CS 594: Representations in Algorithm Design **Spring 2022**

Lecture on 03/10/2022

Lecturer: Xiaorui Sun *Scribe: Pranavi Alle*

1 Last Lecture's Review

• In the last lecture, we reviewed about Graph Representation and its properties that include Generalization, Decomposition and Sparsification.

2 Today's, Lecture:

• Approximate of Edit Distance the way it is solved in Dynamic Program.
• Faster algorithm for Edit Distance and it's approximation factor and the way it is evolved over the years.

3 Approximate Of Edit Distance:

3.1 Algorithm

Given Two strings S1 and S2 and the below operations that can be performed on S1. Find the minimum number of edits(operations) required to convert S1 into S2.
The operation are:
• Add a char to S1
• Delete a char to S1
• Replace a char from S1

3.2 Example

There are two strings S1=CTACCG and S2= TACATG. Three operation to be performed that is add, delete , replace operations are:

- **DELETE** first character TACCG
- **ADD** 'A' after 3rd character TACACG
- **REPLACE** 5th character i.e 'C' with 'T' TACATG

In this way the recursive operations i.e add, delete and replace operations are done to make the two strings equal.

4 Dynamic Approach for Edit Distance

4.1 Definition of DP:

Dynamic Programming (DP) is an algorithmic technique for solving an optimization problem by breaking it down into simpler subproblems and utilizing the fact that the optimal solution to the overall problem depends upon the optimal solution to its subproblems.

4.2 Edit Distance in DP:

Edit distance in dynamic program is represented in d(i,j)
where d(i,j)= edit distance between substring of first i character in X and the substring of the first j character in Y.

- In 2-dimensional function d(i,j) is used in dynamic approach for all possible i and j $\mid S1 \mid = \mid S2 \mid = n$
- Edit Distance is always seen in the view point of String Alignment.

Example: X=CTACCG, Y=TACATG
Throughout 3 operations some characters in S1 are never changed or touched in the above example TAC are never touched is called **"not arbitrarily matching"**. Another property is matching is 'A' can be matched after T i.e 'TA' order is preserved it is called **"order preserve matching"**.

While choosing alignments:

$$\boxed{Editdistance \leq (n - sizeoflargestalignment)}$$

- Edit Distance between Partial substring in optimal alignment:

Four Possibilities:

- Xi is matched to Yi in optimal alignment

Example: i=3 “CTA” , j=2 “TA”
Last two characters get matched.

$$\boxed{d(i,j) = d(i-1,j-1)+1}$$

- Xi is removed

Example: i=3 ”CTA”,j=2 ”CT”
A should be removed to get an optimal solution

$$\boxed{d(i,j) = d(i-1,j+1)}$$

- Add some Characters after Xi

Example: i=3 ”CTA”,j=3 ”TAC”
ADD new character to the first string

$$\boxed{d(i,j) = d(i,j-1)+1}$$

- Replace Xi by Yi

Example: i=5 ”CTACC” ,j=4. ”TACA”
Last character in Xi is replaced by ‘A’ character

$$\boxed{d(i,j) = d(i-1,j-1)+1}$$

$$d(i,j) = \begin{cases} j & \text{if } i = 0 \\ i & \text{if } j = 0 \\ min(d(i-1,j-1)+w+d(i-1,j)+1+d(i-1,j-1)+1) & \text{if } i \geq 0, j \geq 0 \\ & w = 0 if x_{\text{i}} = j_{\text{i}}, else, w = 1 \end{cases}$$

According to the above equation:

```
fori=0 to n:
forj=0 to n:
   d(i,j)
```

Time Complexity – O(n^2)

In particular this algorithm is called **exact algorithm** if you apply this algorithm, we will get the exact edit distance

4.3 Faster Algorithm for EDIT distance:

- A lot of people from 1960's spent a lot of time for improving the algorithm but couldn't come with an significant solution .
- If we ask exact algorithm for edit distance $O(n^2)$ then tiny improvement $O(n^2/logn^2)$ for $\mid E \mid$=O(1) result obtained by Masek peterson in 1980.
- According to pivot, SETH (i.e Strong Exponential Time Hypothesis), If SETH is true then there is no algorithm solved in edit distance in time $n^{2-O(1)}$ .Edit distance cannot be solved in time $n^{1.999}$
- 3 SAT problem cannot be solved in $2^{O(n)}$
- 3 SAT problem is supposed to be an harder problem i.e NP Complete

4.4 Approximation Factor:

$$ApproximationFactor = \frac{editdistanceestimationbyalgorithm}{Optimaleditdistance}$$

- Approximation Factor is always greater than 1.
- The Goal is to minimize the Approximation factor for worst case.
- In 2000's people developed an algorithm with $(logn)^{O(1/e)}$ approximate algorithm with running time of $O(n^{(1+E)})$ for any E greater than eqaul to 0. This result is obtained by Andoni Krauthgamer (ONAK) in the year 2008
- In 2018 an algorithm is developed with O(1) approximate algorithm in time O($n^{1.618}$)*by* $ChakrabortyDasGoldenbergKonchySales$.
- In 2020 developed an another algorithm $2^{O(1/E)}$ approximate in time $O(n^{1+e})$ is developed by Andoni Nasatzki.

5 IDEA About 2018 Algorithm:

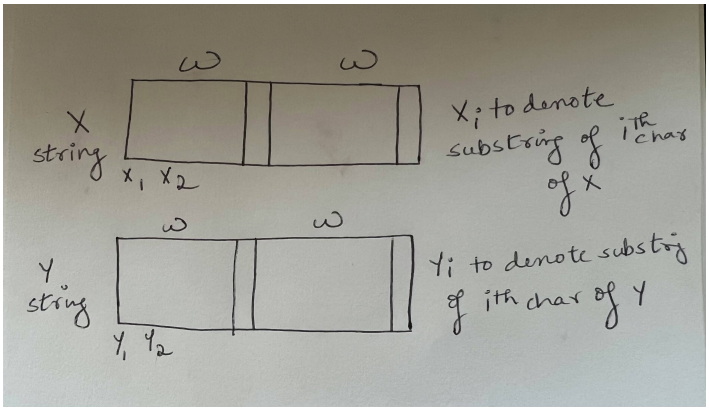
- Replacement operation is not used because it increases the edit distance.
- Only allows insertion and deletion.
- Edit Distance become optimal if input string are of equal or same length.

5.1 Alignment:

π: [n] $->$ [n] U 1
Each position is aligned with another character with special symbol. With a restriction such that π and not equal to 1 and $\pi(i)$ less than $\pi(j)$. Among all the alignments this is the best one.

5.2 1st Idea (SUBSTRING ALIGNMENT)

- Generalize alignment for substring.
- More specifically we break two strings i.e X and Y into substring of fixed length w approximately $n^{0.9}$



- We can have strings matching from one element from one substring to the some other substring present. Example is provide below:

Example: i=5 "CTACC",j=3 "TACA"
Here CTA matches aligns with TA

- Here we assume that Xi is approxiamately equally to Yi

$$EditDistance(X,Y) = \frac{min(editdistance(Xi,Yi)}{w}$$

THE END