta(i) - 1] & \text{if } j \geq \delta(i) \end{cases}$$

$C_2[4] = \text{LCS}(A[1..2], B[1..4]).$

By the recursive formula,
 $C_2[4]=\max\{C_1[4], 1+C_1[3]\}=2$
(Note: $\delta(2)=4$.)

Example (IV)

- A[1..8]=12345678
- B[1..8]=41325768
- Fill the table row by row with the recursive formula:

$$C_i[j] = \max \begin{cases} C_{i-1}[j] \\ 1 + C_{i-1}[\delta(i)-1] & \text{if } j \geq \delta(i) \end{cases}$$

LCS(A,B)=C₈[8]=5.

C	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1
2	0	0	1	1	2	2	2	2	2
3	0	0	1	2	2	2	2	2	2
4	0	1	1	2	2	2	2	2	2
5	0	1	1	2	2	3	3	3	3
6	0	1	1	2	2	3	3	4	4
7	0	1	1	2	2	3	4	4	4
8	0	1	1	2	2	3	4	4	5

We can extract more information

- As a side-product, we can retrieve $\text{LCS}(A[1..i], B[1..j])$, where $1 \leq i, j \leq n$, in $O(1)$ time.

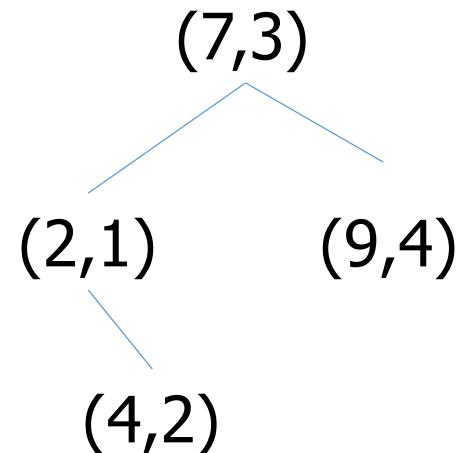
Sparsification

- Observation:
 - $C_i[1] \leq C_i[2] \leq \dots \leq C_i[n]$
- Instead of storing $C_i[0..n]$ explicitly, we can store only the boundaries at which the values change.
- Precisely, we store $(j, C_i[j])$ for all j such that $C_i[j] > C_i[j-1]$.
- Example: suppose $C_i[0..9]=0011222334$.
 - We store $(2,1), (4,2), (7,3), (9, 4)$.

How to store the list of tuples?

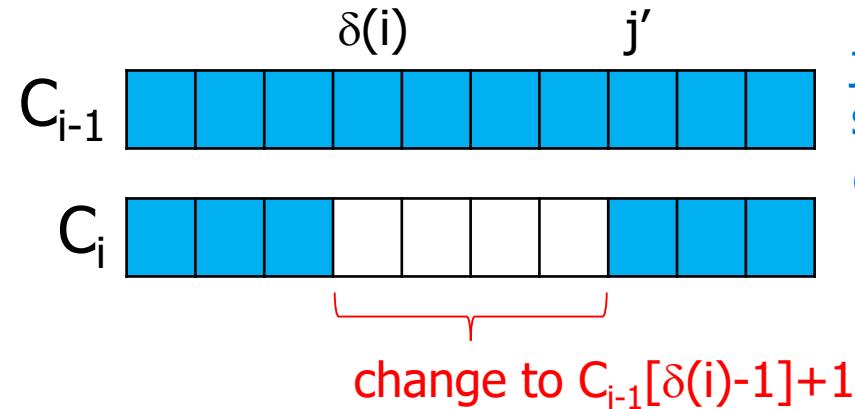
- We can store all tuples $(j, C_i[j])$ in a balanced binary search tree T_i .
- Binary search tree ensures, for every tuple, all tuples on the left are smaller than it while all tuples on the right are bigger.
- Balanced binary search tree ensures the height of the tree is $O(\log n)$.
- Then, every search, insert, delete operation takes $O(\log n)$ time.

- Example: suppose $C_i[0..9]=0011222334$.
 - The list of tuples is $(2,1), (4,2), (7,3), (9, 4)$.
 - We store the balanced binary search tree T_i .



$$C_i[j] = \max \begin{cases} C_{i-1}[j] \\ 1 + C_{i-1}[\delta(i) - 1] & \text{if } j \geq \delta(i) \end{cases}$$

Lemma



- Lemma:

$$C_i[j] = \begin{cases} C_{i-1}[\delta(i) - 1] + 1 & \text{if } j \geq \delta(i) \text{ and } C_{i-1}[\delta(i) - 1] + 1 > C_{i-1}[j] \\ C_{i-1}[j] & \text{otherwise} \end{cases}$$

Proof:

- When $j < \delta(i)$, $A[i]$ cannot involve. Hence, $C_i[j] = C_{i-1}[j]$.
- When $\delta(i) \leq j$, $C_i[j] = \max\{C_{i-1}[j], C_{i-1}[\delta(i) - 1] + 1\}$. There are two cases:
 - Case 1: $C_{i-1}[j] > C_{i-1}[\delta(i) - 1] + 1$. Then $C_i[j] = C_{i-1}[j]$.
 - Case 2: Otherwise, $C_i[j] = C_{i-1}[\delta(i)-1]+1$.

j' is the smallest position
s.t. $j' \geq \delta(i)$ and
 $C_{i-1}[j'] \geq C_{i-1}[\delta(i) - 1] + 1$.

Example

- $A[1..9]=123456789$
- $B[1..9]=245831697$
- $\delta(8)=4$.
- $T_7=(1,1), (2,2), (3,3), (7,4), (9,5)$.
- If $j < \delta(8)$, $C_8[j] = C_7[j]$.
- If $j \geq \delta(8)$, $C_7[\delta(8)-1]+1=4$
- $j'=7$ is the smallest position s.t.
 $C_7[j'] \geq C_7[\delta(8) - 1] + 1$
- $C_8[j]=4$ for $\delta(8) \leq j < 7$

$\delta(8) = 4$
 j'

C_{8-1}	0	1	2	3	3	3	3	4	4	5
C_8										

change to $C_8[\delta(8)-1]+1=4$

C	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1	1
2	0	1	1	1	1	1	1	1	1	1
3	0	1	1	1	1	2	2	2	2	2
4	0	1	2	2	2	2	2	2	2	2
5	0	1	2	3	3	3	3	3	3	3
6	0	1	2	3	3	3	3	4	4	4
7	0	1	2	3	3	3	3	4	4	5
8	0	1	2	3	4	4	4	4	4	5
9										

$\delta(8)$

Algorithm

Algorithm Suffix-tree-MUM

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

Ensure: The LCS between A and B

1: Construct T_1 which contains one tuple $(\delta(1), 1)$.

2: **for** $i = 2$ to K **do**

3: Delete all tuples $(j, C_{i-1}[j])$ where $j \geq \delta(i)$ and $C_{i-1}[j] \leq C_{i-1}[\delta(i) - 1] + 1$;

4: Insert $(\delta(i), C_{i-1}[\delta(i)-1]+1)$.

5: **end for**

Construct T_i from T_{i-1}

Example: Transform T_7 to T_8

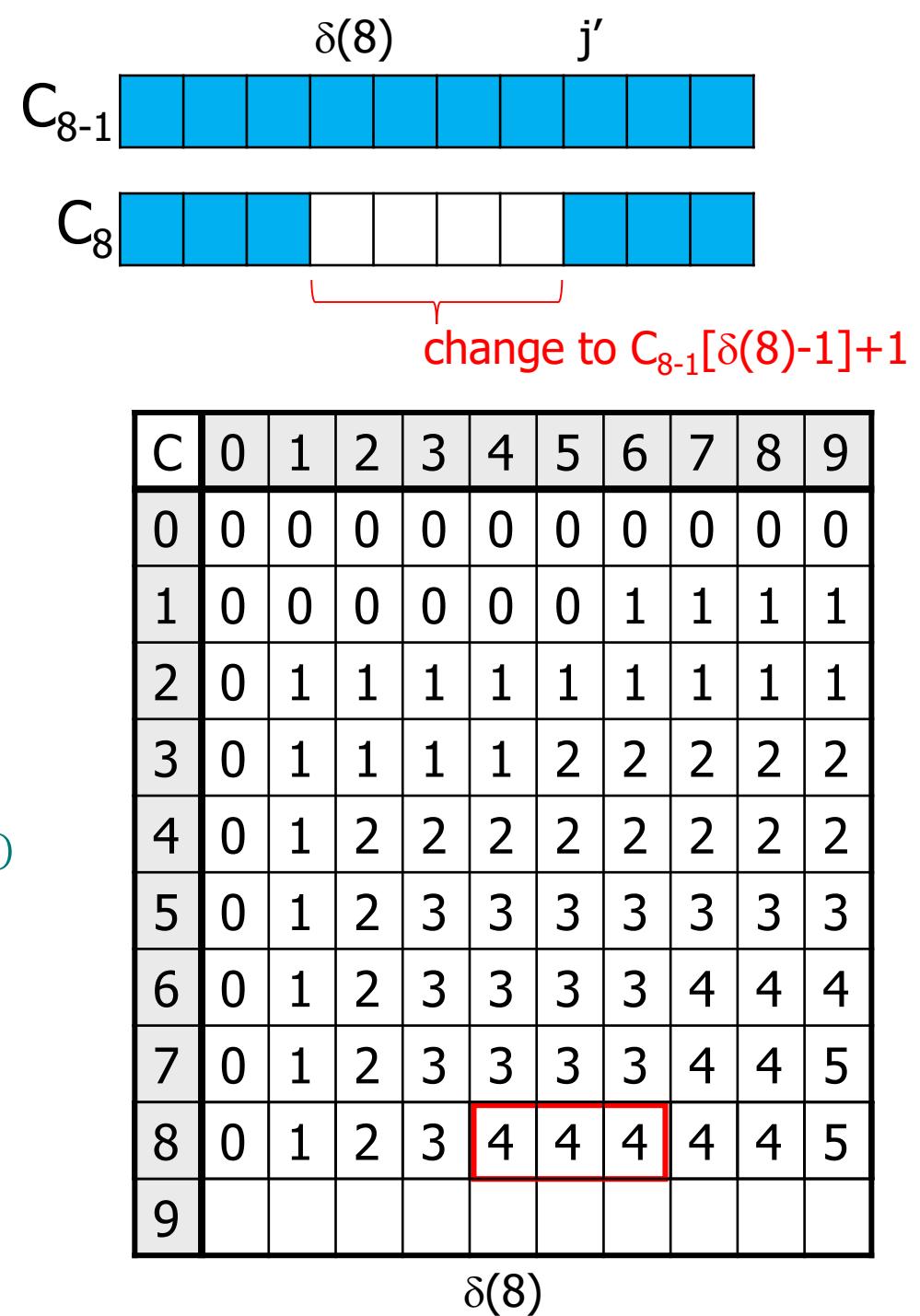
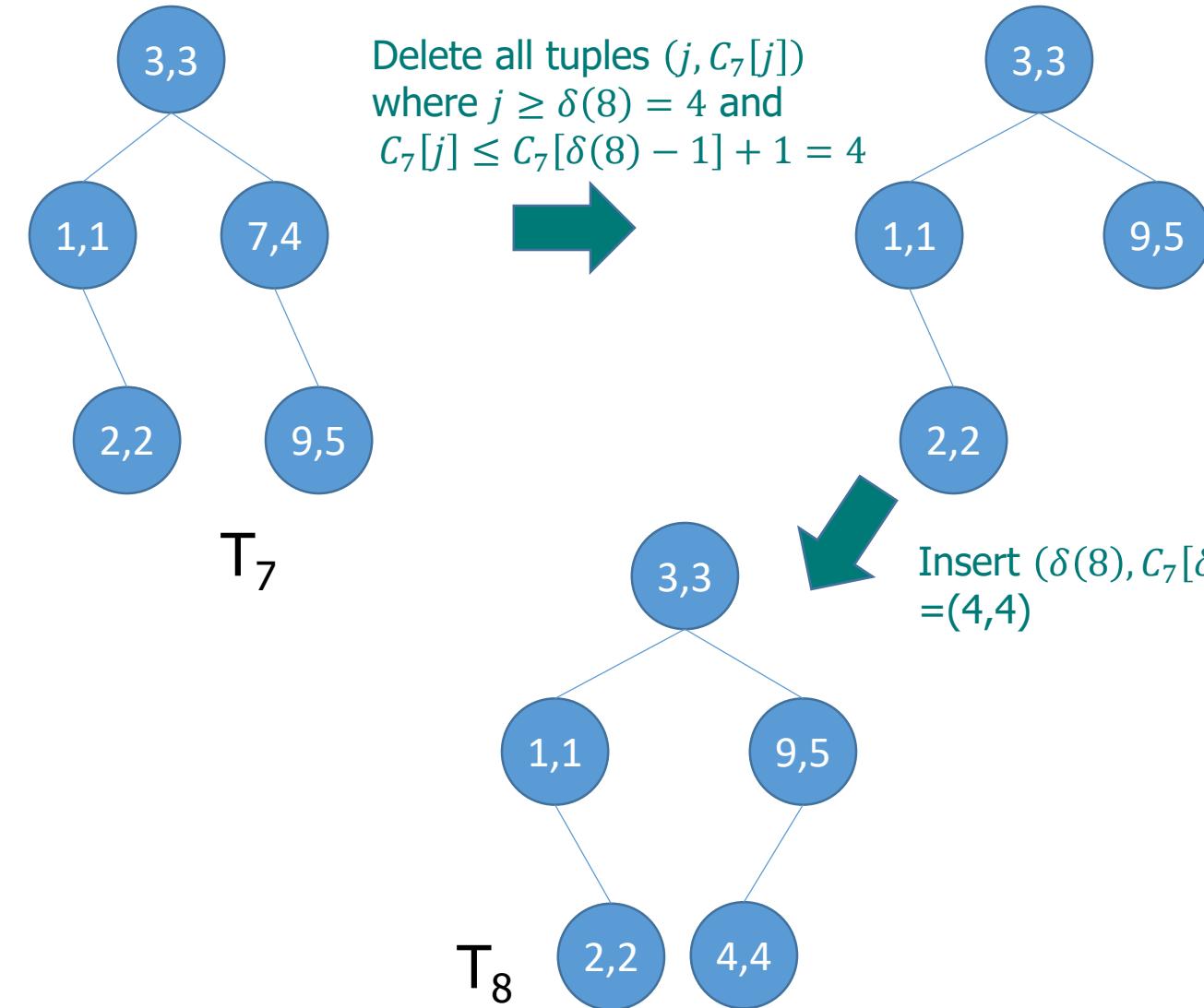
123456789

A=123456789

B=245831697

- $\delta(1)=6, \delta(2)=1, \delta(3)=5, \delta(4)=2, \delta(5)=3, \delta(6)=7, \delta(7)=9, \delta(8)=4, \delta(9)=8.$
- $C_7[0..9]=0123333445 T_7=(1,1),(2,2),(3,3),(7,4),(9,5)$
- Note that $C_7[\delta(8)-1]+1=4$
- Delete all tuples $(j, C_7[j])$ where $j \geq \delta(8)=4$ and $C_7[j] \leq 4$
 - Remove $(7,4)$ from T_7
- Insert $(\delta(8), C_7[\delta(8)-1]+1)$
 - Insert $(4,4)$ to $T_7 \rightarrow T_8$
- $C_8[0..9]=0123444445 T_8=(1,1),(2,2),(3,3),(4,4),(9,5)$

$T_7 \rightarrow T_8$



Time analysis

- To build T_i from T_{i-1} , suppose we need to delete α_i tuples.
- Then, T_i can be constructed in $O((\alpha_i + 1) \log n)$ time.
- In total, T_n can be constructed in $O((n + \alpha_1 + \alpha_2 + \dots + \alpha_n) \log n)$ time.
- Since we can delete at most n tuples, T_n can be constructed in $O(n \log n)$ time.

We can extract more information

- As a side-product, we can retrieve $\text{LCS}(A[1..i], B[1..j])$, where $1 \leq i, j \leq n$, in $O(\log n)$ time.
 - Idea: By searching the tuple $(j', C_i[j'])$ in T_i such that j' is just smaller than j .
 - Then, we report $C_i[j']$.
 - Searching in T_i takes $O(\log n)$ time.

Analysis

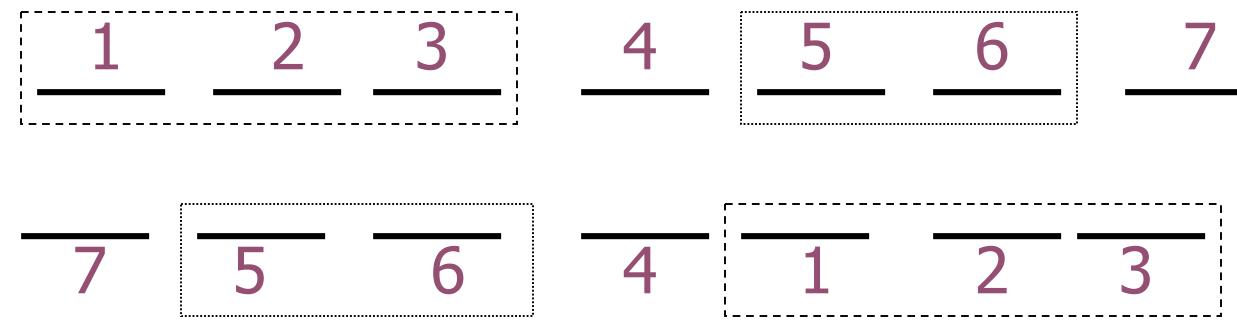
- In conclusion, LCS can be computed in $O(n \log n)$ time.
- This approach is used by **MUMmer1**.

Problem of this approach

- This approach assumes there exist a single long alignment. Moreover, such assumption may not be always true.
- Therefore, for many cases, LCS can only discover few genes.

Observation 3

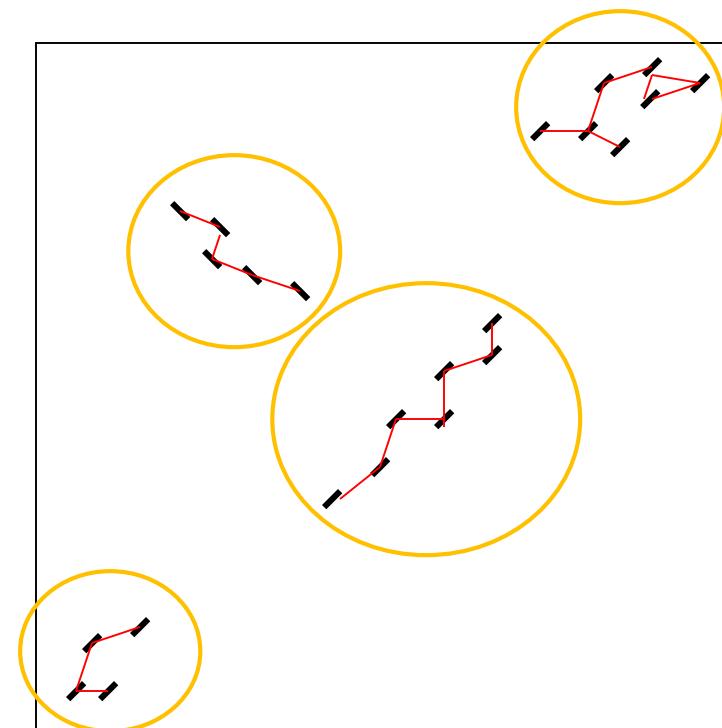
- A pair of conserved genes are likely corresponding to a sequence of MUMs that are consecutive and close in both genomes and have sufficient length.



- The set of such substrings is called a **cluster**.

Clustering method of MUMmer2

- By suffix tree, obtains a set of MUMs.
- Two MUMs will be linked if they are sufficiently close and on similar diagonal.
- Each connected component form a cluster.
- Run MUMmer1 to obtain LCS.



Solution 3

- Based on Observation 3, MUMmer2 and MUMmer3 try to identify maximal clusters in the genomes.
- This approach is quite good. In our experiment, MUMmer3 can identify ~76.6% of the published gene pairs.

Can we further improve?

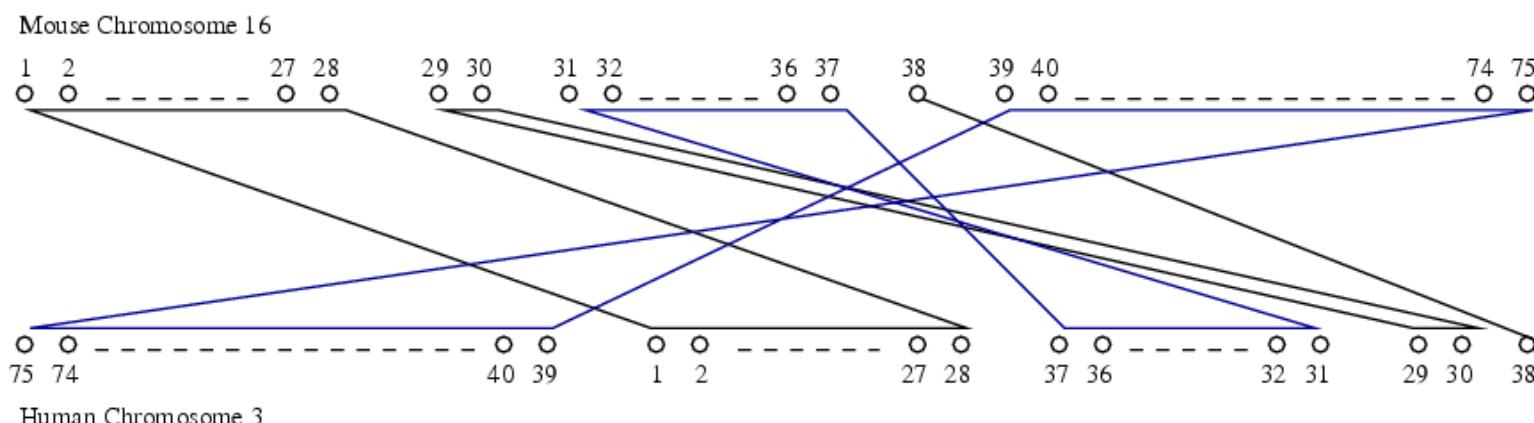
- Yes. We propose the Similar Subsequence Problem.
- In our experiment, we can identify ~91.3% of the published gene pairs.

Observation 4

- If two genomes are closely related, they can be transformed from each other using a few transpositions/reversals.

Example

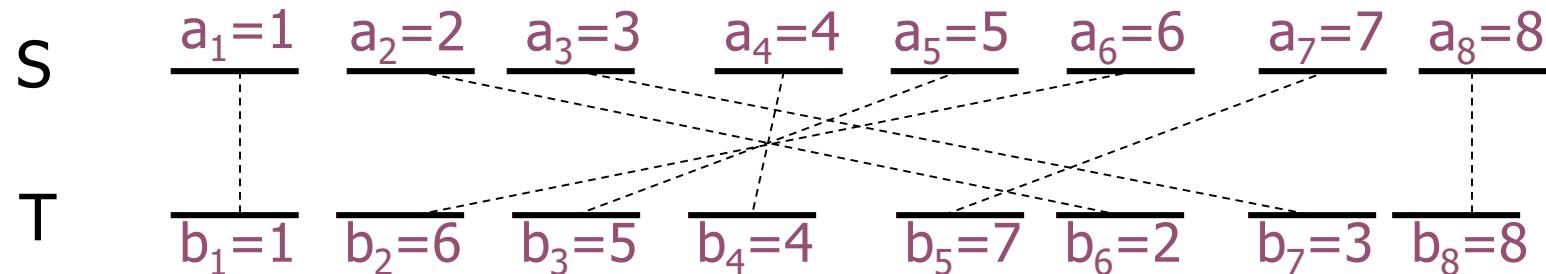
- By two transposition/reversal operations, we can transform Mouse Chr 16 to Human Chr 3.



(b) There are 31 conserved gene pairs found in Mouse chromosome 16 and Human chromosome 3. The genes are labeled as in (a). Note that the relative ordering of Genes 31 to 37 in human chromosome is exactly the reverse of that in the mouse chromosome. The same situation occurs in Genes 39 to 75.

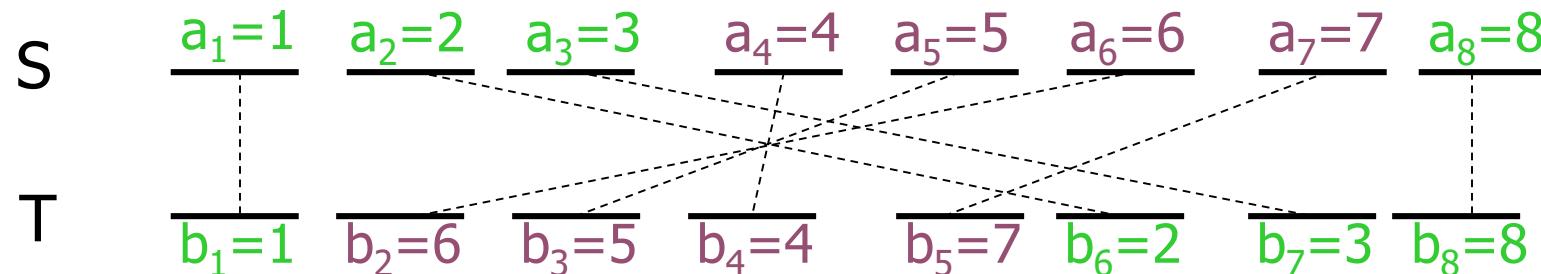
Input

- Given two genomes S and T. Assume we already know the n MUMs.
- Let $A=(a_1, a_2, \dots, a_n)$ and $B=(b_1, b_2, \dots, b_n)$, respectively, be the order of the n MUMs in S and T.



Common subsequence

- A sequence $C=(c_1, c_2, \dots, c_m)$ is a **common subsequence** of A and B if C is a subsequence of both A and B
- For example, $C=(1,2,3,8)$ is a common subsequence of A and B.
- The **weight** of a common subsequence is the total weight of the MUMs
- A **maximum weight common subsequence (MWCS)** of A and B is a subsequence with the heaviest weight.

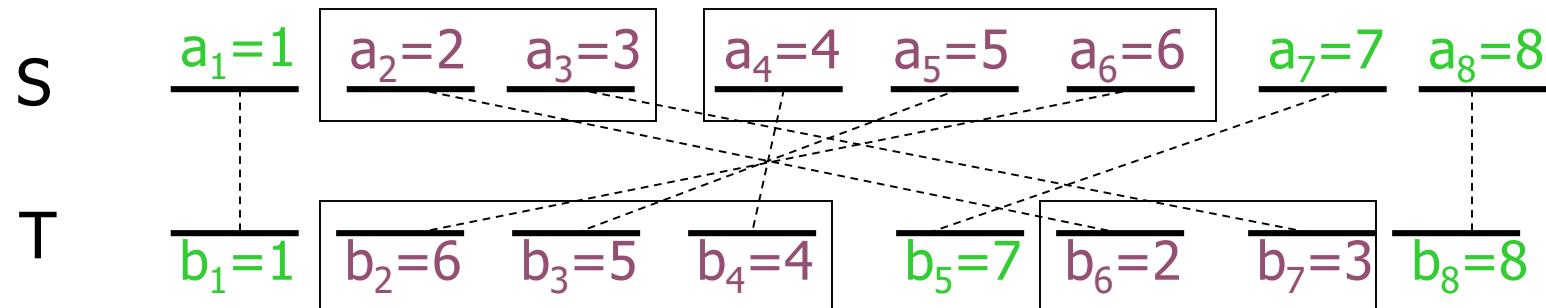


Maximum weight common subsequence (MWCS) problem

- Given $A[1..n]$ and $B[1..n]$,
 - we can compute an MWCS of A and B in $O(n \log n)$ time.
- Idea: similar to the computation of LCS.
- Remark: as a side-product, we can retrieve MWCS of $A[1..i]$ and $B[1..j]$, where $1 \leq i, j \leq n$, in $O(\log n)$ time.

Similar Subsequence

- A **k-similar subsequence** consists of k blocks and a backbone.
 - The backbone is a common subsequence with k blocks inserted into it.
 - Each block is a common subsequence or reversed common subsequence while all of them are disjoint.
- Below is an example of 2-similar subsequence.
- In some sense, k-similar subsequence models k transpositions/reversals.



Similar Subsequence Problem

- Given two sequences A and B and a parameter k,
 - the [Similar Subsequence Problem](#) finds a k-similar subsequence with the heaviest weight.
-
- This problem is NP-complete in general.
 - For a constant k, we can solve the problem in $O(n^{2k+1} \log n)$ time.
 - We devise a heuristic algorithm to solve it in $O(n^2(\log n + k))$ time.

Heuristic algorithm: idea

1. Find the backbone first.
2. For every interval $i..j$, compute the score of inserting such subsequence into the backbone.
3. Find k non-overlapping intervals which maximizes the total score.

Heuristic algorithm: Step 1

- Find the MWCS of A and B. Treat this as the backbone.
 - $O(n \log n)$ time

Heuristic algorithm: Step 2

- This step compute the score for inserting one block $A[i..j]$ to $B[\delta(i)..δ(j)]$
- Example: Moving $A[1..3]$ to $B[8..5]$
 - Note: This is a transposition and reversal
 - $A = \underline{123}456789$
 - $B = 4769\underline{325}18$
- This step consists of two substeps.

Heuristic algorithm: Step 2.1

- Compute MWCS($A[i..j]$, $B[\delta(i).. \delta(j)]$) for all $1 \leq i, j \leq n$.
 - Brute force: $O(n^3 \log n)$ time
 - Better algorithm: $O(n^2 \log n)$ time
 - For each i ,
 - Find the MWCS for $A[i..n]$ and $B[\delta(i)..n]$ in $O(n \log n)$ time
 - Find the MWCS for $A[i..n]$ and reverse of $B[1..\delta(i)]$ in $O(n \log n)$ time
 - Retrieve MWCS($A[i..j]$, $B[\delta(i).. \delta(j)]$) in $O(\log n)$ time for every $j \geq i$

Example

- $A[1..8]=123456789$, $B[1..8]=746932518$
- For $i=6$, $\delta(i)=3$, we build the DP table for $(A[6..9], B[3..9])$ and the DP table for $(A[6..9], \text{reverse of } B[1..3])$

		B[3..9]							
		-	6	9	3	2	5	1	8
-	-	0	0	0	0	0	0	0	0
	6	0	1	1	1	1	1	1	1
7	0	1	1	1	1	1	1	1	1
8	0	1	1	1	1	1	1	1	2
9	0	1	2	2	2	2	2	2	2

Reverse of B[1..3]

		-	6	4	7
		0	0	0	0
-	-	0	1	1	1
	6	0	1	1	2
7	0	1	1	2	
8	0	1	1	2	
9	0	1	1	2	

A[6..9]

A[6..9]

Heuristic algorithm: Step 2.2

- For every interval $i..j$,
 - Define $\text{Score}(i..j)$ to be
 - $\text{MWCS}(A[i..j], B[\delta(i).. \delta(j)])$ –
 - the total weight of characters in the backbone that fall into $A[i..j]$ or $B[\delta(i).. \delta(j)]$
 - (See example in the next slide.)

Example

- $A[1..8]=12345678$
- $B[1..8]=41325768$
- Backbone is 12568
 - $A[1..8]=\underline{12}34\underline{56}78$
 - $B[1..8]=4\underline{13}25768$
- Consider mutating $A[3..4]$ to $B[\delta(3).. \delta(4)] = B[3..1]$.
 - $A[1..8]=\underline{12}\underline{34}5678$
 - $B[1..8]=\underline{413}25768$
- After mutation, we have
 - $A[1..8]=12\underline{34}5678$
 - $B[1..8]=\underline{413}25768$

Note:

$MWCS(A[3..4], B[\delta(3).. \delta(4)])=2$

‘1’ falls in $B[1..3]$.

So, $Score(3..4)=2-1=1$

Heuristic algorithm: Step 3

- Among all intervals,
 - Find k intervals $i_1..j_1, i_2..j_2, \dots, i_k..j_k$ such that they are mutually disjoint and maximize the sum of their scores, that is, $\sum_{p=1,2,\dots,k} \text{Score}(i_p..j_p)$.
- This step can be done in $O(k n^2)$ time by dynamic programming.
[Exercise]

Heuristic algorithm: Step 4

- Refine the k intervals $i..j$ so that $B[\delta(i)..δ(j)]$ are disjoint.
- While there exists $\delta(i)..δ(j)$ and $\delta(i')..δ(j')$ that are overlapping,
 - We examine all possible ways to shrink the intervals $i..j$ and $i'..j'$ so that the score is maximize and $\delta(i)..δ(j)$ and $\delta(i')..δ(j')$ are not overlap.
- $O(k^2)$ time

Heuristic algorithm: summary

- The total time is $O(n^2(\log n + k))$.

Solution 4

- Given two genomes S and T,

Mutation Sensitive Alignment (MSA) Algorithm

- Find all the MUMs
- Solve the similar subsequence problem
- Report all the MUMs on the k-similar subsequence.

Example

S	$\underline{a_1=1}$	$\underline{a_2=2}$	$\underline{a_3=3}$	$\underline{\underline{a_4=4}}$	$\underline{a_5=5}$	$\underline{a_6=6}$	$\underline{a_7=7}$	$\underline{a_8=8}$	$\underline{a_9=9}$
T	$\underline{b_1=2}$	$\underline{b_2=1}$	$\underline{b_3=7}$	$b_4=6$	$\underline{b_5=5}$	$\underline{b_6=8}$	$\underline{b_7=3}$	$\underline{b_8=4}$	$\underline{b_9=9}$



S	$\underline{a_1=1}$	$\underline{a_2=2}$	$\boxed{a_3=3 \quad \underline{a_4=4}}$	$\boxed{a_5=5 \quad a_6=6 \quad a_7=7}$	$\underline{a_8=8}$	$\underline{a_9=9}$	
T	$\underline{b_1=2}$	$\underline{b_2=1}$	$\boxed{b_3=7 \quad b_4=6}$	$\boxed{b_5=5}$	$\underline{b_6=8}$	$\boxed{b_7=3 \quad b_8=4}$	$\underline{b_9=9}$

Experiment on human and mouse

- We apply MUMmer3 and MSA to the following 15 pairs of chromosomes.

For MSA, we set
k=4!

Mouse Chr No.	Human Chr No.	# 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

Experiment on human and mouse

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%

- Coverage: % of published genes covered
- Preciseness: % of MUMs reside in some published gene pairs.

Experiment result on Baculoviridae genomes that are in the same genus

Experiment	Virus	Virus	Length	# of MUMs	# of Conserved Genes
1	AcMNPV	BmNPV	133K*128K	35,166	134
2	AcMNPV	HaSNPV	133K*131K	64,291	98
3	AcMNPV	LdMNPV	133K*161K	65,227	95
4	AcMNPV	OpMNPV	133K*131K	59,949	126
5	AcMNPV	SeMNPV	133K*135K	66,898	100
6	BmNPV	HaSNPV	128K*131K	63,939	98
7	BmNPV	LdMNPV	128K*161K	63,086	93
8	BmNPV	OpMNPV	128K*131K	58,657	122
9	BmNPV	SeMNPV	128K*135K	66,448	99
10	HaSNPV	LdMNPV	131K*161K	57,618	92
11	HaSNPV	OpMNPV	131K*131K	59,125	95
12	HaSNPV	SeMNPV	131K*135K	64,980	101
13	LdMNPV	OpMNPV	161K*131K	75,906	98
14	LdMNPV	SeMNPV	161K*135K	62,545	102
15	OpMNPV	SeMNPV	131K*135K	63,261	101
16	CpGV	PxGV	123K*100K	59,733	97
17	CpGV	XcGV	123K*178K	63,258	107
18	PxGV	XcGV	100K*178K	81,020	99

- We apply MUMmer3 and MSA to the following 15 pairs of viruses.
- MUM pairs of length at least three amino acids
- For MSS, k=20

Experiment result on Baculoviridae genomes that are in the same genus

Experiment	Coverage		Preciseness	
	MUMmer	MSS	MUMmer	MSS
1	100% (134)	100% (134)	44%	91%
2	58% (57)	80% (78)	80%	85%
3	58% (55)	69% (66)	64%	80%
4	83% (105)	95% (120)	91%	94%
5	61% (61)	68% (68)	70%	85%
6	59% (58)	73% (72)	78%	87%
7	58% (54)	69% (64)	47%	79%
8	83% (101)	94% (115)	86%	94%
9	63% (62)	69% (68)	65%	86%
10	75% (69)	85% (78)	75%	85%
11	53% (50)	68% (65)	77%	83%
12	75% (76)	87% (88)	71%	93%
13	60% (59)	67% (66)	52%	89%
14	74% (75)	75% (77)	65%	90%
15	52% (53)	63% (64)	66%	82%
16	61% (59)	85% (82)	83%	89%
17	58% (62)	74% (79)	83%	84%
18	61% (60)	76% (75)	76%	86%
average	66%	78%	71%	87%

- Coverage: % of published genes covered
- Preciseness: % of MUMs reside in some published gene pairs.

Reference

- A.L. Delcher, S. Kasif, R.D. Fleischmann, J. Peterson, O. White, and S.L. Salzberg. Alignment of whole genomes, Nucleic Acids Research, 27(11):2369-2376, 1999. (MUMmer1)
- A.L. Delcher, A. Phillippy, J. Carlton, and S.L. Salzberg. Fast algorithms for large-scale genome alignment and comparison, Nucleic Acids Research, Nuclei Acids Research, 30(11):2478-2483, 2002. (MUMmer2)
- T.W. Lam, N. Lu, H.F. Ting, W.H. Wong, and S.M. Yiu. Efficient Algorithms for Optimizing Whole Genome Alignment with Noise, ISAAC, 2003.
- H.L. Chan, T.W. Lam, W.K. Sung, W.H. Wong, S.M. Yiu, and X. Fan. The Mutated Subsequence Problem and Locating Conserved Genes, Bioinformatics, 2005.
- Tutorial:
 - <https://mummer4.github.io/tutorial/tutorial.html>