Approximating Edit Distance

Mahdi Safarnejad

Joint work with Saeed Seddighin and Soheil Ehsani
Under the supervision of Dr. Ghodsi and Dr. HajiAghai

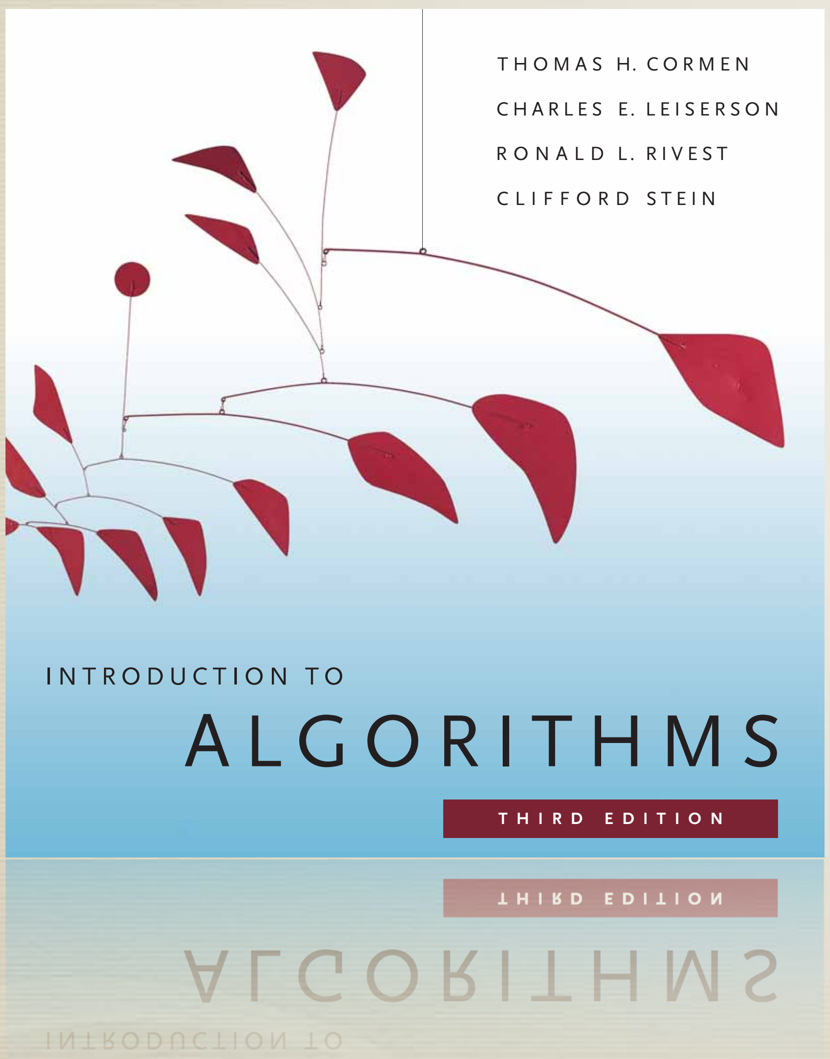
Winter Seminar Series / Sharif University of Technology
29 December 2016

Edit Distance

(a.k.a Levenshtein distance)

The smallest number of insertions, deletions, and substitutions that need to be made on one string (S_1) to transform it to another one (S_2).

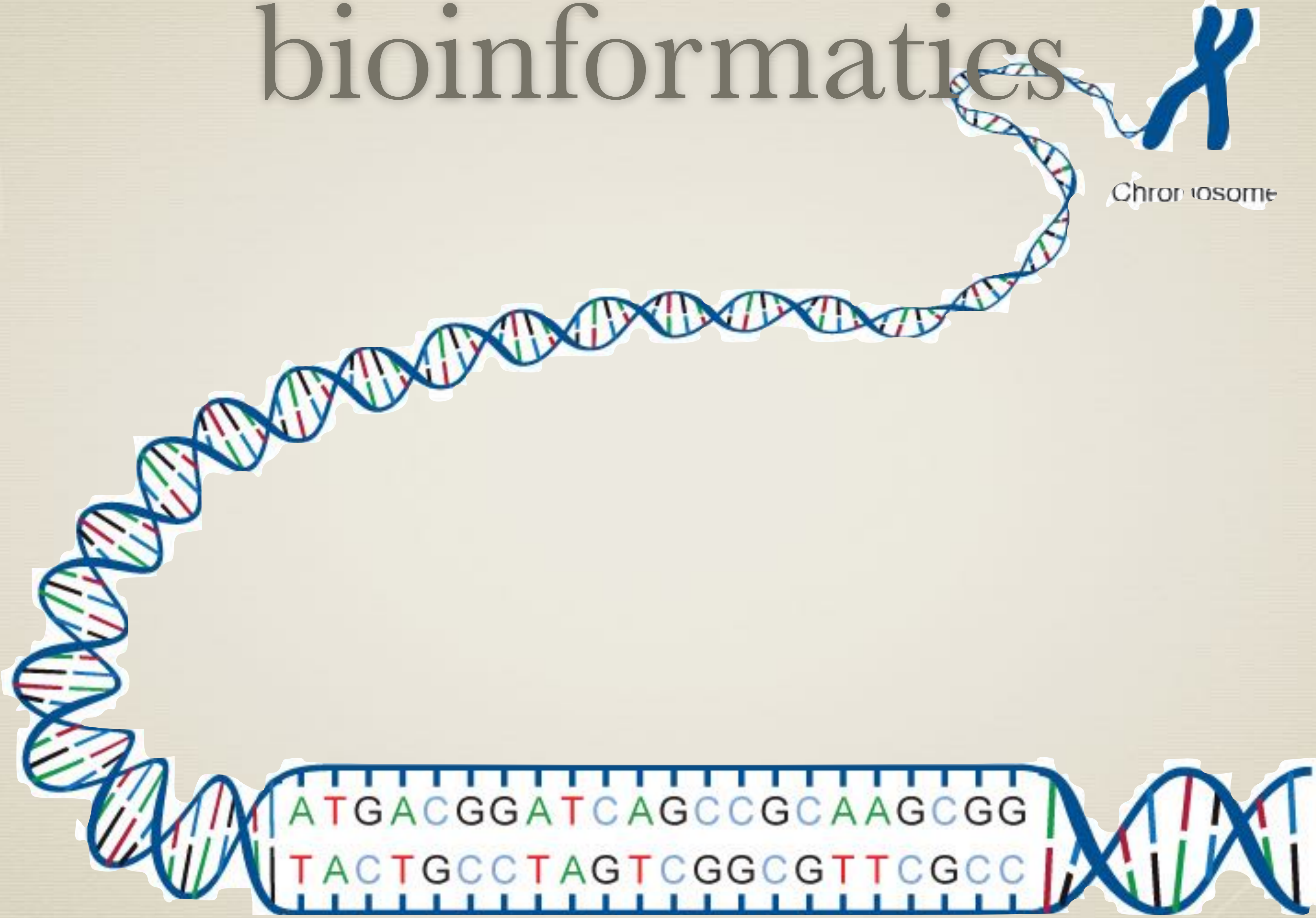
Edit Distance: a textbook example



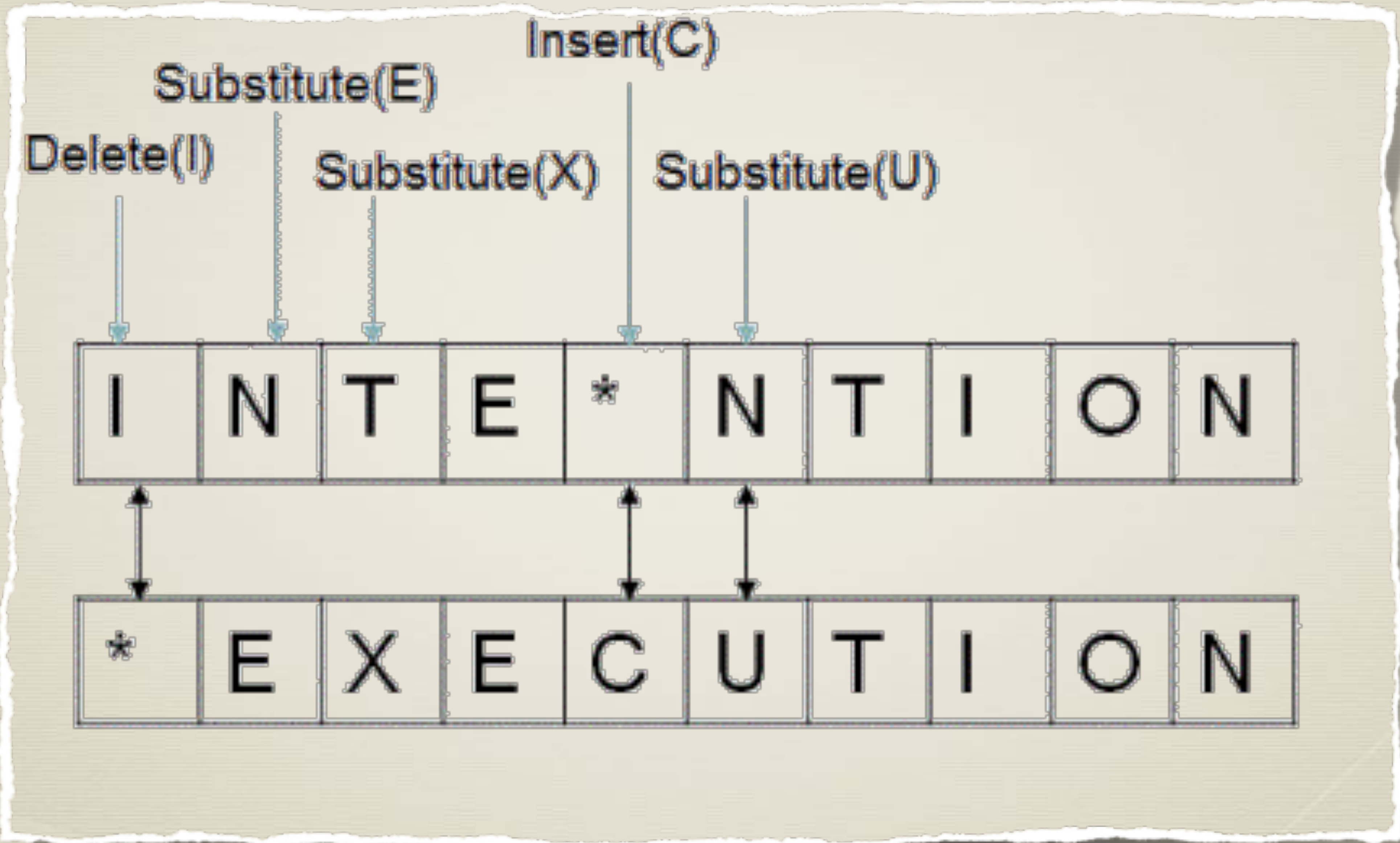
15-5 Edit distance

In order to transform one source string of text $x[1\,..\,m]$ to a target string $y[1\,..\,n]$, we can perform various transformation operations. Our goal is, given x and y, to produce a series of transformations that change x to y. We use an array z—assumed to be large enough to hold all the characters it will need—to hold the intermediate results. Initially, z is empty, and at termination, we should have $z[j] = y[j]$ for $j = 1, 2, \ldots, n$. We maintain current indices i into x and j into z, and the operations are allowed to alter z and these indices. Initially, $i = j = 1$. We are required to examine every character in x during the transformation, which means that at the end of the sequence of transformation operations, we must have

Edit Distance: bioinformatics



Edit Distance: an example



Edit Distance: classic algorithms

* Dynamic Programming - ~1970 - $O(n^2)$

$$d_{i,j} = \begin{cases} d_{i-1,j-1}, & \text{if } s_1[i] = s_2[j] \\ 1 + \min\{d_{i-1,j-1}, d_{i,j-1}, d_{i-1,j}\} & \text{if } s_1[i] \neq s_2[j]. \end{cases}$$

Edit Distance: dynamic programming

-	0	E	X	E	C	U	T	I	O	N
0	0	1	2	3	4	5	6	7	8	9
I	1	1	2	3	4	5	6	6	7	8
N	2	2	2	3	4	5	6	7	7	7
T	3	3	3	3	4	5	5	6	7	8
E	4	3	4	3	4	5	6	6	7	8
N	5	4	4	4	4	5	6	7	7	7
T	6	5	5	5	5	5	5	6	7	8
I	7	6	6	6	6	6	6	5	6	7
O	8	7	7	7	7	7	7	6	5	6
N	9	8	8	8	8	8	8	7	6	5

Edit Distance: classic algorithms

* Dynamic Programming - ~1970 - $O(n^2)$
* Best algorithm - ~1980 - $O(n^2/\log^2 n)$
* No better algorithm unless SETH fails
    * [Backurs & Indyk STOC'15]: no truly subquadratic is possible - $O(n^{2-\varepsilon})$
    * [Abboud et. al. STOC'16]: not many log shaving are possible - $O(n^2/\log^{1000} n)$

Edit Distance: approximation algorithms (in truly subquadratic time)

* $\sqrt{n}$ approximation algorithm (SIAM Journal on Comp. 1998)
* $n^{3/7}$ approximation algorithm (FOCS'04)
* $n^{1/3+o(1)}$ approximation algorithm (SODA'06)
* $2^{O(\sqrt{\log n})}$ approximation algorithm (STOC'09)
* $O(\text{polylog } n)$ approximation algorithm (FOCS'10)
* No constant factor approximation algorithm!

Quantum Algorithms

Quantum Algorithms

* Substantially improve the running time of many algorithmic problems such as
  * Prime factorization and discrete logarithm problems
  * Many graph problems (connectivity, single source shortest paths, minimum spanning tree)
  * Pattern matching (finding a substring in a large string)
  * ...

Quantum Algorithms

* But NO improvement for many classic problems such as
  * Many fundamental problems: sorting / counting
  * All problem with a dynamic programming solution!
    * Edit distance
      * No exact or constant approximation algorithm in truly subquadratic time!

Main Quantum Technique: The Grover's Search (1996)

- Element Listing
  - Input: an array **f** of length **m**, which has **k** ones, and **m-k** zeroes.**f(i)** is available via an oracle access.
  - Output: a list of **k** indices for which **f(i) = 1**.
- The element listing problems can be solved with $O(\sqrt{mk})$ quantum oracle queries (vs. O(m) classical queries).

Our Results and Techniques

Our Results

* We give the first quantum algorithm for approximating the edit distance within a constant factor in subquadratic time.
* An $O(n^{1.857})$ quantum algorithm with an approximation factor of 7
* An $O(n^{1.781})$ quantum algorithm with a (large) **constant** approximation factor

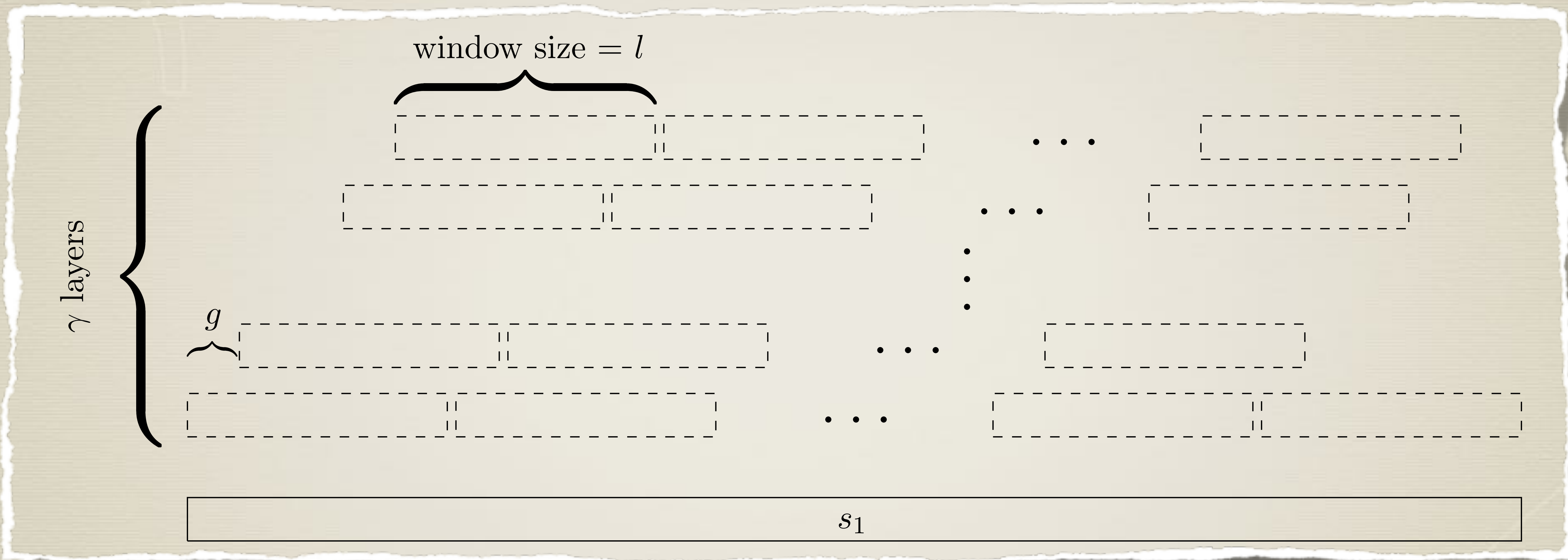
Our Approach

to find a good transformation of S1 into S2

Our Approach: outline

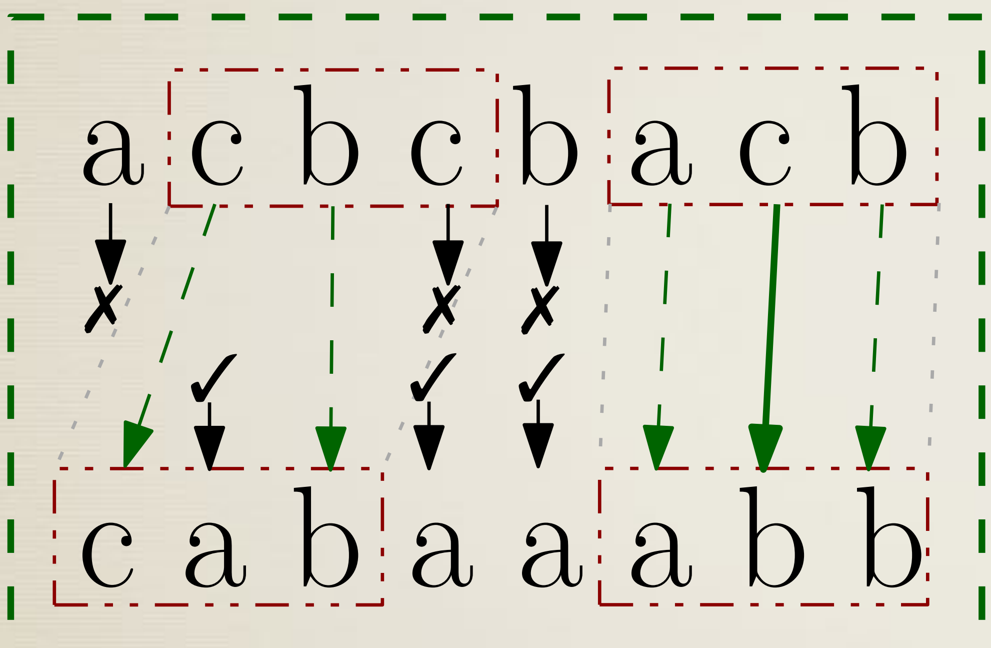
* I. Define some windows (substrings) on S1 and S2 and restrict ourselves to window-compatible transformations
* II. Find an approximate of edit distance between windows by using our metric estimation algorithm [the only step using quantum computing]
* III. Use these distances to find the best windows-compatible transformation, using dynamic programming
* IV. Show that the best windows-compatible transformation is not far from the best (general) transformation

Windows

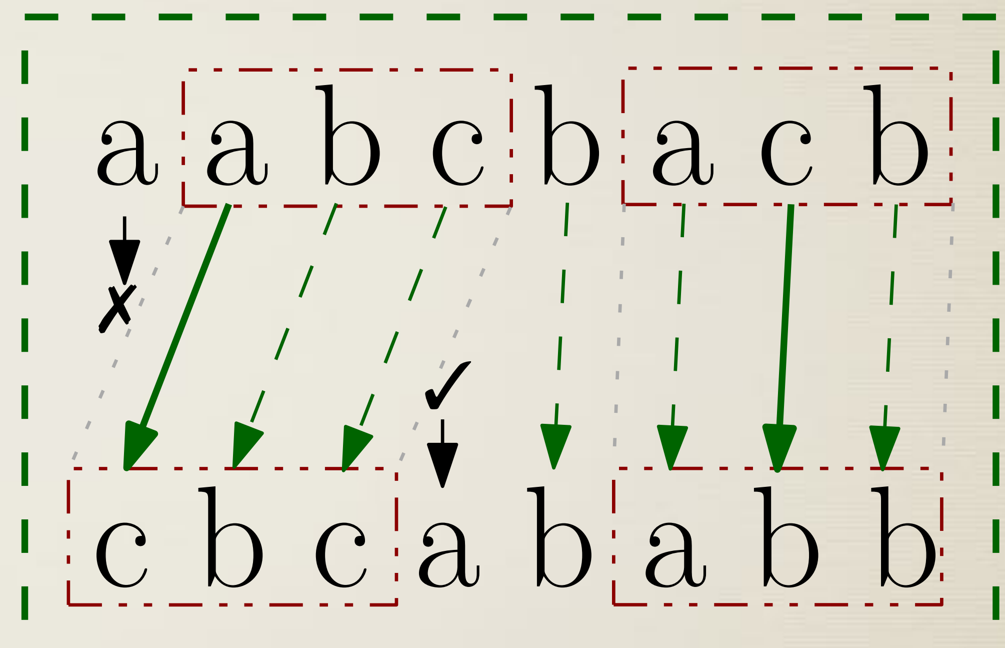


$$l = n^{1/7}, \gamma = 4/\epsilon\delta, g = l/\gamma$$

What is a windows-compatible transformation?

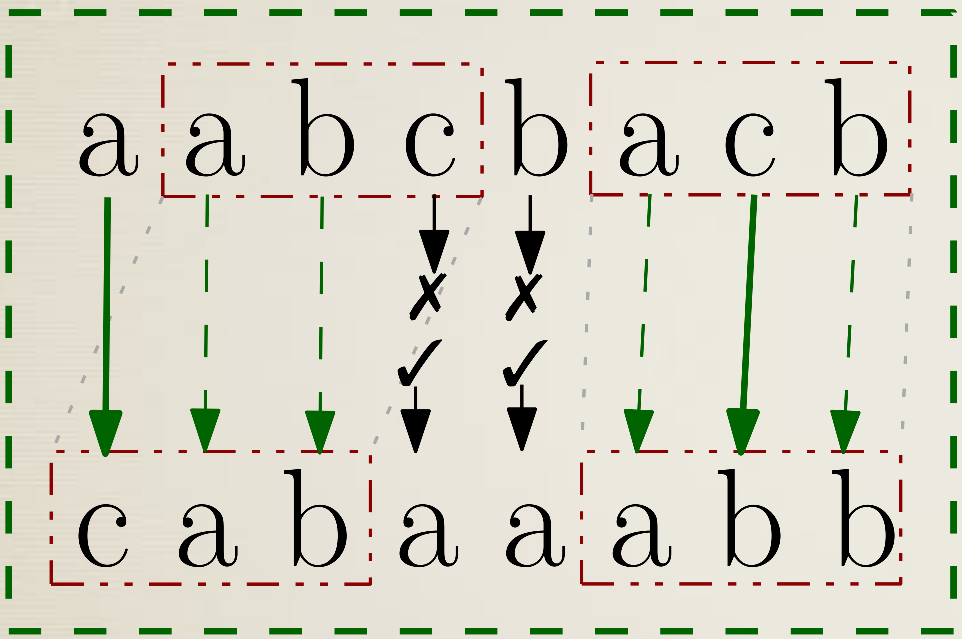


(a) An example of a window-compatible transformation.

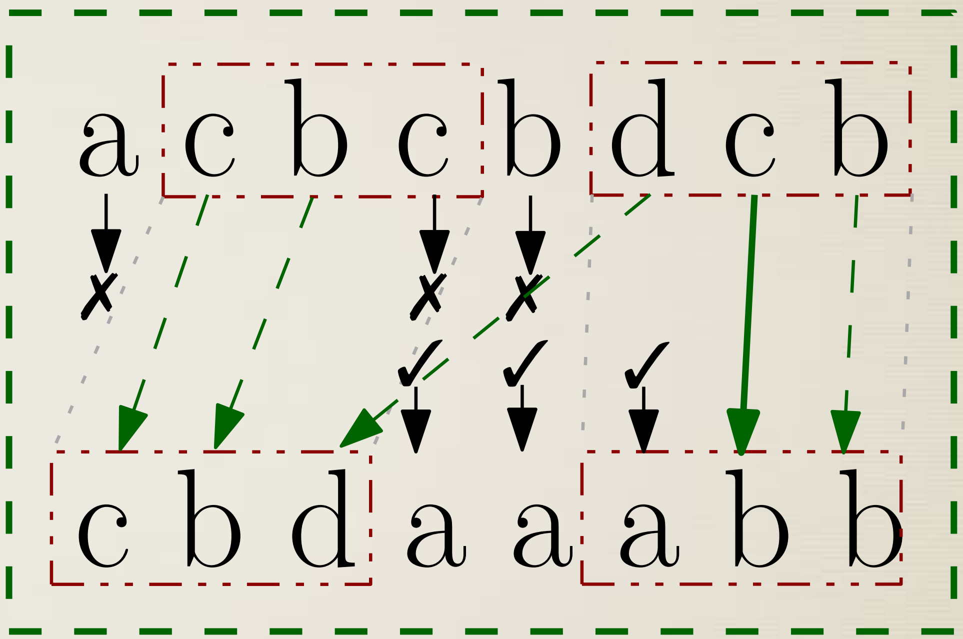


(b) The transformation is not window-compatible since character 5 of the second string is old but doesn't lie in any windows.

What is a windows-compatible transformation?



(c) The transformation is not window-compatible since character 1 of the second string is old but prior to the transformation, it was not placed in any windows.

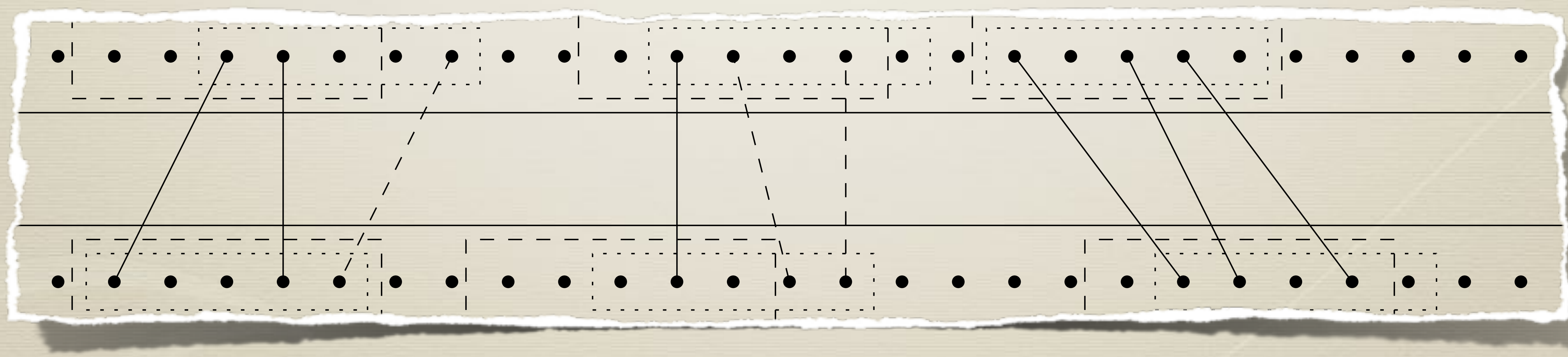


(d) The transformation is not window-compatible since character 3 of the second string is old but prior to the transformation, it was not placed in the corresponding window.

How good is the best windows-compatible transformation?

* If edit(S1, S2) ≤ δn, there exists a window-compatible transformation of S1 into S2 with at most $3\delta n + n/\gamma + 2l$ operations.

* Proof idea: substitution and intact links / construct windows / shift constructed windows to actual windows



How do we find the best windows-compatible transformation?

- Using dynamic programming
  - Time complexity: $O(n+|W_1||W_2|)$
  - If we have all edit distances between windows
- How can we find all edit distances between windows?

Metric Estimation

Metric Estimation

* For any set of **m** strings like M, <M, edit> forms a metric system

* We give two approximation algorithms:

	Approximation Factor	Query Complexity	Time Complexity
Algorithm I	$3+\epsilon$	$\tilde{O}(m^{5/3}\mathsf{poly}(1/\epsilon))$	$O(m^2\mathsf{poly}(1/\epsilon))$
Algorithm II	$O(1/\epsilon)$	$\tilde{O}(m^{3/2+\epsilon}\mathsf{poly}(1/\epsilon))$	$O(m^2\mathsf{poly}(1/\epsilon))$

Metric Estimation: Algorithm I

* The first idea: discretize the problem
* The second idea: solving for a single threshold as a graph
* The third idea: low degree and high degree vertices

Metric Estimation: Algorithm I (cont.)

* The first idea: discretize the problem
  * Intervals of the form $[x, (1 + \epsilon/3)x]$
    * $[(1 + \epsilon/3)^{k-1}, (1 + \epsilon/3)^k]$
  * $\log_{1+\epsilon/3} U = \tilde{O}(\mathsf{poly}(1/\epsilon))$ disjoint intervals
  * Add a $(1+\epsilon/3)$ term to the approximation factor

Metric Estimation: Algorithm I (cont.)

- The second idea: threshold
  - Given a threshold t, find all pairs (p_i,p_j) such that $d(p_i,p_j) \leq t$
  - With some false positives $d(p_i,p_j) \leq 3t$
    - A term of 3 to approximation factor
  - No false negative

Metric Estimation: Algorithm I (cont.)

- The third idea
  - For low degree ($<m^{1/3}$) vertices, find all neighbors via the Grover's search, then remove it
  - For high degree ($\geq m^{1/3}$) vertices, find all neighbors $N(v_i,t)$, then find all neighbors with threshold 2t, $N(v_i,2t)$
    - Connect the two sets, all of the neighbors of $N(v_i,t)$ is in $N(v_i,2t)$, and they're not far away.
    - Remove v and $N(v_i,t)$

Metric Estimation: Algorithm I Analysis

* Amortized analysis (query complexity)
  * For low degree ($<m^{1/3}$) vertices, we spend $O(\sqrt{m}.m^{1/3})=O(m^{2/3})$ for 1 vertex
  * For high degree ($\geq m^{1/3}$) vertices, we spend $O(m)$ for at least $O(m^{1/3})$ vertices, or $O(m^{2/3})$ for 1 vertex
* $O(m^{5/3})$ for a fixed threshold
* $O(m^{5/3}\text{poly}(1/\epsilon))$ for Algorithm 1 (which is truly subquadratic)

Edit Distance

Edit Distance

- Compute the edit distance of S_1 and S_2
- We use Algorithm I and Algorithm II as a subroutine

	Approximation	Time and Query Complexity
Algorithm III	$7+\epsilon$	$\tilde{O}(n^{2-1/7}\mathsf{poly}(1/\epsilon))$ $= O(n^{1.857})$
Algorithm IV	$O(1/\epsilon)^{O(\log 1/\epsilon)}$	$\tilde{O}(n^{2-(5-\sqrt{17})/4+\epsilon}\mathsf{poly}(1/\epsilon))$ $= O(n^{1.781})$

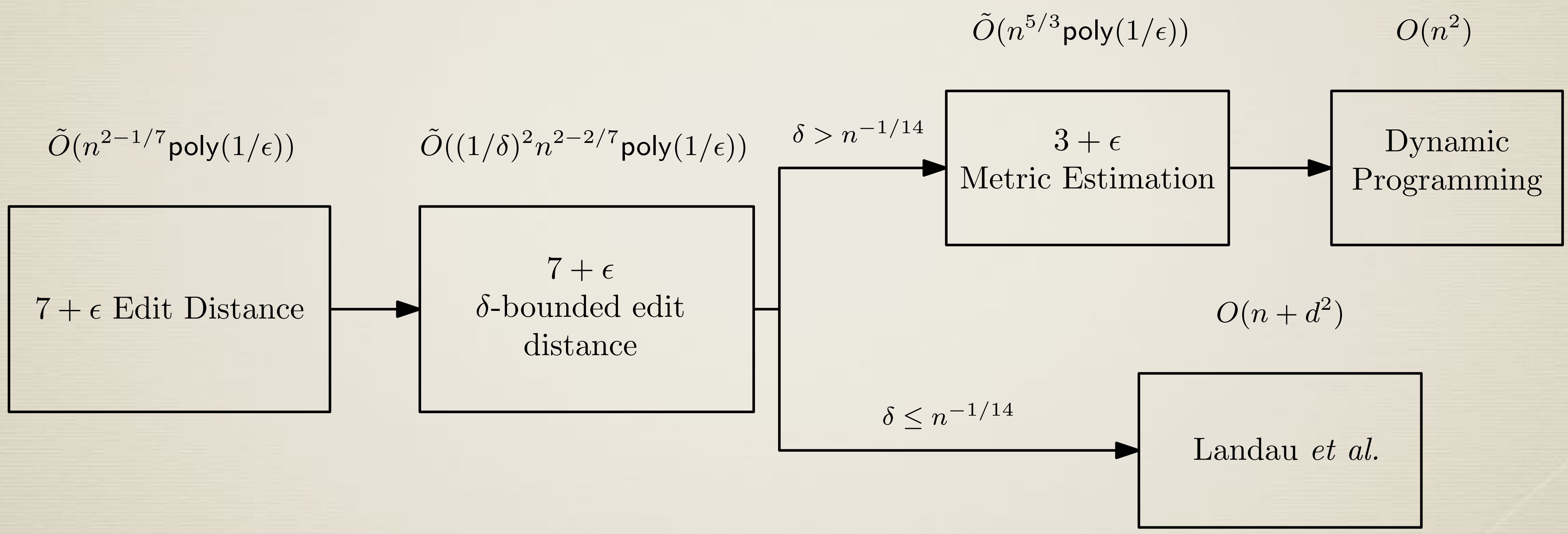
Edit Distance: δ-bounded edit distance

* Solve the δ-bounded edit distance
* Guarantee: $\mathrm{edit}(S_1, S_2) \le \delta(|S_1| + |S_2|)$
  * For $\delta \le n^{-1/14}$ we run an exact algorithm of $O(n+d^2)=O(n^{2-1/7})$
  * So we can assume $\delta > n^{-1/14}$
* Note: δ does not affect the approximation factor, it affects the time, so we tune the parameters with δ!

Edit Distance: Algorithm III

* For a $\delta > n^{-1/14}$
* Construct some windows of S_1 and S_2 where the number of windows and the size of them are $o(n)$
* Solve the metric estimation problem for this windows using Algorithm I and the classical dynamic programming as the distance oracle
* Find the best windows-compatible transformation

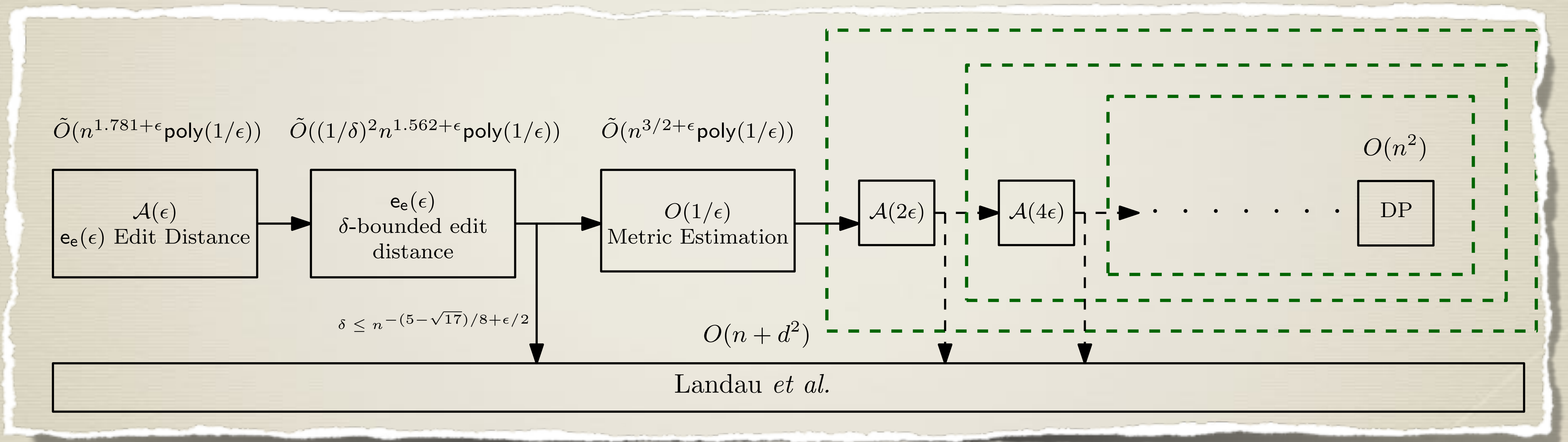
Algorithm III



Metric Estimation: Algorithm II

* Approximation factor $O(1/\epsilon)$
* Query Complexity $\tilde{O}(m^{3/2+\epsilon}\,\mathrm{poly}(1/\epsilon))$
* Time Complexity $\tilde{O}(m^2\mathrm{poly}(1/\epsilon))$
* Idea: handle high degree vertices with a hitting set (random sampling) and solve them recursively

Algorithm IV: Bootstraping



Edit Distance (recall)

	Approximation	Time and Query Complexity
Algorithm III	$7+\epsilon$	$\tilde{O}(n^{2-1/7}\mathsf{poly}(1/\epsilon))$ $= O(n^{1.857})$
Algorithm IV	$O(1/\epsilon)^{O(\log 1/\epsilon)}$	$\tilde{O}(n^{2-(5-\sqrt{17})/4+\epsilon}\mathsf{poly}(1/\epsilon))$ $= O(n^{1.781})$

Any question?

Thank you for your time