



Longest Common Subsequence as Private Search

Payman Mohassel and Mark Gondree

U of Calgary

NPS

Longest Common Subsequence

- Strings A, B

- $|A| = n, |B| = m$

- $LCS(A, B)$

- Subsequence of A
 - Subsequence of B
 - Maximum length

A = a d e f t c g

B = c e d e t f g

$LCS(A, B) = d e t g$

Three Variants

- LCS length : 4
 - Length of LCS
- LCS string : “detg”
 - A string that is an LCS
- LCS embed : $(\{2,3,5,7\},\{3,4,5,7\})$
 - Position of an LCS in A and B

A = a d e f t c g

B = c e d e t f g

Motivation

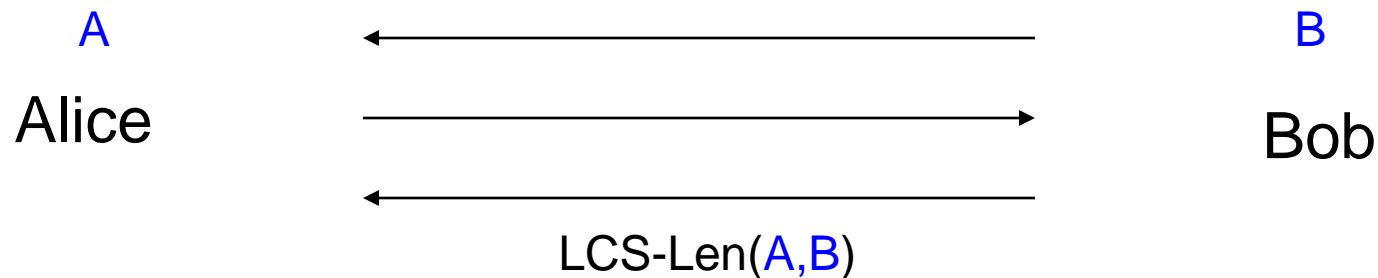
- Genomic computation
 - Measure of similarity between DNA sequences
- File comparison
 - Used in diff program
- Many problems related to LCS
 - Edit distance (sequence alignment)
 - Shortest common supersequence
 - Longest increasing sequence
- Beyond LCS
 - Problems with **dynamic programming solutions**
 - Problems **formulated using DFAs**



Privacy Concerns

- DNA data is sensitive
 - Patients' privacy: health, background
- Many institutions hold genomic data
 - Governments
 - Pharmaceutical companies
 - Research institutes
- Privacy rules (e.g. HIPAA)
 - Can lead to fines or legal action
 - Focus on data access and sharing, not computation

Secure Two-Party LCS



- Atallah et. al. [WPES 03](#)
- Jha et. al. [S&P 08](#)
 - With implementation
 - Using secure computation techniques
- What if we output more than length?

Output More Than Length

- LCS string or embedding
- More than one possible output
 - Can be exponentially many
- Which one do we output?
 - There are privacy concerns
 - Output can work as a “covert channel”
- **Private search**
 - Beimel et. al. [STOC 06](#), [CRYPTO 07](#)
 - Output sampling, equivalence protecting algorithms

Output Sampling Algorithms

- Choice of output leaks information
 - E.g. lexicographically first LCS reveals
 - Any string before the output not an LCS
- Sample solution space randomly
- Not possible for all problems
 - Stable matching



Output Sampling for LCS

- Fill in the LCS length table
- Fill in the LCS counting table
- Devise a randomized backtracking strategy using the tables

Step 1: LCS Length

	a	f	b	c	a	a	a	d
a	1	1	1	1	1	1	1	1
b	1	1	2	2	1	1	1	1
f	1	2	2	2	2	2	2	2
c	1	2	2	3	3	3	3	3
e	1	2	2	3	3	3	3	3
a	1	2	2	3	4	4	4	4
d	1	2	2	3	4	4	4	5

- Dynamic programming
- $O(mn)$ time

Length of LCS between
“afbcaa” and “abfc”

Between “afbcaaad” and “abfceaad”

Step 2: LCS Counting

	a	f	b	c	a	a	a	d
a	1	1	1	1	1	1	1	1
b	1	1	1	1	1	1	1	1
f	1	1	2	2	2	2	2	2
c	1	1	2	2	2	2	2	2
e	1	1	2	2	2	2	2	2
a	1	1	2	2	2	2	2	2
d	1	1	2	2	2	2	2	2

- Greenberg, 2003
- $O(mn \log S)$ time
- S = number of solutions
- Different for string and embed

Number of LCS strings
between "afbcaa" and "abfce"

Between "afbcaaad" and "abfcead"

Step 3: Randomized Backtracking

- Backtracking on the counting table
- Create multiple choices at each step
- To get a uniformly random solution
 - Cover all possible solutions
 - No **solution overlap** between choices
 - **Weighted coin-toss** based on number of solutions in each choice
- LCS embed strategy: at (i,j)
 - $A[i]$ and $B[j]$ are in LCS
 - $A[i]$ in LCS, $B[j]$ not
 - $B[j]$ in LCS, $A[i]$ not
 - Neither one in LCS
- Different strategy for LCS string
- $O(mn \log S)$ time



Equivalence Protecting Algorithms

- What if protocol is run multiple times?
 - Leaks additional information
- Protect privacy of inputs with the same set of solutions

Equivalence Classes

- LCS-Str(A,B)

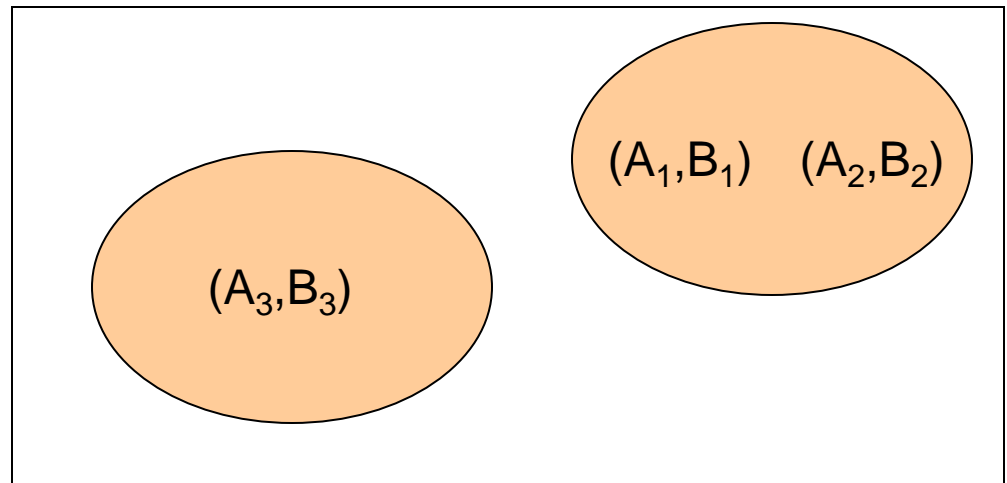
- Set of all LCS strings between A and B

$$(A_1, B_1) \approx (A_2, B_2)$$

$$\text{LCS-Str}(A_1, B_1) = \text{LCS-Str}(A_2, B_2)$$

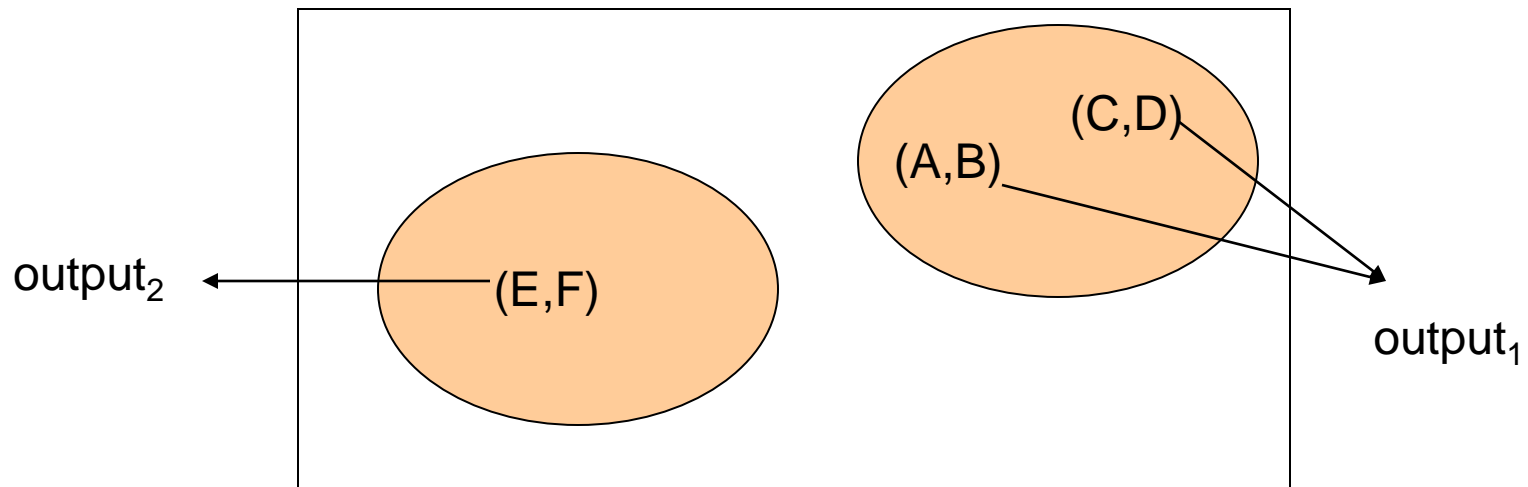
$$(A_3, B_3) \not\approx (A_2, B_2)$$

$$\text{LCS-Str}(A_3, B_3) \neq \text{LCS-Str}(A_2, B_2)$$



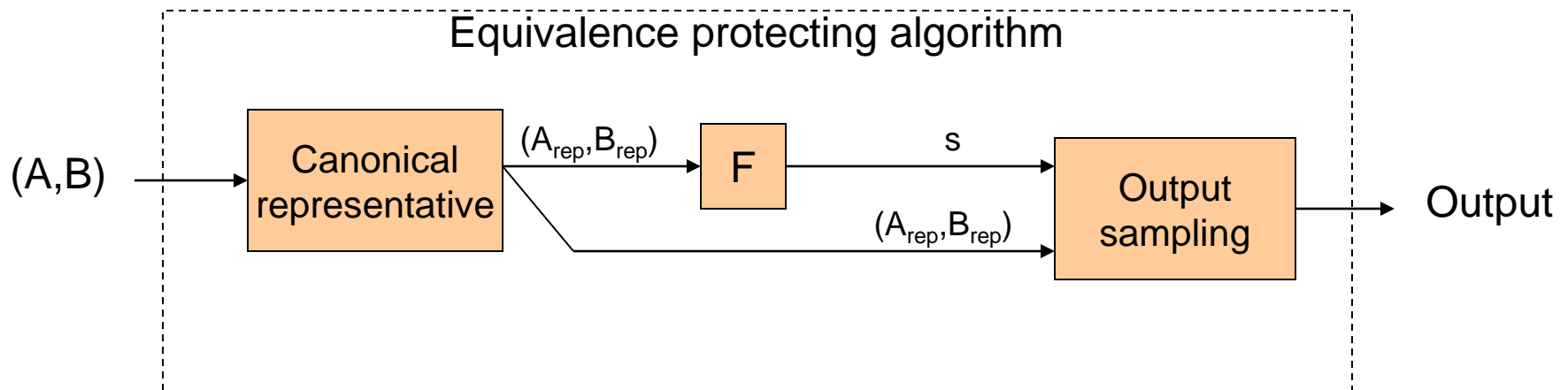
Equivalence Protecting Algorithms

- Samples $\text{LCS-Str}(A,B)$ randomly
- Sampling is seeded
- For every $(C,D) \approx (A,B)$ use the same seed



A Generic Construction

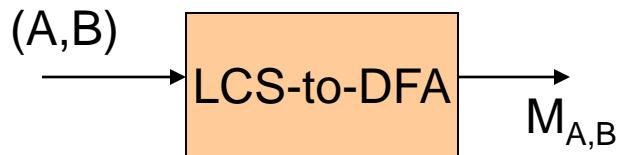
- Compute a $(A_{\text{rep}}, B_{\text{rep}})$
 - Representative of (A, B) 's equivalence class
 - **Canonical representative** algorithm
- Compute the seed $s = F(A_{\text{rep}}, B_{\text{rep}})$
- Run an output sampling algorithm on $(A_{\text{rep}}, B_{\text{rep}})$ using s for randomness



Equivalence Protecting Algorithms

- Problem reduced to
 - Canonical representative algorithms
- LCS embed
 - Direct solution
- **LCS string**
 - **Via reduction to DFAs**

Step 1



- $M_{A,B}$ is a DFA of size $O(mn)$
- $L(M_{A,B}) = \text{LCS-Str}(A, B)$

- M_A generates all subsequence of A
- M_B generates all subsequences of B
- Combine and prune the DFAs
- Runs in $O(mn)$ time

Equivalence protecting algorithm that outputs a word in $L(M_{A,B})$

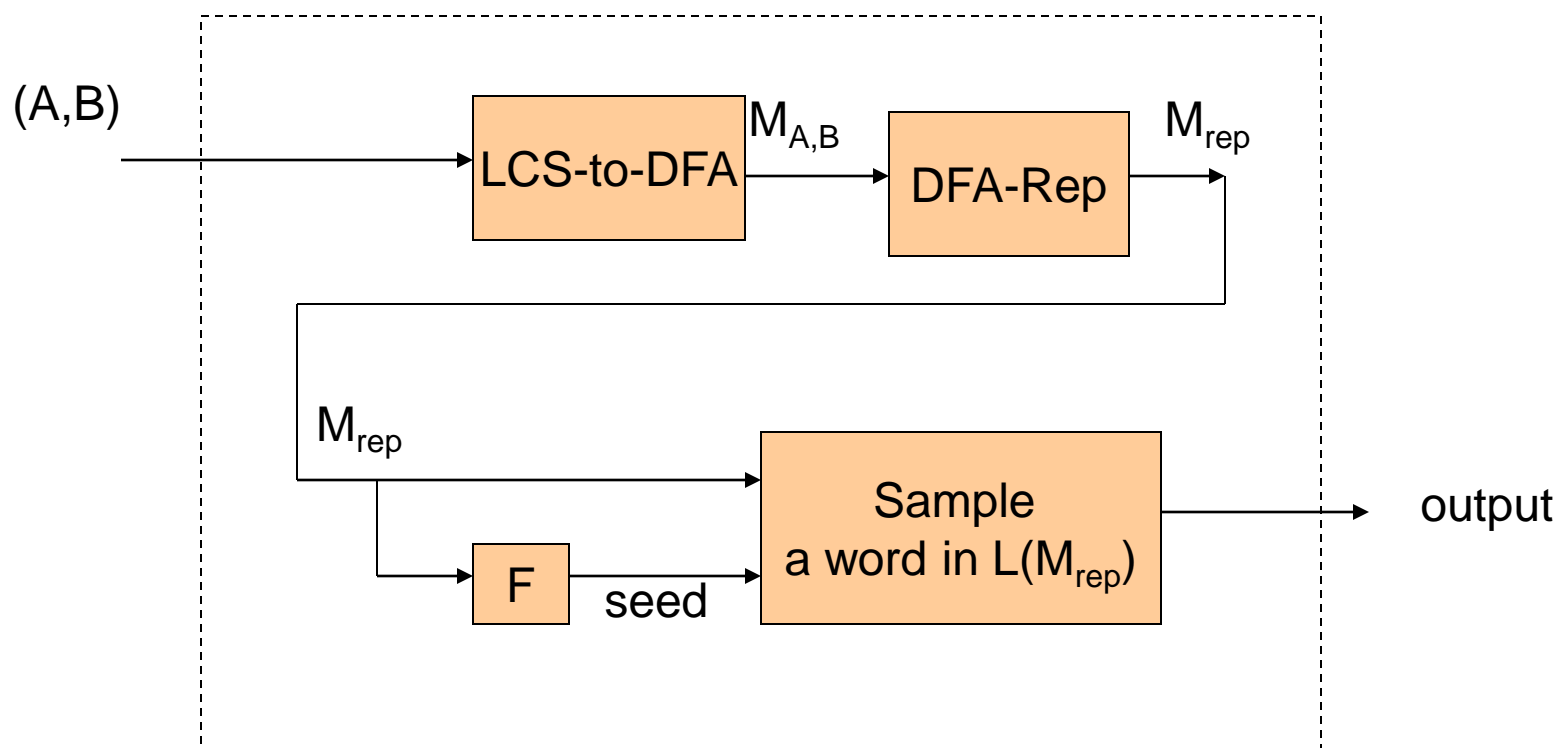
Step 2



- For M, M' where $L(M) = L(M')$
 - $\text{DFA-Rep}(M) = \text{DFA-Rep}(M')$
- $L(M_{A,B}) = L(M_{\text{rep}})$

- Generate a minimal-state DFA, M_{rep}
- Myhill-Nerode theorem:
 - M_{rep} is unique
- Runs $O(mn)$ time

Equivalence Protecting Algorithm for LCS String



Summary

- Private search algorithms for LCS
 - LCS string
 - LCS embed
- Matching best counting algorithms
 - $O(mn \log S)$ time
- Can be implemented in a variety settings
- Private search algorithms for
 - DFA represented functionalities
 - Problems with D.P. solutions

Open Problems

■ Privacy

- Alternative privacy definitions
 - The setting can effect privacy concerns
 - Take into account input distribution

■ Algorithms

- Better private search for LCS
 - Computationally indistinguishable from uniform
- Better counting for LCS