Lecture 5: Hardness for Sequence Problems under SETH and OVC

Thanks to Piotr Indyk and Arturs Backurs for some slides
Plan
Plan

• Define sequence problems:
  – (Discrete) Frechet Distance
  – Edit Distance and LCS
  – Dynamic Time Warping (DTW)
Plan

• Define sequence problems:
  – (Discrete) *Frechet* Distance
  – *Edit Distance* and *LCS*
  – Dynamic Time Warping (*DTW*)

• Birds eye view on the upper bounds
  – Dynamic programming, quadratic time
Plan

• Define sequence problems:
  – (Discrete) **Frechet** Distance
  – **Edit Distance** and **LCS**
  – Dynamic Time Warping (**DTW**)  

• Birds eye view on the upper bounds
  – Dynamic programming, quadratic time

• Show conditional quadratic lower bounds
  – Assuming SETH / OV, example: **LCS**
Walks on sequences

Given two sequences \(\{p_i\}\) and \(\{q_j\}\), a walk on them starts at \(p_1\) and \(q_1\). In each step it is in some position \((p_i, q_j)\) and can next:
Walks on sequences

Given two sequences \(\{p_i\}\) and \(\{q_j\}\), a walk on them starts at \(p_1\) and \(q_1\). In each step it is in some position \((p_i, q_j)\) and can next:
Walks on sequences

Given two sequences \( \{p_i\} \) and \( \{q_j\} \), a walk on them starts at \( p_1 \) and \( q_1 \). In each step it is in some position \( (p_i, q_j) \) and can next:

- go right only on \( p \) to \( (p_{i+1}, q_j) \)
Walks on sequences

Given two sequences \{p_i\} and \{q_j\}, a walk on them starts at \(p_1\) and \(q_1\). In each step it is in some position \((p_i, q_j)\) and can next:

- go right only on \(p\) to \((p_{i+1}, q_j)\)
Walks on sequences

Given two sequences \{p_i\} and \{q_j\}, a *walk* on them starts at \(p_1\) and \(q_1\). In each step it is in some position \((p_i, q_j)\) and can next:

- go right *only on p* to \((p_{i+1}, q_j)\)
Walks on sequences

Given two sequences \( \{p_i\} \) and \( \{q_j\} \), a \textit{walk} on them starts at \( p_1 \) and \( q_1 \). In each step it is in some position \( (p_i, q_j) \) and can next:

- go right \textit{only on p} to \( (p_{i+1}, q_j) \)
- go right \textit{only on q} to \( (p_i, q_{j+1}) \)
Walks on sequences

Given two sequences \{p_i\} and \{q_j\}, a \textit{walk} on them starts at \( p_1 \) and \( q_1 \). In each step it is in some position \((p_i, q_j)\) and can next:

- go right \textbf{only on} \( p \) to \((p_{i+1}, q_j)\)
- go right \textbf{only on} \( q \) to \((p_i, q_{j+1})\)
Walks on sequences

Given two sequences \( \{p_i\} \) and \( \{q_j\} \), a walk on them starts at \( p_1 \) and \( q_1 \). In each step it is in some position \( (p_i, q_j) \) and can next:

- go right only on \( p \) to \( (p_{i+1}, q_j) \)
- go right only on \( q \) to \( (p_i, q_{j+1}) \)
Walks on sequences

Given two sequences \( \{p_i\} \) and \( \{q_j\} \), a walk on them starts at \( p_1 \) and \( q_1 \). In each step it is in some position \((p_i, q_j)\) and can next:

- go right only on \( p \) to \((p_{i+1}, q_j)\)
- go right only on \( q \) to \((p_i, q_{j+1})\)
- go right on both to \((p_{i+1}, q_{j+1})\)
Walks on sequences

Given two sequences \( \{p_i\} \) and \( \{q_j\} \), a walk on them starts at \( p_1 \) and \( q_1 \). In each step it is in some position \( (p_i, q_j) \) and can next:

- go right only on \( p \) to \( (p_{i+1}, q_j) \)
- go right only on \( q \) to \( (p_i, q_{j+1}) \)
- go right on both to \( (p_{i+1}, q_{j+1}) \)
Walks on sequences

Given two sequences \( \{p_i\} \) and \( \{q_j\} \), a *walk* on them starts at \( p_1 \) and \( q_1 \). In each step it is in some position \((p_i,q_j)\) and can next:

- go right only on \( p \) to \((p_{i+1}, q_j)\)
- go right only on \( q \) to \((p_i, q_{j+1})\)
- go right on both to \((p_{i+1}, q_{j+1})\)

Sequence walk problems optimize, over all such walks, some measure depending on the distances between \( p_i \) and \( q_j \) over all steps \((p_i,q_j)\) of the walk.
(Discrete) Frechet Distance  [Alt-Godau’95]

• “Dog walking distance”
  – Smallest length leash that enables dog-walking along two routes
(Discrete) Frechet Distance  [Alt-Godau’95]

• “Dog walking distance”
  – Smallest length leash that enables dog-walking along two routes
(Discrete) Frechet Distance  [Alt-Godau’95]

- "Dog walking distance"
  - Smallest length leash that enables dog-walking along two routes
(Discrete) Frechet Distance  [Alt-Godau’95]

• “Dog walking distance”
  – Smallest length leash that enables dog-walking along two routes

• Definition:
  – Let $F = \text{set of monotone functions } [0,1] \rightarrow [0,1]$
  – For two curves $P, Q: [0,1] \rightarrow \mathbb{R}^2$
  
  \[ D_{Fr}(P,Q) = \min_{f,g \in F} \max_{t \in [0,1]} \| P(f(t)) - Q(g(t)) \| \]
(Discrete) Frechet Distance  [Alt-Godau’95]

• “Dog walking distance”
  – Smallest length leash that enables dog-walking along two routes

• Definition:
  – Let \( F = \text{set of monotone functions } [0,1] \rightarrow [0,1] \)
  – For two curves \( P, Q: [0,1] \rightarrow \mathbb{R}^2 \):
    \[
    D_{Fr}(P, Q) = \min_{f, g \in F} \max_{t \in [0,1]} ||P(f(t)) - Q(g(t))||
    \]

• Discrete version:
  – \( F = \{ f: [0,1] \rightarrow \{1...n\} , \text{ nondecreasing}\} \),
  – \( P, Q: \{1...n\} \rightarrow \mathbb{R}^2 \): Curves are sequences of points in the plane
(Discrete) Frechet Distance  [Alt-Godau’95]

• “Dog walking distance”
  – Smallest length leash that enables dog-walking along two routes

• Definition:
  – Let \( F = \text{set of monotone functions } [0,1] \rightarrow [0,1] \)
  – For two curves \( P, Q: [0,1] \rightarrow \mathbb{R}^2 \):
    \[
    D_{Fr}(P, Q) = \min_{f, g \in F} \max_{t \in [0,1]} ||P(f(t)) - Q(g(t))||
    \]

• Discrete version:
  – \( F = \{ f: [0,1] \rightarrow \{1\ldots n\}, \text{ nondecreasing} \} \),
  – \( P, Q: \{1\ldots n\} \rightarrow \mathbb{R}^2 \): Curves are sequences of points in the plane

Find a walk along \( P \) and \( Q \) that minimizes the max distance over all steps.
Frechet Distance: Algorithm
Frechet Distance: Algorithm

• Discrete version:
  – Let $F = \{ f: [0,1] \rightarrow \{1...n\}, \text{nondecreasing}\}$, mapping time to position,
  – For two sequences of points, $P, Q: \{1...n\} \rightarrow \mathbb{R}^2$:
    \[ D_{Fr}(P, Q) = \min_{f, g \in F} \max_{t \in [0,1]} ||P(f(t)) - Q(g(t))|| \]
Frechet Distance: Algorithm

• **Discrete version:**
  - Let $F = \{ f: [0,1] \to \{1...n\}, \text{nondecreasing} \}$, mapping *time* to *position*,
  - For two sequences of points, $P, Q$: $\{1...n\} \to \mathbb{R}^2$:
    \[ D_{Fr}(P,Q) = \min_{f,g \in F} \max_{t \in [0,1]} | |P(f(t)) - Q(g(t))| | \]

• **Dynamic programming:**
  - $A[i, j] = \text{distance between curves } P(1)...P(i) \text{ and } Q(1)...Q(j)$
  - $A[i, j]=\max[ | |P(i)-Q(j)| | , \min (A[i-1, j-1], A[i, j-1], A[i-1, j]) ]$
Frechet Distance: Algorithm

• Discrete version:
  – Let $F = \{ f: [0,1] \rightarrow \{1...n\}, \text{nondecreasing} \}$, mapping time to position,
  – For two sequences of points, $P,Q: \{1...n\} \rightarrow \mathbb{R}^2$:
    \[
    D_{Fr}(P,Q) = \min_{f,g \in F} \max_{t \in [0,1]} \| P(f(t)) - Q(g(t)) \|
    \]

• Dynamic programming:
  – $A[i, j] = \text{distance between curves } P(1)...P(i) \text{ and } Q(1) ...Q(j)$
  – $A[i, j] = \max[ \| P(i)-Q(j) \|, \min (A[i-1, j-1], A[i, j-1], A[i-1, j]) ]$

• Time: $O(n^2)$
Frechet Distance: Algorithm

• Discrete version:
  – Let $F = \{ f: [0,1] \to \{1...n\}, \text{nondecreasing } \}$, mapping \textit{time} to \textit{position},
  – For two sequences of points, $P, Q: \{1...n\} \to \mathbb{R}^2$:
    $$D_{Fr}(P, Q) = \min_{f, g \in F} \max_{t \in [0,1]} | |P(f(t)) - Q(g(t))||$$

• Dynamic programming:
  – $A[i, j] = \text{distance between curves } P(1)...P(i) \text{ and } Q(1) ...Q(j)$
  – $A[i, j] = \max[ | |P(i)-Q(j)| |, \min( A[i-1, j-1], A[i, j-1], A[i-1, j]) ]$
  – \text{Time: } O(n^2)$

• Can be improved to $O(n^2 \log \log n / \log n)$ [Agarwal-Avraham-Kaplan-Sharir’12] (also [Buchin-Buchin-Meulemans-Mulzer’14])
Frechet Distance: Algorithm

• Discrete version:
  – Let $F = \{ f: [0,1] \to \{1...n\}, \text{nondecreasing} \}$, mapping time to position,
  – For two sequences of points, $P,Q: \{1...n\} \to \mathbb{R}^2$:
    $$D_{Fr}(P,Q) = \min_{f,g \in F} \max_{t \in [0,1]} \|P(f(t)) - Q(g(t))\|$$

• Dynamic programming:
  – $A[i, j] = \text{distance between curves } P(1)...P(i) \text{ and } Q(1)...Q(j)$
  – $A[i, j] = \max[\|P(i)-Q(j)\|, \min(A[i-1, j-1], A[i, j-1], A[i-1, j])]$

• Time: $O(n^2)$

• Can be improved to $O(n^2 \log \log n / \log n)$ [Agarwal-Avraham-Kaplan-Sharir’12] (also [Buchin-Buchin-Meulemans-Mulzer’14])

• Many algorithms for special cases and variants
Dynamic Time Warping
Dynamic Time Warping

- **Definition:**
  - x, y: two sequences of points of length n
  - A[i, j] = dist(x_i, y_j) + min(A[i-1, j], A[i-1, j-1], A[i, j-1])
  - DTW(x, y) = A[n, n]

*Find a walk along x and y that minimizes the sum of distances at each step.*
Dynamic Time Warping

• Definition:
  – $x, y$: two sequences of points of length $n$
  – $A[i, j]$ = $\text{dist}(x_i, y_j) + \min(A[i-1,j], A[i-1,j-1], A[i,j-1])$
  – $\text{DTW}(x,y) = A[n,n]$

  **Find a walk along $x$ and $y$ that minimizes the sum of distances at each step.**

• Speech processing and other applications
Dynamic Time Warping

• Definition:
  – $x, y$: two sequences of points of length $n$
  – $A[i, j]=\text{dist}(x_i, y_j)+\min(A[i-1,j], A[i-1,j-1], A[i,j-1])$
  – $\text{DTW}(x,y)=A[n,n]$
    
    **Find a walk along $x$ and $y$ that minimizes the sum of distances at each step.**

• Speech processing and other applications

• A simple $O(n^2)$ time dynamic programming algorithm
Longest Common Subsequence (LCS)

• Definition:
  – two sequences s and t of letters, length n
  – find a subsequence of both s and t of max length
• Example: LCS(meaning, matching) = maing
Longest Common Subsequence (LCS)

• Definition:
  – two sequences s and t of letters, length n
  – find a subsequence of both s and t of max length

• Example: LCS(meaning, matching) = maing

• Simple $O(n^2)$ time algorithm:
Longest Common Subsequence (LCS)

• Definition:
  – two sequences s and t of letters, length n
  – find a subsequence of both s and t of max length

• Example: LCS(meaning, matching) = maing

• Simple $O(n^2)$ time algorithm:

\[
A[i,j] = \begin{cases} 
\max\{A[i-1, j], A[i, j-1], 1+A[i-1, j-1]\} & \text{if } s[i]=t[i] \\
\max\{A[i-1, j], A[i, j-1]\} & \text{otherwise.}
\end{cases}
\]
Longest Common Subsequence (LCS)

• Definition:
  – two sequences s and t of letters, length n
  – find a subsequence of both s and t of max length

• Example: LCS(meaing, matching) = maing

• Simple $O(n^2)$ time algorithm:

$$A[i,j] = \begin{cases} \max\{A[i-1, j], A[i, j-1], 1+A[i-1, j-1]\} & \text{if } s[i]=t[i] \\ \max\{A[i-1, j], A[i, j-1]\} & \text{otherwise.} \end{cases}$$

Best algorithm: $O(n^2/\log n)$ [Masek-Paterson’80]
Edit distance
(a.k.a. Levenshtein distance)

• Definition:
  – \( x, y \) – two sequences of symbols of length \( n \)
Edit distance
(a.k.a. Levenshtein distance)

• Definition:
  – \(x, y\) – two sequences of symbols of length \(n\)
  – \(\text{edit}(x, y)\) = the minimum number of symbol \textit{insertions}, \textit{deletions} or \textit{substitutions} needed to transform \(x\) into \(y\)
Edit distance
(a.k.a. Levenshtein distance)

• Definition:
  – $x,y$ – two sequences of symbols of length $n$
  – $\text{edit}(x,y) =$ the minimum number of symbol insertions, deletions or substitutions needed to transform $x$ into $y$

• Example: $\text{edit}(\text{meaning},\text{matching}) = 4$
Edit distance
(a.k.a. Levenshtein distance)

• Definition:
  – \( x, y \) – two sequences of symbols of length \( n \)
  – \( \text{edit}(x, y) \) – the minimum number of symbol insertions, deletions or substitutions needed to transform \( x \) into \( y \)

• Example: \( \text{edit}(\text{meaning}, \text{matching}) = 4 \)
Edit distance
(a.k.a. Levenshtein distance)

• Definition:
  – $x,y$ – two sequences of symbols of length $n$
  – $\text{edit}(x,y)$ = the minimum number of symbol insertions, deletions or substitutions needed to transform $x$ into $y$

• Example: $\text{edit}(\text{meaning}, \text{matching}) = 4$

meaning $\xrightarrow{\text{insert a}}$ meaning
Edit distance
(a.k.a. Levenshtein distance)

• Definition:
  – \(x, y\) – two sequences of symbols of length \(n\)
  – \(\text{edit}(x, y)\) = the minimum number of symbol insertions, deletions or substitutions needed to transform \(x\) into \(y\)

• Example: \(\text{edit}(\text{meaning}, \text{matching}) = 4\)

  \[
  \text{meaning} \quad \xrightarrow{\text{insert a}} \quad \text{maeaning} \quad \xrightarrow{e \rightarrow t} \quad \text{mating}
  \]
Edit distance
(a.k.a. Levenshtein distance)

• Definition:
  – \( x, y \) – two sequences of symbols of length \( n \)
  – \( \text{edit}(x, y) \) – the minimum number of symbol insertions, deletions or substitutions needed to transform \( x \) into \( y \)

• Example: \( \text{edit}(\text{meaning}, \text{matching}) = 4 \)

\[ \text{meaning} \xrightarrow{\text{insert } a} \text{maeaning} \xrightarrow{e \rightarrow t} \text{mataning} \]
\[ \text{mataning} \xrightarrow{a \rightarrow c} \text{matching} \]
Edit distance  
(a.k.a. Levenshtein distance)

• Definition:
  – \(x, y\) – two sequences of symbols of length \(n\)
  – \(edit(x, y)\) = the minimum number of symbol insertions, deletions or substitutions needed to transform \(x\) into \(y\)

• Example: \(edit(\text{meaning}, \text{matching}) = 4\)

\[
\begin{align*}
\text{meaning} & \rightarrow \text{maeaning} \rightarrow \text{mataning} \rightarrow \text{matcning} \\
& \quad \downarrow \quad \quad \downarrow \quad \quad \downarrow \\
& \quad a \rightarrow c \quad e \rightarrow t \\
& \quad \quad \quad \quad \downarrow \\
& \quad \quad \quad \quad \text{matcning} \rightarrow \text{matching}
\end{align*}
\]
Computing edit distance
Computing edit distance

• A simple $O(n^2)$ time dynamic programming algorithm [Wagner-Fischer’74]
Computing edit distance

• A simple $O(n^2)$ time dynamic programming algorithm [Wagner-Fischer’74]

• Can be improved to $O(n^2/\log n)$ [Masek-Paterson’80]
Computing edit distance

• A simple $O(n^2)$ time dynamic programming algorithm [Wagner-Fischer’74]

• Can be improved to $O(n^2/\log n)$ [Masek-Paterson’80]

• Better algorithms for special cases: [U83, LV85, M86, GG88, GP89, UW90, CL90, CH98, LMS98, U85, CL92, N99, CPSV00, MS00, CM02, BCF08, AK08, AKO10…]
Computing edit distance

• A simple $O(n^2)$ time dynamic programming algorithm [Wagner-Fischer’74]

• Can be improved to $O(n^2 / \log n)$ [Masek-Paterson’80]

• Better algorithms for special cases: [U83, LV85, M86, GG88, GP89, UW90, CL90, CH98, LMS98, U85, CL92, N99, CPSV00, MS00, CM02, BCF08, AK08, AKO10...]

• Approximation algorithms: $O(1)$ –approx in $O(n^{2-\varepsilon})$ time [Chakraborty-Das-Goldenberg-Koucky-Saks’18],

$O(f(\varepsilon))$ –approx in $O(n^{1+\varepsilon})$ time [Andoni-Nowatzki’20]
What do these problems have in common?
What do these problems have in common?

- Widely used metrics
What do these problems have in common?

• Widely used metrics
• Simple dynamic-programming algorithms with (essentially) quadratic running time
What do these problems have in common?

- Widely used metrics
- Simple dynamic-programming algorithms with (essentially) quadratic running time
- We have no idea if/how we can do any better
What do these problems have in common?

• Widely used metrics
• Simple dynamic-programming algorithms with (essentially) quadratic running time
• We have no idea if/how we can do any better

• Plausible explanation:
  – 3SUM-hard? People tried for years...
What do these problems have in common?

• Widely used metrics
• Simple dynamic-programming algorithms with (essentially) quadratic running time
• We have no idea if/how we can do any better

• Plausible explanation:
  – 3SUM-hard? People tried for years...
  – hard under OVH and SETH?
Plan

- Define sequence problems:
  - (Discrete) Frechet Distance
  - Edit Distance and LCS
  - Dynamic Time Warping (DTW)
- Birds eye view on the upper bounds
  - Dynamic programming, quadratic time
- Show conditional quadratic lower bounds
  - Assuming SETH / OVH
  - Basic approach
  - Hardness for LCS
Reminder: Orthogonal Vectors Hypothesis (OVH)
Reminder: Orthogonal Vectors Hypothesis (OVH)

• Orthogonal Vectors Problem (OV). Given a set of vectors $S \subseteq \{0, 1\}^d$, $d = \omega(\log n)$, $|S| = n$, are there $a, b \in S$ s. t. $\sum_{i=1}^{d} a_i b_i = 0$ ?

  – Can be solved trivially in $O(n^2d)$ time
  – Best known algorithm runs in $n^{2-1/O(\log c(n))}$ time, where $d = c(n) \cdot \log n$ [Abboud-Williams-Yu’15]
Reminder: Orthogonal Vectors Hypothesis (OVH)

• **Orthogonal Vectors Problem (OV).** Given a set of vectors $S \subseteq \{0, 1\}^d$, $d = \omega(\log n)$, $|S| = n$, are there $a, b \in S$ s. t. $\sum_{i=1}^{d} a_i b_i = 0$ ?
  
  – Can be solved trivially in $O(n^2d)$ time
  
  – Best known algorithm runs in $n^{2-1/O(\log c(n))}$ time, where $d = c(n) \cdot \log n$ [Abboud-Williams-Yu’15]

• **OV Hypothesis (implied by SETH):**
Reminder: Orthogonal Vectors Hypothesis (OVH)

• **Orthogonal Vectors Problem (OV).** Given a set of vectors $S \subseteq \{0, 1\}^d$, $d = \omega(\log n)$, $|S| = n$, are there $a, b \in S$ s. t. $\sum_{i=1}^{d} a_i b_i = 0$?
  - Can be solved trivially in $O(n^2d)$ time
  - Best known algorithm runs in $n^{2-1/O(\log c(n))}$ time, where $d = c(n) \cdot \log n$ [Abboud-Williams-Yu’15]

• **OV Hypothesis (implied by SETH):**
  OV can’t be solved in $n^{2-\varepsilon \cdot d^{O(1)}}$ time for any $\varepsilon > 0$. 
Quadratic hardness under OVC

Theorem*: No $n^{2-\Omega(1)}$ time algorithm for EDIT, DTW, Frechet distances or LCS unless OVC fails [Bringmann’14; Backurs-Indyk’15; Abboud-Backurs-VW’15; Bringmann-Kunnemann’15]

*See also [Abboud-V. Williams-Weimann’14]
Quadratic hardness under OVC

Theorem*: No $n^{2-\Omega(1)}$ time algorithm for EDIT, DTW, Frechet distances or LCS unless OVC fails [Bringmann’14; Backurs-Indyk’15; Abboud-Backurs-VW’15; Bringmann-Kunnemann’15]

• Approach: reduce OV to distance computation:

*See also [Abboud-V. Williams-Weimann’14]
Quadratic hardness under OVC

Theorem*: No $n^{2-\Omega(1)}$ time algorithm for EDIT, DTW, Frechet distances or LCS unless OVC fails [Bringmann’14; Backurs-Indyk’15; Abboud-Backurs-VW’15; Bringmann-Kunnemann’15]

• Approach: reduce OV to distance computation:
  – $S \subseteq \{0,1\}^d \rightarrow$ sequence $x$, $|x| \leq n \cdot d^{O(1)}$
  – $S \subseteq \{0,1\}^d \rightarrow$ sequence $y$, $|y| \leq n \cdot d^{O(1)}$

*See also [Abboud-V. Williams-Weimann’14]
Quadratic hardness under OVC

Theorem*: No $n^{2-\Omega(1)}$ time algorithm for EDIT, DTW, Frechet distances or LCS unless OVC fails [Bringmann’14; Backurs-Indyk’15; Abboud-Backurs-VW’15; Bringmann-Kunnemann’15]

• Approach: reduce OV to distance computation:
  – $S \subseteq \{0,1\}^d \rightarrow$ sequence $x$, $|x| \leq n \cdot d^{O(1)}$
  – $S \subseteq \{0,1\}^d \rightarrow$ sequence $y$, $|y| \leq n \cdot d^{O(1)}$
  – $\text{distance}(x,y) = \text{small}$ if exists $a, b \in S$ with $\Sigma_i a_i b_i = 0$
  – $\text{distance}(x,y) = \text{large}$, otherwise
  – The construction time is $n \cdot d^{O(1)}$
  – Gadgets for coordinates and vectors

*See also [Abboud-V. Williams-Weimann’14]
Quadratic hardness under OVC

Theorem*: No $n^{2-O(1)}$ time algorithm for EDIT, DTW, Frechet distances or LCS unless OVC fails [Bringmann’14; Backurs-Indyk’15; Abboud-Backurs-VW’15; Bringmann-Kunnemann’15]

• Approach: reduce OV to distance computation:
  – $S \subseteq \{0,1\}^d \rightarrow$ sequence $x$, $|x| \leq n \cdot d^{O(1)}$
  – $S \subseteq \{0,1\}^d \rightarrow$ sequence $y$, $|y| \leq n \cdot d^{O(1)}$
  – $\text{distance}(x,y) = \text{small}$ if exists $a, b \in S$ with $\sum_i a_i b_i = 0$
  – $\text{distance}(x,y) = \text{large}$, otherwise
  – The construction time is $n \cdot d^{O(1)}$
  – Gadgets for coordinates and vectors

*See also [Abboud-V. Williams-Weimann’14]
Hardness for LCS

I will present the ideas behind the proof from [Abboud-Backurs-VW’15].

Full construction. NO full proof.

[Bringmann-Kunnemann’15] obtained an independent proof.
OV to LCS

Given vectors \( \{s_1, \ldots, s_n\}, s_i \in \{0,1\}^d \forall i, \) OV is

\[
\bigvee_{i,j \in [n]} \bigwedge_{k \in [d]} (\neg s_i[k] \lor \neg s_j[k]).
\]
OV to LCS

Given vectors \( \{s_1, ..., s_n\}, s_i \in \{0,1\}^d \) \( \forall i \), \( \text{OV} \) is

\[
\bigvee_{i,j \in [n]} \bigwedge_{k \in [d]} (\neg s_i[k] \lor \neg s_j[k]).
\]

**Coordinate gadgets** \( c, e \) taking bits to short sequences s.t.

\[
LCS(c(x), e(y)) = 0 \text{ if } x = y = 1, \\
LCS(c(x), e(y)) = 1 \text{ if } x \cdot y = 0.
\]
OV to LCS

Given vectors \( \{s_1, \ldots, s_n\}, s_i \in \{0,1\}^d \ \forall i \), OV is

\[
\bigvee_{i,j \in [n]} \bigwedge_{k \in [d]} \left( \neg s_i[k] \lor \neg s_j[k] \right).
\]

**Vector gadgets** \( f, g \) taking bit vectors to short sequences s.t. for some \( T \)

\[
\text{LCS} \left( f(s_i), g(s_j) \right) = T + 1 \text{ if } s_i \cdot s_j = 0,
\]
\[
\text{LCS} \left( f(s_i), g(s_j) \right) = T \text{ if } s_i \cdot s_j \neq 0.
\]

**Coordinate gadgets** \( c, e \) taking bits to short sequences s.t.

\[
\text{LCS}(c(x), e(y)) = 0 \text{ if } x = y = 1,
\]
\[
\text{LCS}(c(x), e(y)) = 1 \text{ if } x \cdot y = 0.
\]
OV to LCS

Given vectors \( \{s_1, \ldots, s_n\}, s_i \in \{0,1\}^d \forall i \), OV is

\[
\bigvee_{i,j \in [n]} \bigwedge_{k \in [d]} (\neg s_i[k] \lor \neg s_j[k]).
\]

**Coordinate gadgets** \( c, e \) taking bits to short sequences s.t.

\[
LCS(c(x), e(y)) = 0 \text{ if } x = y = 1,
\]

\[
LCS(c(x), e(y)) = 1 \text{ if } x \cdot y = 0.
\]

**Outer OR gadgets** \( x, y \) taking sets of bit vectors \( \{s_1, \ldots, s_n\} \), to short sequences s.t. for some \( Q \)

\[
LCS(x, y) = Q \text{ if } \forall i, j: s_i \cdot s_j \neq 0,
\]

\[
LCS(x, y) \geq Q + 1 \text{ if } \exists i, j: s_i \cdot s_j = 0.
\]

**Vector gadgets** \( f, g \) taking bit vectors to short sequences s.t. for some \( T \)

\[
LCS\left(f\left(s_i\right), g\left(s_j\right)\right) = T + 1 \text{ if } s_i \cdot s_j = 0,
\]

\[
LCS\left(f\left(s_i\right), g\left(s_j\right)\right) = T \text{ if } s_i \cdot s_j \neq 0.
\]
\[ \bigvee_{i,j \in [n]} \land_{c \in [d]} (\neg s_i[c] \lor \neg s_j[c]) \]

Encoding the outer Boolean OR for OV to LCS
Let $S = \{s_1, s_2, \ldots, s_n\}$ be the vectors from OV instance.

Suppose we have $s_i \rightarrow$ gadget sequences $f(s_i)$ and $g(s_i)$.

$LCS(f(s_i), g(s_j)) = \beta$ if $s_i \cdot s_j \neq 0$, $LCS(f(s_i), g(s_j)) = \beta + 1$ otherwise.

Encoding the outer Boolean OR for OV to LCS:

$$\bigvee_{i,j \in [n]} \bigwedge_{c \in [d]} (\neg s_i[c] \lor \neg s_j[c])$$
Let $S = \{s_1, s_2, \ldots, s_n\}$ be the vectors from OV instance.

Suppose we have $s_i \rightarrow$ gadget sequences $f(s_i)$ and $g(s_i)$.

$LCS(f(s_i), g(s_j)) = \beta$ if $s_i \cdot s_j \neq 0$, $LCS(f(s_i), g(s_j)) = \beta + 1$ otherwise.

Encoding the outer Boolean OR for OV to LCS.

$$\bigvee_{i,j \in [n]} \bigwedge_{c \in [d]} (\neg s_i[c] \lor \neg s_j[c])$$

Want to create sequences $x$ and $y$ so that $LCS(x,y)$ is Large if there is an OV pair and $LCS(x,y)$ is Small otherwise.
Let $S = \{s_1, s_2, \ldots, s_n\}$ be the vectors from OV instance.

Suppose we have $s_i \rightarrow$ gadget sequences $f(s_i)$ and $g(s_i)$.

$LCS(f(s_i), g(s_j)) = \beta$ if $s_i \cdot s_j \neq 0$, $LCS(f(s_i), g(s_j)) = \beta + 1$ otherwise.

$s_0$ – vector of all 1s (no vector orthog. to $s_0$)

Want to create sequences $x$ and $y$ so that $LCS(x, y)$ is Large if there is an OV pair and $LCS(x, y)$ is Small otherwise.
\[ \lor_{i,j \in [n]} (\neg s_i[c] \lor \neg s_j[c]) \]

Encoding the outer Boolean OR for OV to LCS

- Let \( S = \{s_1, s_2, \ldots, s_n\} \) be the vectors from OV instance
- Suppose we have \( s_i \rightarrow \text{gadget} \) sequences \( f(s_i) \) and \( g(s_i) \)
  \[ \text{LCS}(f(s_i), g(s_j)) = \beta \text{ if } s_i \cdot s_j \neq 0, \text{ LCS}(f(s_i), g(s_j)) = \beta + 1 \text{ otherwise.} \]
- \( s_0 \) – vector of all 1s (no vector orthog. to \( s_0 \))

**Attempt 1:**
\[ x = f(s_1) f(s_2) \ldots f(s_i) \ldots f(s_n) \]
\[ y = (g(s_0))^{n-1} g(s_1) g(s_2) \ldots g(s_j) \ldots g(s_n) (g(s_0))^{n-1} \]

Want to create sequences \( x \) and \( y \) so that LCS(\( x, y \)) is Large if there is an OV pair and LCS(\( x, y \)) is Small otherwise.
\[ \bigvee_{i,j \in [n]} \land_{c \in [d]} (\neg s_i[c] \lor \neg s_j[c]) \]

- Let \( S = \{s_1, s_2, \ldots, s_n\} \) be the vectors from OV instance.
- Suppose we have \( s_i \rightarrow \text{gadget} \) sequences \( f(s_i) \) and \( g(s_i) \).
  - \( \text{LCS}(f(s_i), g(s_j)) = \beta \) if \( s_i \cdot s_j \neq 0 \), \( \text{LCS}(f(s_i), g(s_j)) = \beta + 1 \) otherwise.
- \( s_0 \) – vector of all 1s (no vector orthog. to \( s_0 \)).

**Attempt 1:**
\[ x = f(s_1) f(s_2) \ldots f(s_i) \ldots f(s_n) \]
\[ y = (g(s_0))^{n-1} g(s_1) g(s_2) \ldots g(s_j) \ldots g(s_n) (g(s_0))^{n-1} \]

**Idea:** Imagine gadgets are letters.
If no OV, LCS length is \( n \beta \); If \( s_i \cdot s_j = 0 \) can align \( f(s_i) \) and \( g(s_j) \) and all other \( f(s_k) \) with \( g(s_0) \) to get LCS length \( \geq (n-1) \beta + (\beta+1) > n \beta \).
Let $S = \{s_1, s_2, ..., s_n\}$ be the vectors from OV instance.

Suppose we have $s_i \rightarrow$ gadget sequences $f(s_i)$ and $g(s_i)$

$LCS(f(s_i), g(s_j)) = \beta$ if $s_i \cdot s_j \neq 0$, $LCS(f(s_i), g(s_j)) = \beta + 1$ otherwise.

$s_0$ – vector of all 1s (no vector orthog. to $s_0$)

**Attempt 1:**

$x = f(s_1) \cdot f(s_2) \cdots f(s_i) \cdots f(s_n)$

$y = (g(s_0))^{n-1} \cdot g(s_1) \cdot g(s_2) \cdots g(s_j) \cdots g(s_n) \cdot (g(s_0))^{n-1}$

**Idea:** Imagine gadgets are letters.

If no OV, LCS length is $n \beta$; If $s_i \cdot s_j = 0$ can align $f(s_i)$ and $g(s_j)$ and all other $f(s_k)$ with $g(s_0)$ to get LCS length $\geq (n-1) \beta + (\beta+1) > n \beta$. 

Encoding the outer Boolean OR for OV to LCS

\[ \bigwedge_{c \in [d]} (\neg s_i[c] \lor \neg s_j[c]) \\]
Let $S = \{s_1, s_2, \ldots, s_n\}$ be the vectors from OV instance.

Suppose we have $s_i \rightarrow$ gadget sequences $f(s_i)$ and $g(s_i)$.

$LCS(f(s_i), g(s_j)) = \beta$ if $s_i \cdot s_j \neq 0$, $LCS(f(s_i), g(s_j)) = \beta + 1$ otherwise.

$s_0$ – vector of all 1s (no vector orthog. to $s_0$)

**Attempt 1:**

$x = f(s_1) f(s_2) \ldots f(s_i) \ldots f(s_n)$

$y = (g(s_0))^{n-1} g(s_1) g(s_2) \ldots g(s_j) \ldots g(s_n) (g(s_0))^{n-1}$

**Idea:** Imagine gadgets are letters.

If no OV, LCS length is $n \beta$; If $s_i \cdot s_j = 0$ can align $f(s_i)$ and $g(s_j)$ and all other $f(s_k)$ with $g(s_0)$ to get LCS length $\geq (n-1) \beta + (\beta+1) > n \beta$.

**Problem:** Opt LCS might not align entire gadgets!

Encoding the outer Boolean OR for OV to LCS
Let \( S = \{s_1, s_2, \ldots, s_n\} \) be the vectors from OV instance.

Suppose we have \( s_i \rightarrow \text{gadget} \) sequences \( f(s_i) \) and \( g(s_i) \).

\[
\text{LCS}(f(s_i), g(s_j)) = \beta \text{ if } s_i \cdot s_j \neq 0, \text{ LCS}(f(s_i), g(s_j)) = \beta + 1 \text{ otherwise.}
\]

- \( s_0 \) – vector of all 1s (no vector orthog. to \( s_0 \))

**Attempt 1:**

\[
x = f(s_1) f(s_2) \ldots f(s_i) \ldots f(s_n)
\]

\[
y = (g(s_0))^{n-1} g(s_1) g(s_2) \ldots g(s_j) \ldots g(s_n) (g(s_0))^{n-1}
\]

**Idea:** Imagine gadgets are letters.

If no OV, LCS length is \( n \beta \); If \( s_i \cdot s_j = 0 \) can align \( f(s_i) \) and \( g(s_j) \) and all other \( f(s_k) \) with \( g(s_0) \) to get LCS length \( \geq (n-1) \beta + (\beta+1) > n \beta \).

**Problem:** Opt LCS might not align entire gadgets!
Let $S = \{s_1, s_2, \ldots, s_n\}$ be the vectors
Each $s_i$ → gadget sequences $f(s_i)$ and $g(s_i)$
$LCS(f(s_i), g(s_j)) = \beta$ if $s_i \cdot s_j \neq 0$, $LCS(f(s_i), g(s_j)) = \beta + 1$ otherwise.
$s_0$ – vector of all 1s (no vector orthog. to $s_0$)
Let $S = \{s_1, s_2, \ldots, s_n\}$ be the vectors.
Each $s_i \rightarrow$ gadget sequences $f(s_i)$ and $g(s_i)$.

$LCS(f(s_i), g(s_j)) = \beta$ if $s_i \cdot s_j \neq 0$, $LCS(f(s_i), g(s_j)) = \beta + 1$ otherwise.

$s_0$ – vector of all 1s (no vector orthog. to $s_0$).

**Attempt 2:**

$x = Qf(s_1)R \ Qf(s_2)R \ldots \ Qf(s_n) \ R$

$y = (Qg(s_0) \ R)^{n-1} \ Qg(s_1)R \ Qg(s_2)R \ldots \ Qg(s_n)R \ (Qg(s_0) \ R)^{n-1}$
Let $S = \{s_1, s_2, \ldots, s_n\}$ be the vectors
Each $s_i \rightarrow$ gadget sequences $f(s_i)$ and $g(s_i)$

$LCS(f(s_i), g(s_j)) = \beta$ if $s_i \cdot s_j \neq 0$, $LCS(f(s_i), g(s_j)) = \beta + 1$ otherwise.

$s_0$ – vector of all 1s (no vector orthog. to $s_0$)

**Attempt 2:**

$x = Q f(s_1) R Q f(s_2) R \ldots Q f(s_n) R$

$y = (Q g(s_0) R)^{n-1} Q g(s_1) R Q g(s_2) R \ldots Q g(s_n) R (Q g(s_0) R)^{n-1}$

**Lemma:** If a 0 (or 1) is matched, its entire $0^q$ (or $1^q$) block is matched.
Let $S = \{s_1, s_2, \ldots, s_n\}$ be the vectors.
Each $s_i \rightarrow$ gadget sequences $f(s_i)$ and $g(s_i)$.
$LCS(f(s_i), g(s_j)) = \beta$ if $s_i \cdot s_j \neq 0$, $LCS(f(s_i), g(s_j)) = \beta + 1$ otherwise.
$s_0$ – vector of all 1s (no vector orthog. to $s_0$)

**Attempt 2:**

$x = Q \ f(s_1) R \ Q \ f(s_2) R \ldots \ Q \ f(s_n) R$

$y = (Qg(s_0) R)^{n-1} \ Qg(s_1) R \ Qg(s_2) R \ldots \ Qg(s_n) R \ (Qg(s_0) R)^{n-1}$

**Lemma:** If a 0 (or 1) is matched, its entire $0^q$ (or $1^q$) block is matched.

**Idea:** Pick $q$ big so all $Q$s and $R$s of $x$ must be matched in an LCS.
Let $S = \{s_1, s_2, \ldots, s_n\}$ be the vectors
Each $s_i \rightarrow$ gadget sequences $f(s_i)$ and $g(s_i)$

$LCS(f(s_i), g(s_j)) = \beta$ if $s_i \cdot s_j \neq 0$, $LCS(f(s_i), g(s_j)) = \beta + 1$ otherwise.

$s_0$ – vector of all 1s (no vector orthog. to $s_0$)

**Attempt 2:**

$x = Q f(s_1) R Q f(s_2) R \ldots Q f(s_n) R$

$y = (Q g(s_0) R)^{n-1} Q g(s_1) R Q g(s_2) R \ldots Q g(s_n) R (Q g(s_0) R)^{n-1}$

**Lemma:** If a 0 (or 1) is matched, its entire $0^q$ (or $1^q$) block is matched.

**Idea:** Pick $q$ big so all $Q$s and $R$s of $x$ must be matched in an LCS.

Now no $g(s_k)$ is aligned with two different $f(s_i)$ and $f(s_j)$.
Let $S = \{s_1, s_2, \ldots, s_n\}$ be the vectors.

Each $s_i \rightarrow$ gadget sequences $f(s_i)$ and $g(s_i)$

$LCS(f(s_i), g(s_j)) = \beta$ if $s_i \cdot s_j \neq 0$, $LCS(f(s_i), g(s_j)) = \beta + 1$ otherwise.

$s_0$ – vector of all 1s (no vector orthog. to $s_0$)

**Idea for hardness for LCS**

0 and 1 don’t appear in the $f$ and $g$ gadgets

**Attempt 2:**

$x = Q f(s_1) R Q f(s_2) R \ldots Q f(s_n) R$

$y = (Q g(s_0) R)^{n-1} Q g(s_1) R Q g(s_2) R \ldots Q g(s_n) R (Q g(s_0) R)^{n-1}$

**Lemma:** If a 0 (or 1) is matched, its entire $0^q$ (or $1^q$) block is matched.

**Idea:** Pick $q$ big so all $Q$s and $R$s of $x$ must be matched in an LCS.

Now no $g(s_k)$ is aligned with two different $f(s_i)$ and $f(s_j)$. 

$Q = 0^q$, $R = 1^q$
Let \( S = \{s_1, s_2, \ldots, s_n\} \) be the vectors
Each \( s_i \rightarrow \) gadget sequences \( f(s_i) \) and \( g(s_i) \)
\( \text{LCS}(f(s_i), g(s_j)) = \beta \) if \( s_i \cdot s_j \neq 0 \), \( \text{LCS}(f(s_i), g(s_j)) = \beta + 1 \) otherwise.
\( s_0 \) – vector of all 1s (no vector orthog. to \( s_0 \))

**Idea for hardness for LCS**

0 and 1 don’t appear in the f and g gadgets

**Attempt 2:**
\[
x = Q \ f(s_1) R \ Q \ f(s_2) R \ldots \ Q \ f(s_n) R
\]
\[
y = (Qg(s_0) R)^{n-1} Qg(s_1) R \ Q \ g(s_2) R \ldots \ Q \ g(s_n) R \ (Qg(s_0) R)^{n-1}
\]

**Lemma:** If a 0 (or 1) is matched, its entire \( 0^q \) (or \( 1^q \)) block is matched.

**Idea:** Pick \( q \) big so all \( Qs \) and \( Rs \) of \( x \) must be matched in an LCS.
Now no \( g(s_k) \) is aligned with two different \( f(s_i) \) and \( f(s_j) \).

**Problem:** LCS might align \( f(s_i) \) with several \( g(s_k) \).
Let $S = \{s_1, s_2, ..., s_n\}$ be the vectors.

Each $s_i \rightarrow$ gadget sequences $f(s_i)$ and $g(s_i)$

$LCS(f(s_i), g(s_i)) = \beta$ if $s_i \cdot s_j \neq 0$, $LCS(f(s_i), g(s_i)) = \beta + 1$ otherwise.

$s_0$ – vector of all 1s (no vector orthog. to $s_0$)

**Attempt 2:**

$x = Q f(s_1) R Q f(s_2) R ... Q f(s_n) R$

$y = (Qg(s_0) R)^{n-1} Qg(s_1) R Q g(s_2) R ... Q g(s_n) R (Qg(s_0) R)^{n-1}$

**Lemma:** If a 0 (or 1) is matched, its entire $0^q$ (or $1^q$) block is matched.

**Idea:** Pick $q$ big so all $Q$s and $R$s of $x$ must be matched in an LCS.

Now no $g(s_k)$ is aligned with two different $f(s_i)$ and $f(s_j)$.

**Problem:** LCS might align $f(s_i)$ with **several** $g(s_k)$.

The $g(s_k)$ are partitioned into blocks aligned with at most a single $f(s_i)$. 
Let $S = \{s_1, s_2, \ldots, s_n\}$ be the vectors. Each $s_i \rightarrow$ sequences $f(s_i)$ and $g(s_i)$. 

$LCS(f(s_i), g(s_j)) = \beta$ if $s_i \cdot s_j \neq 0$, $\geq \beta + 1$ otherwise.

$s_0$ – vector of all 1s (no vector orthog. to $s_0$)

**LCS hardness idea**

Attempt 3:

$x = P|y|Q f(s_1) R Q f(s_2) R Q \ldots R Q f(s_n) R P|y|$

$y = P (Q g(s_0) R P)^{n-1} Q g(s_1) R P \ Q g(s_2) R P \ldots \ Q g(s_n) R P (Q g(s_0) R P)^{n-1}$

$Q=0^q, R=1^q, P=2^r$
Let $S = \{s_1, s_2, \ldots, s_n\}$ be the vectors
Each $s_i \rightarrow$ sequences $f(s_i)$ and $g(s_i)$
$LCS(f(s_i), g(s_j)) = \beta$ if $s_i \cdot s_j \neq 0$, $\geq \beta + 1$ otherwise.
$s_0$ – vector of all 1s (no vector orthog. to $s_0$)

**LCS hardness idea**

**Attempt 3:**

$x = P | y | Q f(s_1) R Q f(s_2) R Q \ldots R Q f(s_n) R P | y |$

$y = P (Q g(s_0) R P)^{n-1} Q g(s_1) R P \ldots Q g(s_n) R P (Q g(s_0) R P)^{n-1}$

**Idea:**

$Q=0^q,R=1^q,P=2^r$
Let $S = \{s_1, s_2, \ldots, s_n\}$ be the vectors.
Each $s_i \rightarrow$ sequences $f(s_i)$ and $g(s_i)$
$LCS(f(s_i), g(s_j)) = \beta$ if $s_i \cdot s_j \neq 0$, $\geq \beta + 1$
otherwise.
$s_0$ – vector of all 1s (no vector orthog. to $s_0$)

**LCS hardness idea**

**Attempt 3:**

$x = P^{|y|} Q f(s_1) R Q f(s_2) R Q \ldots R Q f(s_n) R P^{|y|}$

$y = P (Q g(s_0) R P)^{n-1} Q g(s_1) R P Q g(s_2) R P \ldots Q g(s_n) R P (Q g(s_0) R P)^{n-1}$

**Idea:**

$P = 2^r$, $r$ big but $r << q$, so that in an LCS all Qs and Rs of $x$ are still aligned,
and also as many Ps as possible from $y$ are aligned.
Let $S = \{s_1, s_2, \ldots, s_n\}$ be the vectors. Each $s_i$ → sequences $f(s_i)$ and $g(s_i)$. LCS($f(s_i), g(s_j)$) = $\beta$ if $s_i \cdot s_j \neq 0$, $\geq \beta + 1$ otherwise.

$s_0$ – vector of all 1s (no vector orthog. to $s_0$)

**Attempt 3:**

$x = P | y | Q \ f(s_1) R \ Q \ f(s_2) R \ Q \ ... \ R Q \ f(s_n) R \ P | y |$

$y = P \ (Q g(s_0) R P)^{n-1} \ Q g(s_1) R \ P \ Q g(s_2) R \ P \ ... \ Q g(s_n) R \ P \ (Q g(s_0) R P)^{n-1}$

**Idea:**

$P = 2^r$, $r$ big but $r << q$, so that in an LCS all $Q$s and $R$s of $x$ are still aligned, and also as many $P$s as possible from $y$ are aligned.

$\geq n - 1$ Ps of $y$ not matched in an LCS due to the matched $Q$s and $R$s of $x$. 

Q=0^q, R=1^q, P=2^r
Let $S = \{s_1, s_2, \ldots, s_n\}$ be the vectors
Each $s_i \rightarrow$ sequences $f(s_i)$ and $g(s_i)$
$LCS(f(s_i), g(s_j)) = \beta$ if $s_i \cdot s_j \neq 0$, $\geq \beta + 1$ otherwise.
$s_0$ – vector of all 1s (no vector orthog. to $s_0$

**Attempt 3:**

$$x = P^{|y|} Q f(s_1) R Q f(s_2) R Q \ldots R Q f(s_n) R P^{|y|}$$

$$y = P (Q g(s_0) R P)^{n-1} Q g(s_1) R P Q g(s_2) R P \ldots Q g(s_n) R P (Q g(s_0) R P)^{n-1}$$

**Idea:**

$P = 2^r$, $r$ big but $r \ll q$, so that in an LCS all Qs and Rs of $x$ are still aligned,
and also as many Ps as possible from $y$ are aligned.

$\geq n-1$ Ps of $y$ not matched in an LCS due to the matched Qs and Rs of $x$.
Thus, exactly $n-1$ Ps will be unmatched, and every $f(s_i)$ will be fully aligned with some $g(s_j)$ (possibly $j=0$).
Let $S = \{s_1, s_2, ..., s_n\}$ be the vectors
Each $s_i \rightarrow$ sequences $f(s_i)$ and $g(s_i)$
$LCS(f(s_i), g(s_j)) = \beta$ if $s_i \cdot s_j \neq 0$, $\geq \beta + 1$ otherwise.
$s_0$ – vector of all 1s (no vector orthog. to $s_0$)

**LCS hardness idea**

**Attempt 3:**

$x = P^|y| Q f(s_1) R Q f(s_2) R Q \ldots R Q f(s_n) R P^|y|$

$y = P (Q g(s_0) R P)^{n-1} Q g(s_1) R P \ldots Q g(s_n) R P (Q g(s_0) R P)^{n-1}$

**Idea:**

$P = 2^r$, $r$ big but $r \ll q$, so that in an LCS all Qs and Rs of $x$ are still aligned, and also as many Ps as possible from $y$ are aligned.

$\geq n-1$ Ps of $y$ not matched in an LCS due to the matched Qs and Rs of $x$. Thus, **exactly** $n-1$ Ps will be unmatched, and every $f(s_i)$ will be fully aligned with some $g(s_j)$ (possibly $j=0$).

The gadgets $f(s_i)$ and $g(s_j)$ act as letters!
LCS hardness idea

Attempt 3:

\[ x = p^{\mid y\mid}q f(s_1)R \ Q \ f(s_2)R \ Q \ ... \ RQ \ f(s_n) \ R \ p^{\mid y\mid} \]

\[ y = P (Qg(s_0) \ R \ P)^{n-1} Q \ g(s_1) \ R \ P \ Q \ g(s_2) \ R \ P \ ... \ Q \ g(s_n) \ R \ P (Q \ g(s_0) \ R \ P)^{n-1} \]

LCS length:
Let $S = \{s_1, s_2, \ldots, s_n\}$ be the vectors
Each $s_i \rightarrow$ sequences $f(s_i)$ and $g(s_i)$
$LCS(f(s_i), g(s_j)) = \beta$ if $s_i \cdot s_j \neq 0$, $\geq \beta + 1$ otherwise.
s_0 \rightarrow$ vector of all 1s (no vector orthog. to s_0)

**LCS hardness idea**

**Attempt 3:**

$$x = P^{|y|}Qf(s_1)RQf(s_2)RQ \ldots RQf(s_n)R P^{|y|}$$

$$y = P(Qg(s_0)RP)^{n-1}Qg(s_1)R P Qg(s_2)R P \ldots Qg(s_n)R P(Qg(s_0)RP)^{n-1}$$

**LCS length:**

$$2n|P| + n(|Q| + |R|) + \sum_{i=1}^{n} LCS(f(s_i), g(s_j)), g(s_j) \text{ aligned with } f(s_i)$$

$Q=0^q, R=1^q, P=2^r$
Let $S = \{s_1, s_2, \ldots, s_n\}$ be the vectors
Each $s_i \rightarrow$ sequences $f(s_i)$ and $g(s_i)$
$LCS(f(s_i), g(s_j)) = \beta$ if $s_i \cdot s_j \neq 0$, $\geq \beta + 1$ otherwise.

$s_0$ – vector of all 1s (no vector orthog. to $s_0$)

**LCS hardness idea**

**Attempt 3:**

$x = P^{\mid y \mid} Q f(s_1) R Q f(s_2) R \ldots R Q f(s_n) R P^{\mid y \mid}$

$y = P (Q g(s_0) R P)^{n-1} Q g(s_1) R P Q g(s_2) R P \ldots Q g(s_n) R P (Q g(s_0) R P)^{n-1}$

#Ps in $y$ is $3n-1$, and $n-1$ are not matched, so $2n$ aligned.

**LCS length:**

$2n|P| + n(|Q| + |R|) + \sum_{i=1}^{n} LCS(f(s_i), g(s_j))$, $g(s_j)$ aligned with $f(s_i)$

$= 2nr + 2qn + n \beta$ if no orthog. pair

$\geq [2nr + 2qn + n \beta] + 1$ if 9 an orthog. pair.
Let $S = \{s_1, s_2, ..., s_n\}$ be the vectors
Each $s_i \rightarrow$ sequences $f(s_i)$ and $g(s_i)$
$LCS(f(s_i), g(s_j)) = \beta$ if $s_i \cdot s_j \neq 0$, $\geq \beta + 1$ otherwise.
$s_0$ – vector of all 1s (no vector orthog. to $s_0$)

**LCS hardness idea**

**Reduction:**

$x = P|y|Q f(s_1)R Q f(s_2)R Q ... RQ f(s_n)R P|y|$

$y = P (Qg(s_0) R P)^{n-1} Q g(s_1) R P Q g(s_2) R P ... Q g(s_n) R P (Q g(s_0) R P)^{n-1}$
Let $S = \{s_1, s_2, \ldots, s_n\}$ be the vectors
Each $s_i \rightarrow$ sequences $f(s_i)$ and $g(s_i)$
$LCS(f(s_i), g(s_j)) = \beta$ if $s_i \cdot s_j \neq 0$, $\geq \beta + 1$
otherwise.
$s_0$ – vector of all 1s (no vector orthog. to $s_0$)

**LCS hardness idea**

**Reduction:**

$x = \mathcal{P}^{\mathcal{y}} \mathcal{Q} f(s_1) \mathcal{R} \mathcal{Q} f(s_2) \mathcal{R} \ldots \mathcal{R} Q f(s_n) \mathcal{R} \mathcal{P}^{\mathcal{y}}$

$y = \mathcal{P} (\mathcal{Q} g(s_0) \mathcal{R} \mathcal{P})^{n-1} \mathcal{Q} g(s_1) \mathcal{R} \mathcal{P} Q g(s_2) \mathcal{R} \mathcal{P} \ldots \mathcal{Q} g(s_n) \mathcal{R} \mathcal{P} (\mathcal{Q} g(s_0) \mathcal{R} \mathcal{P})^{n-1}$

Tricky proof in paper shows the following suffice:

$|Q|, |R|, |P|, |f(s_i)|, |g(s_i)| \leq \text{poly}(d)$, so that

$|x|, |y| \leq n \text{ poly}(d)$. 
OV to LCS

Given vectors \( \{s_1, \ldots, s_n\} \), \( s_i \in \{0,1\}^d \) \( \forall i \), OV is

\[
\bigvee_{i,j \in [n]} \bigwedge_{k \in [d]} (\neg s_i[k] \lor \neg s_j[k]).
\]

**Outer OR gadgets** \( x, y \) taking sets of bit vectors \( \{s_1, \ldots, s_n\} \), to short sequences s.t. for some \( Q \)

- \( LCS(x, y) = Q \) if \( \forall i, j: s_i \cdot s_j \neq 0 \),
- \( LCS(x, y) \geq Q + 1 \) if \( \exists i, j: s_i \cdot s_j = 0 \).

**Vector gadgets** \( f, g \) taking bit vectors to short sequences s.t. for some \( T \)

- \( LCS(f(s_i), g(s_j)) = T + 1 \) if \( s_i \cdot s_j = 0 \),
- \( LCS(f(s_i), g(s_j)) = T \) if \( s_i \cdot s_j \neq 0 \).

**Coordinate gadgets** \( c, e \) taking bits to short sequences s.t.

- \( LCS(c(x), e(y)) = 0 \) if \( x = y = 1 \),
- \( LCS(c(x), e(y)) = 1 \) if \( x \cdot y = 0 \).
OV to LCS

Given vectors \{s_1, \ldots, s_n\}, \(s_i \in \{0,1\}^d\) \(\forall i\), OV is

\[
\bigvee_{i,j \in [n]} \bigwedge_{k \in [d]} (\neg s_i[k] \lor \neg s_j[k]).
\]

**Outer OR gadgets** \(x, y\) taking sets of bit vectors \{s_1, \ldots, s_n\}, to short sequences s.t. for some \(Q\)

\[
LCS(x, y) = Q \text{ if } \forall i, j: s_i \cdot s_j \neq 0,
\]

\[
LCS(x, y) \geq Q + 1 \text{ if } \exists i, j: s_i \cdot s_j = 0.
\]

**Vector gadgets** \(f, g\) taking bit vectors to short sequences s.t. for some \(T\)

\[
LCS(f(s_i), g(s_j)) = T + 1 \text{ if } s_i \cdot s_j = 0,
\]

\[
LCS(f(s_i), g(s_j)) = T \text{ if } s_i \cdot s_j \neq 0.
\]

**Coordinate gadgets** \(c, e\) taking bits to short sequences s.t.

\[
LCS(c(x), e(y)) = 0 \text{ if } x = y = 1,
\]

\[
LCS(c(x), e(y)) = 1 \text{ if } x \cdot y = 0.
\]

**Done!**
Given vectors \( \{s_1, \ldots, s_n\} \), \( s_i \in \{0,1\}^d \) \( \forall i \), OV is

\[
\bigvee_{i,j \in [n]} \bigwedge_{k \in [d]} \left( \neg s_i[k] \lor \neg s_j[k] \right).
\]

\textbf{Outer OR gadgets} \( x, y \) taking sets of bit vectors \( \{s_1, \ldots, s_n\} \), to short sequences s.t. for some \( Q \)
\( LCS(x, y) = Q \) if \( \forall i, j: s_i \cdot s_j \neq 0 \),
\( LCS(x, y) \geq Q + 1 \) if \( \exists i, j: s_i \cdot s_j = 0 \).

\textbf{Vector gadgets} \( f, g \) taking bit vectors to short sequences s.t. for some \( T \)
\( LCS(f(s_i), g(s_j)) = T + 1 \) if \( s_i \cdot s_j = 0 \),
\( LCS(f(s_i), g(s_j)) = T \) if \( s_i \cdot s_j \neq 0 \).

\textbf{Coordinate gadgets} \( c, e \) taking bits to short sequences s.t.
\( LCS(c(x), e(y)) = 0 \) if \( x = y = 1 \),
\( LCS(c(x), e(y)) = 1 \) if \( x \cdot y = 0 \).

\( c(0) = 46 \quad e(0) = 64 \)
\( c(1) = 4 \quad e(1) = 6 \)

LCS\((c(1),e(1)) = 0\), and
LCS\((c(x),e(y)) = 1 \)
for \( (x,y) \neq (1,1) \).
OV to LCS

Given vectors \(\{s_1, \ldots, s_n\}\), \(s_i \in \{0,1\}^d\) \(\forall i\), OV is

\[
\bigvee_{i,j \in [n]} \bigwedge_{k \in [d]} (\neg s_i[k] \vee \neg s_j[k]).
\]

Coordinate gadgets \(c, e\) taking bits to short sequences s.t.
\[
\begin{align*}
LCS(c(x), e(y)) &= 0 \text{ if } x = y = 1, \\
LCS(c(x), e(y)) &= 1 \text{ if } x \cdot y = 0.
\end{align*}
\]

Vector gadgets \(f, g\) taking bit vectors to short sequences s.t. for some \(T\)
\[
\begin{align*}
LCS(f(s_i), g(s_j)) &= T + 1 \text{ if } s_i \cdot s_j = 0, \\
LCS(f(s_i), g(s_j)) &= T \text{ if } s_i \cdot s_j \neq 0.
\end{align*}
\]

Outer OR gadgets \(x, y\) taking sets of bit vectors \(\{s_1, \ldots, s_n\}\), to short sequences s.t. for some \(Q\)
\[
\begin{align*}
LCS(x, y) &= Q \text{ if } \forall i, j: s_i \cdot s_j \neq 0, \\
LCS(x, y) &\geq Q + 1 \text{ if } \exists i, j: s_i \cdot s_j = 0.
\end{align*}
\]

Done!

All that remains!

\[
\begin{align*}
c(0) &= 46 \quad e(0) = 64 \\
c(1) &= 4 \quad e(1) = 6
\end{align*}
\]

\[
\begin{align*}
LCS(c(1), e(1)) &= 0, \text{ and} \\
LCS(c(x), e(y)) &= 1 \text{ for } (x,y) \neq (1,1).
\end{align*}
\]
**Want:** Each $s_i \rightarrow$ sequences $f(s_i)$ and $g(s_i)$
$LCS(f(s_i), g(s_j)) = \beta$ if $s_i \cdot s_j \neq 0$, $= \beta + 1$ otherwise

Recall we have *coordinate gadgets*
$x \in \{0, 1\} \rightarrow c(x)$ and $e(x)$, s.t.
$LCS(c(x), e(y)) = 0$ if $x = y = 1$ and $1$ otherwise; also, $|c(x)|, |e(x)| \leq 2$. 

\[
\bigvee_{i,j \in [n]} \bigwedge_{c \in [d]} (\neg \nu_i[c] \lor \neg \nu_j[c])
\]
**Vector gadgets**

Recall we have coordinate gadgets \( x \in \{0, 1\} \rightarrow c(x) \) and \( e(x) \), s.t.
\[
LCS(c(x), e(y)) = 0 \text{ if } x = y = 1 \text{ and } 1 \text{ otherwise}; \text{ also, } |c(x)|, |e(x)| \leq 2.
\]

**Want:** Each \( s_i \) \( \rightarrow \) sequences \( f(s_i) \) and \( g(s_j) \)
\[
LCS(f(s_i), g(s_j)) = \beta \text{ if } s_i \cdot s_j \neq 0, = \beta + 1 \text{ otherwise}
\]

\[
f(s_i) = 3^r \ 5^u \ c(s_i[1]) \ 5^u \ldots \ 5^u \ c(s_i[d]) \ 5^u
g(s_j) = 5^u \ e(s_j[1]) \ 5^u \ldots \ 5^u \ e(s_j[d]) \ 5^u \ 3^r
\]

where \( r = u(d+1)+d-1 \), \( u > d+1 \).
**Want:** Each $s_i$ $\rightarrow$ sequences $f(s_i)$ and $g(s_j)$
$LCS(f(s_i), g(s_j)) = \beta$ if $s_i \cdot s_j \neq 0$, $= \beta + 1$ otherwise

Recall we have coordinate gadgets
$x \in \{0, 1\} \rightarrow c(x)$ and $e(x)$, s.t.
$LCS(c(x), e(y)) = 0$ if $x = y = 1$ and 1 otherwise; also, $|c(x)|, |e(x)| \leq 2.$

$f(s_i) = 3^r \ 5^u \ c(s_i[1]) \ 5^u \ ... \ 5^u \ c(s_i[d]) \ 5^u$
$g(s_j) = 5^u \ e(s_j[1]) \ 5^u \ ... \ 5^u \ e(s_j[d]) \ 5^u \ 3^r$

where $r = u(d+1)+d-1$, $u > d+1$. 

3,5 brand new symbols
$u$ is large,
$r$ even larger
Want: Each $s_i \rightarrow$ sequences $f(s_i)$ and $g(s_i)$
$LCS(f(s_i),g(s_j)) = \beta$ if $s_i \cdot s_j \neq 0$, $= \beta + 1$ otherwise

Recall we have coordinate gadgets
$x \in \{0, 1\} \rightarrow c(x)$ and $e(x)$, s.t.
$LCS(c(x),e(y)) = 0$ if $x = y =1$ and $1$ otherwise; also, $|c(x)|,|e(x)| \leq 2$.

$f(s_i) = 3^r \ 5^u \ c(s_i[1]) \ 5^u \ ... \ 5^u \ c(s_i[d]) \ 5^u$
$g(s_j) = 5^u \ e(s_j[1]) \ 5^u \ ... \ 5^u \ e(s_j[d]) \ 5^u \ 3^r$

where $r = u(d+1)+d-1$, $u > d+1$.

If two 5s are matched together, their entire 5u blocks are matched.
If any 3 is matched, no other symbols are, so the LCS length is $r$. 

\[
\bigvee_{i,j \in [n]} \bigwedge_{c \in [d]} (\neg v_i[c] \vee \neg v_j[c])
\]
Vector gadgets

Want: Each $s_i \rightarrow$ sequences $f(s_i)$ and $g(s_i)$
$LCS(f(s_i),g(s_j)) = \beta$ if $s_i \cdot s_j \neq 0$, $= \beta + 1$ otherwise

Recall we have coordinate gadgets
$x \in \{0, 1\} \rightarrow c(x) \text{ and } e(x), \text{ s.t.}$
$LCS(c(x),e(y)) = 0 \text{ if } x = y = 1 \text{ and } 1 \text{ otherwise}$; also, $|c(x)|, |e(x)| \leq 2.$

\[ f(s_i) = 3^r \ 5^u \ c(s_i[1]) \ 5^u \ldots \ 5^u \ c(s_i[d]) \ 5^u \]
\[ g(s_j) = 5^u \ e(s_j[1]) \ 5^u \ldots \ 5^u \ e(s_j[d]) \ 5^u \ 3^r \]

where $r = u(d+1)+d-1, u > d+1.$

If two 5s are matched together, their entire 5$^u$ blocks are matched.
If any 3 is matched, no other symbols are, so the LCS length is $r.$
**Want:** Each $s_i \rightarrow$ sequences $f(s_i)$ and $g(s_j)$

$LCS(f(s_i),g(s_j)) = \beta$ if $s_i \cdot s_j \neq 0$, $= \beta + 1$ otherwise

Recall we have **coordinate gadgets**

$x \in \{0, 1\} \rightarrow c(x)$ and $e(x)$, s.t.

$LCS(c(x),e(y)) = 0$ if $x = y = 1$ and 1 otherwise; also, $|c(x)|, |e(x)| \leq 2$.

$f(s_i) = 3^r \; 5^u \; c(s_i[1]) \; 5^u \; \ldots \; 5^u \; c(s_i[d]) \; 5^u$

$g(s_j) = 5^u \; e(s_j[1]) \; 5^u \; \ldots \; 5^u \; e(s_j[d]) \; 5^u \; 3^r$

where $r = u(d+1)+d-1$, $u > d+1$.

If two $5$s are matched together, their entire $5^u$ blocks are matched.

If any $3$ is matched, no other symbols are, so the LCS length is $r$.

If no $3$ is matched in an LCS, then all $5$s must be: if a $5^u$ block is not matched, then the subsequence length would be $\leq du + 2d < r$. 

\[ \bigvee_{i,j \in [n]} \bigwedge_{c \in [d]} (\neg v_i[c] \lor \neg v_j[c]) \]
Recall we have coordinate gadgets $x \in \{0, 1\} \rightarrow c(x)$ and $e(x)$, s.t. $LCS(c(x), e(y)) = 0$ if $x = y = 1$ and $1$ otherwise; also, $|c(x)|, |e(x)| \leq 2$.

Want: Each $s_i \rightarrow$ sequences $f(s_i)$ and $g(s_j)$
$LCS(f(s_i), g(s_j)) = \beta$ if $s_i \cdot s_j \neq 0$, $= \beta + 1$ otherwise

$\forall i, j \in [n] \bigg( \bigwedge_{c \in [d]} (\neg v_i[c] \lor \neg v_j[c]) \bigg)$

$f(s_i) = 3^r \ 5^u \ c(s_i[1]) \ 5^u \ ... \ 5^u \ c(s_i[d]) \ 5^u$
$g(s_j) = 5^u \ e(s_j[1]) \ 5^u \ ... \ 5^u \ e(s_j[d]) \ 5^u \ 3^r$

Where $r = u(d+1)+d-1$, $u > d+1$.

If two $5$s are matched together, their entire $5^u$ blocks are matched.
If any $3$ is matched, no other symbols are, so the LCS length is $r$.
If no $3$ is matched in an LCS, then all $5$s must be: if a $5^u$ block is not matched, then the subsequence length would be $\leq du + 2d < r$. 

3, 5 brand new symbols $u$ is large, $r$ even larger.
**Want:** Each $s_i \rightarrow$ sequences $f(s_i)$ and $g(s_i)$
$LCS(f(s_i), g(s_j)) = \beta$ if $s_i \cdot s_j \neq 0$, $= \beta + 1$ otherwise

Vector gadgets

\[ \bigvee_{i,j \in [n]} \wedge_{c \in [d]} (\neg v_i[c] \lor \neg v_j[c]) \]

Recall we have coordinate gadgets
$x \in \{0, 1\} \rightarrow c(x)$ and $e(x)$, s.t.
$LCS(c(x), e(y)) = 0$ if $x = y = 1$ and 1 otherwise; also, $|c(x)|, |e(x)| \leq 2$.

$f(s_i) = 3^r \ 5^u \ c(s_i[1]) \ 5^u \ldots \ 5^u \ c(s_i[d]) \ 5^u$
$g(s_j) = 5^u \ e(s_j[1]) \ 5^u \ldots \ 5^u \ e(s_j[d]) \ 5^u \ 3^r$
where $r = u(d+1)+d-1$, $u > d+1$.

If two 5s are matched together, their entire $5^u$ blocks are matched.
If any 3 is matched, no other symbols are, so the LCS length is $r$.
If no 3 is matched in an LCS, then all 5s must be: if a $5^u$ block is not matched, then the subsequence length would be $\leq du + 2d < r$. 

3,5 brand new symbols
$u$ is large, $r$ even larger
Want: Each $s_i \rightarrow$ sequences $f(s_i)$ and $g(s_j)$
LCS($f(s_i), g(s_j)$) = $\beta$ if $s_i \cdot s_j \neq 0$, = $\beta + 1$ otherwise

\[ \bigvee_{i,j \in [n]} \land_{c \in [d]} (\neg v_i[c] \lor \neg v_j[c]) \]

Recall we have coordinate gadgets $x \in \{0, 1\} \rightarrow c(x)$ and $e(x)$, s.t.
LCS($c(x), e(y)$) = 0 if $x = y = 1$ and 1 otherwise; also, $|c(x)|, |e(x)| \leq 2.$

\[ f(s_i) = 3^r \ 5^u c(s_i[1]) \ 5^u \ldots \ 5^u c(s_i[d]) \ 5^u \]
\[ g(s_j) = 5^u e(s_j[1]) \ 5^u \ldots \ 5^u e(s_j[d]) \ 5^u 3^r \]

where $r = u(d+1)+d-1$, $u > d+1$.

If two 5s are matched together, their entire $5^u$ blocks are matched.
If any 3 is matched, no other symbols are, so the LCS length is $r$.
If no 3 is matched in an LCS, then all 5s must be: if a $5^u$ block is not matched, then the subsequence length would be $\leq du + 2d < r$. 

3,5 brand new symbols
$u$ is large, $r$ even larger
**Vector gadgets**

Recall we have *coordinate gadgets* $x \in \{0, 1\} \rightarrow c(x)$ and $e(x)$, s.t.
$LCS(c(x), e(y)) = 0$ if $x = y = 1$ and 1 otherwise; also, $|c(x)|, |e(x)| \leq 2$.

$$\forall i,j \in [n] \left( \exists c \in [d] \left( \neg \nu_i [c] \lor \neg \nu_j [c] \right) \right)$$

**Want:** Each $s_i \rightarrow$ sequences $f(s_i)$ and $g(s_i)$
$LCS(f(s_i), g(s_j)) = \beta$ if $s_i \cdot s_j \neq 0$, $= \beta + 1$ otherwise.

$$f(s_i) = 3^r \ 5^u \ c(s_i[1]) \ 5^u \ldots \ 5^u \ c(s_i[d]) \ 5^u$$
$$g(s_j) = 5^u \ e(s_j[1]) \ 5^u \ldots \ 5^u \ e(s_j[d]) \ 5^u \ 3^r$$

where $r = u(d+1)+d-1$, $u > d+1$.

If two 5s are matched together, their entire $5^u$ blocks are matched.
If any 3 is matched, no other symbols are, so the LCS length is $r$.
If no 3 is matched in an LCS, then all 5s must be: if a $5^u$ block is not matched, then the subsequence length would be $\leq du + 2d < r$. 

3,5 brand new symbols $u$ is large, $r$ even larger
Recall that we have **coordinate gadgets**

\[ x \in \{0, 1\} \rightarrow c(x) \text{ and } e(x), \text{ s.t.} \]

\[ \text{LCS}(c(x), e(y)) = 0 \text{ if } x = y = 1 \text{ and } 1 \]

otherwise; also, \(|c(x)|, |e(x)| \leq 2.\]

**Vector gadgets**

\[ f(s_i) = 3^r 5^u c(s_i[1]) 5^u \ldots 5^u c(s_i[d]) 5^u \]

\[ g(s_j) = 5^u e(s_j[1]) 5^u \ldots 5^u e(s_j[d]) 5^u 3^r \]

where \( r = u(d+1)+d-1, u > d.\]
Recall that we have coordinate gadgets $x \in \{0, 1\} \to c(x)$ and $e(x)$, s.t. $LCS(c(x), e(y)) = 0$ if $x = y = 1$ and 1 otherwise; also, $|c(x)|, |e(x)| \leq 2$.

Vector gadgets

$$f(s_i) = 3^r \ 5^u \ c(s_i[1]) \ 5^u \ ... \ 5^u \ c(s_i[d]) \ 5^u$$

$$g(s_j) = 5^u \ e(s_j[1]) \ 5^u \ ... \ 5^u \ e(s_j[d]) \ 5^u \ 3^r$$

where $r = u(d+1)+d-1$, $u > d$.

Assume no 3 is matched. Then all 5s are matched.
Recall that we have coordinate gadgets \( x \in \{0, 1\} \rightarrow c(x) \) and \( e(x) \), s.t. \( \text{LCS}(c(x), e(y)) = 0 \) if \( x = y = 1 \) and 1 otherwise; also, \(|c(x)|, |e(x)| \leq 2\).

Vector gadgets

\[
\begin{align*}
\text{f}(s_i) &= 3^r \ 5^u \ c(s_i[1]) \ 5^u \ ... \ 5^u \ c(s_i[d]) \ 5^u \\
\text{g}(s_j) &= 5^u \ e(s_j[1]) \ 5^u \ ... \ 5^u \ e(s_j[d]) \ 5^u \ 3^r
\end{align*}
\]

where \( r = u(d+1)+d-1 \), \( u > d \).

Assume no 3 is matched. Then all 5s are matched.
Recall that we have coordinate gadgets $x \in \{0, 1\} \rightarrow c(x)$ and $e(x)$, s.t. $LCS(c(x), e(y)) = 0$ if $x = y = 1$ and 1 otherwise; also, $|c(x)|, |e(x)| \leq 2$.

Assume no 3 is matched. Then all 5s are matched. Thus, for all $t$, $c(s_i[t])$ and $e(s_j[t])$ are matched.

$$f(s_i) = 3^r \ 5^u \ c(s_i[1]) \ 5^u \ ... \ 5^u \ c(s_i[d]) \ 5^u$$

$$g(s_j) = 5^u \ e(s_j[1]) \ 5^u \ ... \ 5^u \ e(s_j[d]) \ 5^u \ 3^r$$

where $r = u(d+1)+d-1$, $u > d$.
Recall that we have coordinate gadgets
\( x \in \{0, 1\} \rightarrow c(x) \text{ and } e(x), \text{s.t.} \)
\( \text{LCS}(c(x), e(y)) = 0 \text{ if } x = y = 1 \text{ and } 1 \)
otherwise; also, \(|c(x)|, |e(x)| \leq 2.\)

Vector gadgets

\[ f(s_i) = 3^r 5^u c(s_i[1]) 5^u \ldots 5^u c(s_i[d]) 5^u \]
\[ g(s_j) = 5^u e(s_j[1]) 5^u \ldots 5^u e(s_j[d]) 5^u 3^r \]
where \( r = u(d+1)+d-1, u > d. \)

Assume no 3 is matched. Then all 5s are matched.
Thus, for all \( t \), \( c(s_i[t]) \) and \( e(s_j[t]) \) are matched.

If \( s_i \cdot s_j \neq 0 \), the alignment of \( c(s_i[t]) \) with \( e(s_j[t]) \) for all \( t \) gives \( < d \), so we get \( \leq (d+1)u+d-1 = r. \) (but then the 3s would be matched, so \( = r \))
Recall that we have coordinate gadgets
\( x \in \{0, 1\} \rightarrow c(x) \) and \( e(x) \), s.t.
\( \text{LCS}(c(x), e(y)) = 0 \) if \( x = y = 1 \) and \( 1 \)
otherwise; also, \( |c(x)|, |e(x)| \leq 2 \).

\[
\begin{align*}
f(s_i) &= 3^r 5^u c(s_i[1]) 5^u \ldots 5^u c(s_i[d]) 5^u \\
g(s_j) &= 5^u e(s_j[1]) 5^u \ldots 5^u e(s_j[d]) 5^u 3^r
\end{align*}
\]
where \( r = u(d+1)+d-1, u > d \).

Assume no 3 is matched. Then all 5s are matched.
Thus, for all \( t \), \( c(s_i[t]) \) and \( e(s_j[t]) \) are matched.

If \( s_i \cdot s_j \neq 0 \), the alignment of \( c(s_i[t]) \) with \( e(s_j[t]) \) for all \( t \) gives \( < d \),
so we get \( \leq (d+1)u+d-1 = r \). (but then the 3s would be matched, so \( = r \))

If \( s_i \cdot s_j = 0 \), we get \( (d+1)u+d = r+1 \).
Recall that we have *coordinate gadgets* $x \in \{0, 1\} \rightarrow c(x)$ and $e(x)$, s.t.
$LCS(c(x), e(y)) = 0$ if $x = y = 1$ and 1 otherwise; also, $|c(x)|, |e(x)| \leq 2$.

Vector gadgets

\[
f(s_i) = 3^r \ 5^u \ c(s_i[1]) \ 5^u \ ... \ 5^u \ c(s_i[d]) \ 5^u
\]
\[
g(s_j) = 5^u \ e(s_j[1]) \ 5^u \ ... \ 5^u \ e(s_j[d]) \ 5^u \ 3^r
\]
where $r = u(d+1)+d-1$, $u > d$.

Assume no 3 is matched. Then all 5s are matched. Thus, for all $t$, $c(s_i[t])$ and $e(s_j[t])$ are matched.

If $s_i \cdot s_j \neq 0$, the alignment of $c(s_i[t])$ with $e(s_j[t])$ for all $t$ gives $< d$, so we get $\leq (d+1)u+d-1 = r$. (but then the 3s would be matched, so $= r$)

If $s_i \cdot s_j = 0$, we get $(d+1)u+d = r+1$. 

$LCS(f(s_i), g(s_j)) = r$ if $s_i \cdot s_j \neq 0$ and
$LCS(f(s_i), g(s_j)) = r+1$ otherwise.
OV to LCS

Given vectors \( \{s_1, \ldots, s_n\}, s_i \in \{0,1\}^d \ \forall i \), OV is

\[
\bigvee_{i,j \in [n]} \bigwedge_{k \in [d]} \left( \neg s_i[k] \lor \neg s_j[k] \right).
\]

**Coordinate gadgets** \(c, e\) taking bits to short sequences s.t.

\[
LCS(c(x), e(y)) = 0 \text{ if } x = y = 1, \quad LCS(c(x), e(y)) = 1 \text{ if } x \cdot y = 0.
\]

\[
\begin{align*}
\text{c(0)} &= 46 & \text{e(0)} &= 64 \\
\text{c(1)} &= 4 & \text{e(1)} &= 6
\end{align*}
\]

LCS(c(1),e(1)) = 0, and LCS(c(x),e(y)) = 1 for \((x,y) \neq (1,1)\).

**Outer OR gadgets** \(x, y\) taking sets of bit vectors \(\{s_1, \ldots, s_n\}\), to short sequences s.t.

\[
LCS(x, y) = Q \text{ if } \forall i, j: s_i \cdot s_j \neq 0, \\
LCS(x, y) \geq Q + 1 \text{ if } \exists i, j: s_i \cdot s_j = 0.
\]

**Vector gadgets** \(f, g\) taking bit vectors to short sequences s.t. for some \(T\)

\[
\begin{align*}
LCS\left(f(s_i), g(s_j)\right) &= T + 1 \text{ if } s_i \cdot s_j = 0, \\
LCS\left(f(s_i), g(s_j)\right) &= T \text{ if } s_i \cdot s_j \neq 0.
\end{align*}
\]

**Done!**
Extensions
Extensions

• **Thm:** For any integer $k \geq 2$,
  
  $k$-LCS cannot be solved in $O(n^{k-\varepsilon})$ time under SETH.
Extensions

• **Thm:** For any integer $k \geq 2$,
  $k$-LCS cannot be solved in $O(n^{k-\varepsilon})$ time under SETH.

• [BK’15]: LCS hard even for binary alphabet
Extensions

• **Thm:** For any integer $k \geq 2$, $k$-LCS cannot be solved in $O(n^{k-\epsilon})$ time under SETH.

• [BK’15]: LCS hard even for binary alphabet

• Hardness based on even more believable assumptions:
Extensions

- **Thm**: For any integer $k \geq 2$, $k$-LCS cannot be solved in $O(n^{k-\varepsilon})$ time under SETH.
- [BK’15]: LCS hard even for binary alphabet

- Hardness based on even more believable assumptions:
  - Reduction works from Max-$k$-SAT, so base on:
Extensions

• **Thm:** For any integer $k \geq 2$, $k$-LCS cannot be solved in $O(n^{k-\epsilon})$ time under SETH.

• [BK’15]: LCS hard even for binary alphabet

• Hardness based on even more believable assumptions:
  – Reduction works from Max-k-SAT, so base on: MAX-k-SAT cannot be solved in $2^{n(1-\epsilon)} \text{poly}(n)$ time for all $k$. 
Extensions

• **Thm:** For any integer $k \geq 2$,
  $k$-LCS cannot be solved in $O(n^{k-\varepsilon})$ time under SETH.

• [BK’15]: LCS hard even for binary alphabet

• Hardness based on even more believable assumptions:
  – Reduction works from Max-k-SAT, so base on:
    MAX-k-SAT cannot be solved in $2^{n(1-\varepsilon)} \text{poly}(n)$ time for all $k$.
    (although – maybe this is equivalent to SETH...)
Extensions

- **Thm:** For any integer $k \geq 2$, $k$-LCS cannot be solved in $O(n^{k-\varepsilon})$ time under SETH.
- [BK’15]: LCS hard even for binary alphabet

- Hardness based on even more believable assumptions:
  - Reduction works from Max-$k$-SAT, so base on: **MAX-$k$-SAT** cannot be solved in $2^{n(1-\varepsilon) \text{ poly}(n)}$ time for all $k$. (although – maybe this is equivalent to SETH...)
  - On much more believable assumptions!
Circuit-Strong-ETH

- SETH is ultimately about SAT of *linear size* CNF-formulas
Circuit-Strong-ETH

- SETH is ultimately about SAT of *linear size* CNF-formulas
  There are more difficult satisfiability problems:
Circuit-Strong-ETH

- SETH is ultimately about SAT of *linear size* CNF-formulas

There are more difficult satisfiability problems:

- CIRCUIT-SAT
- NC-SAT
- NC1-SAT ...
Circuit-Strong-ETH

- SETH is ultimately about SAT of *linear size* CNF-formulas

There are more difficult satisfiability problems:

- CIRCUIT-SAT
- NC-SAT
- NC1-SAT ...

**C-SETH**: satisfiability of circuits from circuit class C on n variables and size s requires $2^{n-o(n)} \text{poly}(s)$ time.
Circuit-Strong-ETH

- SETH is ultimately about SAT of *linear size* CNF-formulas
- There are more difficult satisfiability problems:
  - CIRCUIT-SAT
  - NC-SAT
  - NC1-SAT ...

**C-SETH**: satisfiability of circuits from circuit class $C$ on $n$ variables and size $s$ requires $2^{n-o(n)} \text{poly}(s)$ time.

E.g. NC-SETH should be much more believable!
LCS, Edit Distance and Friends are very hard

AHVW’15: reduction from SAT of “Branching Programs”

Many Consequences:
LCS, Edit Distance and Friends are very hard

AHVW’15: reduction from SAT of “Branching Programs”

Many Consequences:

1. Edit Distance / LCS / ... require $n^{2-o(1)}$ time under NC-SETH.
LCS, Edit Distance and Friends are very hard

1. Edit Distance / LCS / ... require $n^{2-o(1)}$ time under NC-SETH.

2. Shaving logarithms from $n^2$ implies novel circuit lower bounds!
LCS, Edit Distance and Friends are very hard

AHVW’15: reduction from SAT of “Branching Programs”

Many Consequences:

1. Edit Distance / LCS / … require $n^{2-o(1)}$ time under NC-SETH.

2. Shaving logarithms from $n^2$ implies novel circuit lower bounds!

An $\frac{n^2}{\log^{\omega(1)} n}$ alg. $\rightarrow$ ENP is not in NC1.
LCS, Edit Distance and Friends are very hard

1. Edit Distance / LCS / … require $n^{2-o(1)}$ time under NC-SETH.

2. Shaving logarithms from $n^2$ implies novel circuit lower bounds!

OV and APSP have such algs. W'14, AWY'15

An $\frac{n^2}{\log^{\omega(1)} n}$ alg. $\rightarrow$ ENP is not in NC1.
LCS, Edit Distance and Friends are very hard

1. Edit Distance / LCS / ... require $n^{2-o(1)}$ time under NC-SETH.

2. Shaving logarithms from $n^2$ implies novel circuit lower bounds!

OV and APSP have such algs. W’14,AWY’15

An $\frac{n^2}{\log^{\omega(1)} n}$ alg. $\rightarrow$ E$^\text{NP}$ is not in NC1.

An $\frac{n^2}{\log^{1000} n}$ time alg. $\rightarrow$ E$^\text{NP}$ has no non-uniform Boolean formulas of size $n^5$. 

AHVW’15: reduction from SAT of “Branching Programs”
LCS, Edit Distance and Friends are very hard

1. Edit Distance / LCS / … require $n^{2-o(1)}$ time under NC-SETH.

2. Shaving logarithms from $n^2$ implies novel circuit lower bounds!

Many Consequences:

- AHVW'15: reduction from SAT of “Branching Programs”
- OV and APSP have such algs. W'14, AWY'15
- An $\frac{n^2}{\log \omega(1) n}$ alg. $\rightarrow$ $E^{NP}$ is not in NC1.
- An $\frac{n^2}{\log^{1000} n}$ time alg. $\rightarrow$ $E^{NP}$ has no non-uniform Boolean formulas of size $n^5$. 
- Best alg: $\frac{n^2}{\log^2 n}$