CS7800: Advanced Algorithms

Soheil Behnezhad

Approximating Edit Distance in Subquadratic Time

Resources:

- Blog post by Aviad Rubinstein: https://theorydish.blog/2018/07/20/approximating-edit-distance/
- **[BEGHS]** "Approximating Edit Distance in Truly Subquadratic Time: Quantum and MapReduce" by Boroujeni, Ehsani, Ghodsi, HajiAghayi, Seddighin
- **[CDGKS]** "Approximating Edit Distance Within Constant Factor in Truly Sub-Quadratic Time" by Chakraborty, Das, Goldenberg, Koucky, and Saks

(Approximate) Edit Distance

- Given two strings $x \in \Sigma^n, y \in \Sigma^n$, the **edit distance** is the number of insertions, deletions, and swaps required to turn x into y.

$ED(x, y)$

- Given an alignment, the cost $\mathrm{EDIT}(x, y)$ is the number of positions where the two strings don't agree

o	c		u	r	r	a	n	c	e
o	c	c	u	r	r	e	n	c	e

$\alpha \geq 1$

- In this lecture, we consider approximating the edit distance problem. We say $\widetilde{\mathrm{EDIT}}$ α-approximates $\mathrm{EDIT}(x, y)$ if:

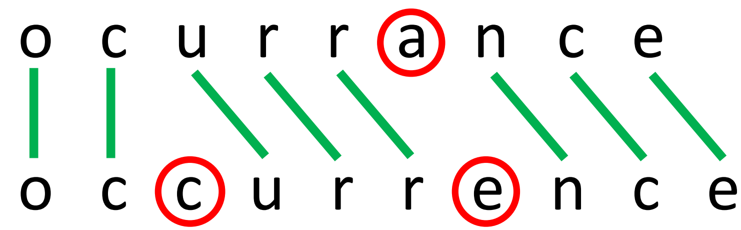
$\alpha = 1$ exact

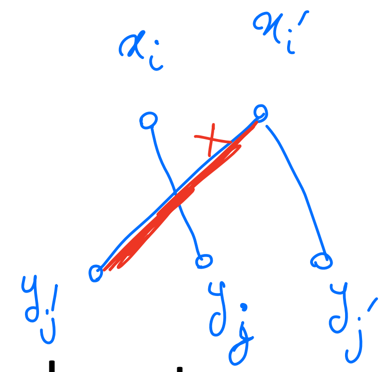
$$\mathrm{EDIT}(x, y) \leq \widetilde{\mathrm{EDIT}} \leq \alpha \cdot \mathrm{EDIT}(x, y)$$

- In our last lecture, we say that Edit Distance can be solved in $O(n^2)$ time. This remains the best known exact algorithm! But can we approximate it faster?

Idea 1: Non-crossing Matchings

- Instead of *alignments*, it will be more convenient to work with *non-crossing matchings*.





- A non-crossing matching matches characters of x and y s.t. :
  1. **It only matches identical characters:**
     For any $x[i]$ and $y[j]$ that are matched, $x[i] = y[j]$
  2. **The matching is non-crossing:**
     If $(x[i], y[j]), (x[i'], y[j'])$ are both in the matching and $i' > i$ then we also have $j' > j$.

Claim:
The number of unmatched characters of x and y in the largest non-crossing matching is a 2-approximation of $\mathrm{EDIT}(x, y)$.

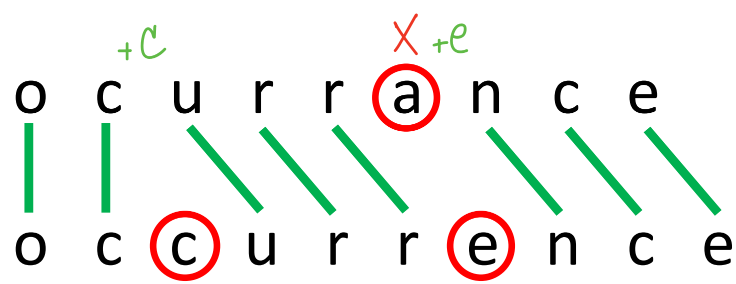
Idea 1: Non-crossing Matchings

Claim: Let $\widetilde{\mathrm{EDIT}}$ be the number of unmatched characters of x and y in the largest non-crossing matching. Then:

$$\mathrm{EDIT}(x,y) \le \widetilde{\mathrm{EDIT}} \le 2 \cdot \mathrm{EDIT}(x,y)$$

Pf: ($\mathrm{EDIT}(x,y) \le \widetilde{\mathrm{EDIT}}$)

Delete every unmatched character of x, and insert every unmatched character of y.



occurrence

Idea 1: Non-crossing Matchings

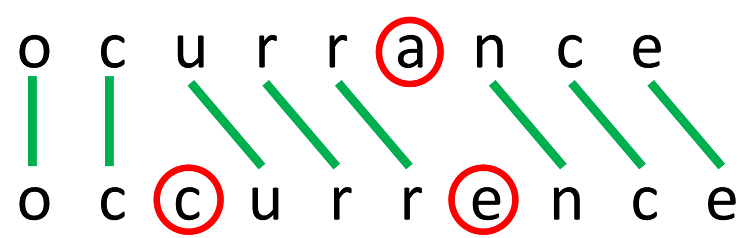
Claim: Let $\tilde{E}$ be the number of unmatched characters of x and y in the largest non-crossing matching. Then:

$$\mathrm{EDIT}(x,y) \le \widetilde{\mathrm{EDIT}} \le 2 \cdot \mathrm{EDIT}(x,y)$$

Pf: $(\widetilde{\mathrm{EDIT}} \le 2 \cdot \mathrm{EDIT}(x,y))$

Take the best alignment and match the characters in identical columns.

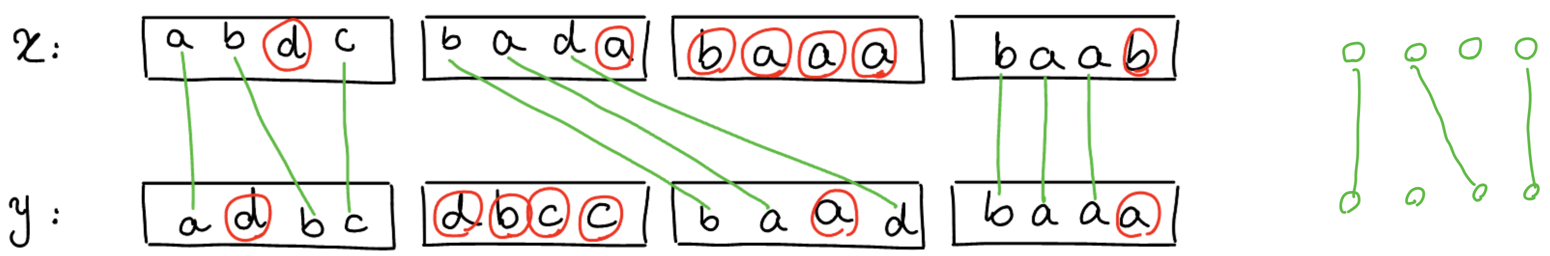
o	c		u	r	r	a	n	c	e
o	c	c	u	r	r	e	n	c	e



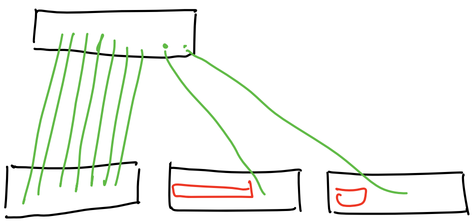
Idea 2: Window Compatibility

Let's partition x and y into t consecutive substrings--aka windows--of length $\ell := n/t$ each.

We say a matching is window-compatible if there are no two characters in the same window that are matched to two different windows.

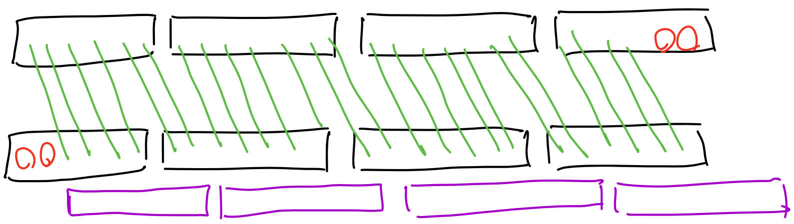


To constant-approximate edit distance, it suffices to find the largest non-crossing window-compatible (NCWC) matching [BEGHS].



If the largest non-crossing matching matches the characters of one window to ≥ 3 other windows, we can afford to leave all of its characters unmatched

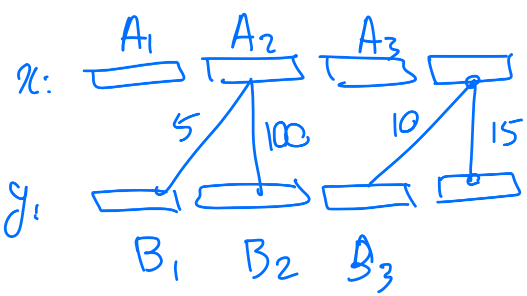
The argument is much more tricky if the characters of a window A_i are matched to two consecutive windows B_j, B_{j+1}.



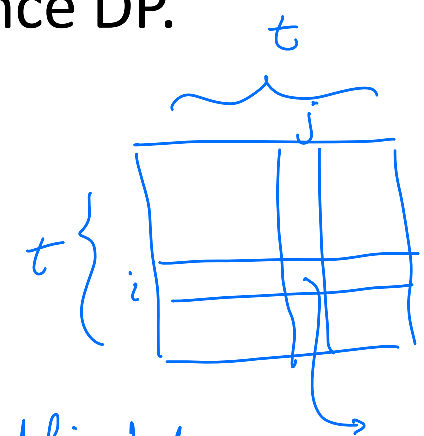
Finding NCWC Matching

Claim: Given the ~~edit distance~~ largest NC matching between any two windows, we can find the (apx) largest NCWC matching in $O(t^2)$ time.

Idea: Solve a weighted version of edit distance DP.



$t \ll n$



$DP[i][j]$: # unmatched characters in the largest NCWC matching between $A_1, \dots, A_i$ and $B_1, \dots, B_j$

$$DP[i][j] = \min \left\{ DP[i-1][j] + \frac{n}{t},\ DP[i][j-1] + \frac{n}{t},\ DP[i-1][j-1] + ED(A_i, B_j) \right\}$$

Computing window pair distances

Computing exact ED between any pair of windows takes

$$\underbrace{t^2}_{\text{\# pairs}} \times \underbrace{\ell^2}_{\text{time per pair}} = t^2 \times \left(\frac{n}{t}\right)^2 = n^2$$

$\ell = \frac{n}{t}$

time! So no progress so far. ☹

$\ell = \frac{n}{t}$

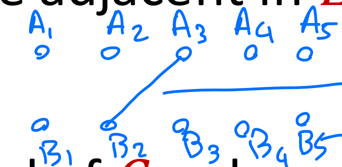
New Goal: *Approximate* ED between all the pairs.

Idea 3: Approximating Window Distances

$\{0, \dots, 2\ell\}$

Given a threshold τ, define G_τ to be the bipartite graph over the windows such that two windows A_i and B_j are adjacent iff $ED(A_i, B_j) \le \tau$.

apx of G_τ $\qquad$ G_τ



iff $ED(A_3, B_2) \le \tau$

It will be easier to compute $\widetilde{G_\tau}$ instead of G_τ where for any (A_i, B_j):

also satisfied by G_τ

- If $ED(A_i, B_j) \le \tau$: $\qquad$ (A_i, B_j) is an edge in $\widetilde{G_\tau}$.
- If $ED(A_i, B_j) > 10\tau$: $\qquad$ (A_i, B_j) is not an edge in $\widetilde{G_\tau}$.
- If $\tau < ED(A_i, B_j) \le 10\tau$: $\qquad$ (A_i, B_j) may or may not be an edge in $\widetilde{G_\tau}$.

any edge in G_τ is also an edge in $\widetilde{G_\tau}$

Claim: To constant approximate all the pairwise distances between the windows, it suffices to compute $\widetilde{G_\tau}$ for $O(\log n)$ values of τ.

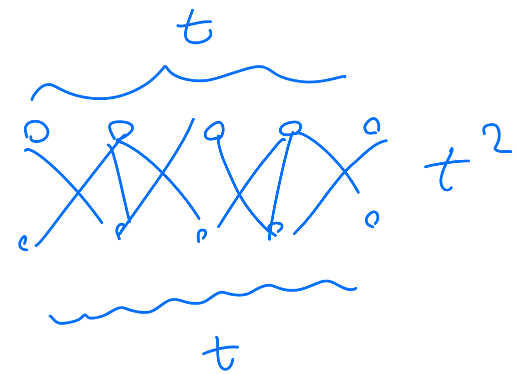
pf: Consider any $\tau \in \{0, 1, 2, 4, 8, 16, \dots, n\}$, and find $\widetilde{G_\tau}$ for each.

Let $\widetilde{ED}(A_i, B_j)$ to be the smallest τ s.t. $(A_i, B_j) \in \widetilde{G_\tau}$.

$$\frac{\tau}{2} \le ED(A_i, B_j) \le 10\tau$$

Idea 3: Approximating Window Distances

Goal: Compute $\widetilde{G_\tau}$ for given τ.



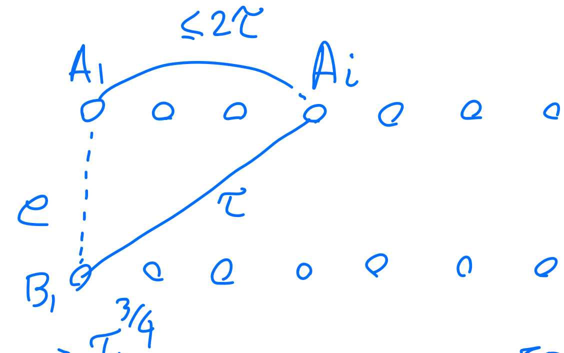
We study two cases separately:

- **Dense Case:** If G_τ has at least $t^{7/4}$ edges.
- **Sparse Case:** If G_τ has at most $t^{7/4}$ edges.

Idea 3: Approximating Window Distances

BLUE: G_τ

- **Dense Case:** If G_τ has at least $t^{7/4}$ edges.



An "average" edge in G_τ has $\geq \frac{t^{7/4}}{t} = t^{3/4}$ edges incident to it.

ALG:

Repeat this for $\tilde{O}(t^{1/4})$ steps:

Total time: $t^{\frac{1}{4}} \cdot \frac{n^2}{t} = \frac{n^2}{t^{3/4}}$

$ED(A_i, B_1) \leq \tau$

$ED(A_i, A_1) \leq 2\tau$

$ED(A_1, B_1) \leq \tau + 2\tau = 3\tau$

Take one random window A_i

Compute $ED(A_i, B_j)$ for all j

Compute $ED(A_i, A_j)$ for all j

$\Big\} \; 2 \times t \times \left(\frac{n}{t}\right)^2 = O\left(\frac{n^2}{t}\right).$

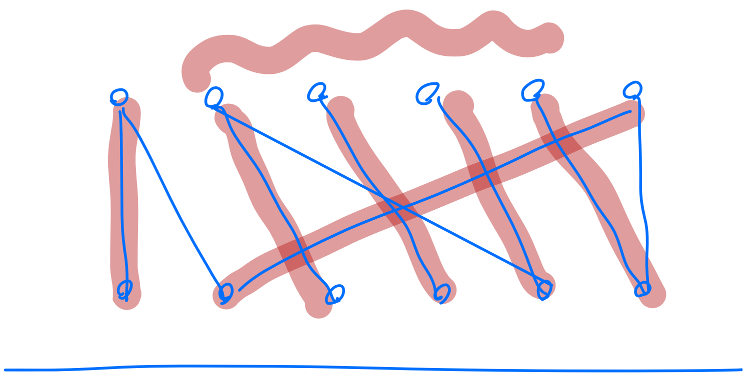
If for a pair A_j, B_k we know $ED(A_j, B_k) \leq 3\tau$, we add it as an edge to $\tilde{G}_\tau$.

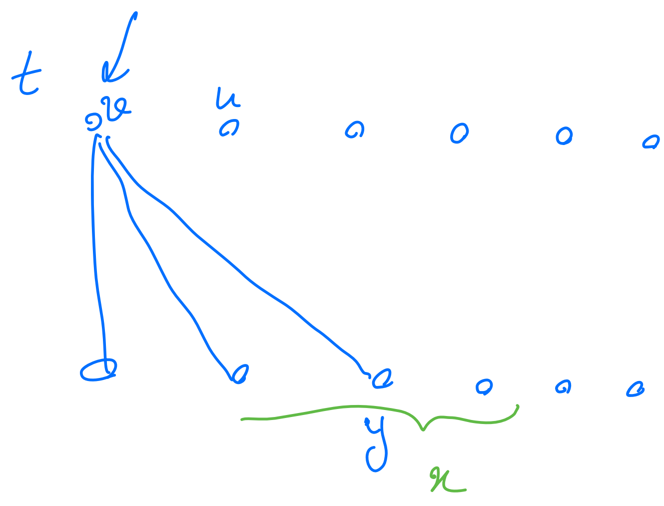
based on distances of A_i

Fix any edge $(A_1, B_1) \in G_\tau$. Every choice of A_i discovers (A_1, B_1) w.p. $\geq \frac{t^{3/4}}{t} = t^{-1/4}$. Therefore, it suffices to repeat for $\tilde{O}(t^{1/4})$ iterations to discover all "average" edges.

Idea 3: Approximating Window Distances

- **Sparse Case:** If G_τ has at most $t^{7/4}$ edges.





Key insight: Take two windows v and u that are close (in position), then it suffices to make ED query calls only for pairs (u, x) where (v, y) is an edge and x is close (in position) to y.

Summary

The edit distance can be solved in $O(n^2)$ time exactly.

no $2^{0.99n}$ time algo for 3SAT

Under a plausible conjecture (Strong Exponential Time Hypothesis – SETH) quadratic time is almost optimal for exact algorithms.

ETH: no $2^{o(n)}$ time

But the edit distance can be constant-approximated in subquadratic time.

OPEN:

Is it possible to $(1+\epsilon)$-approximate edit distance in subquadratic time?

First step: Beat 3-apx.