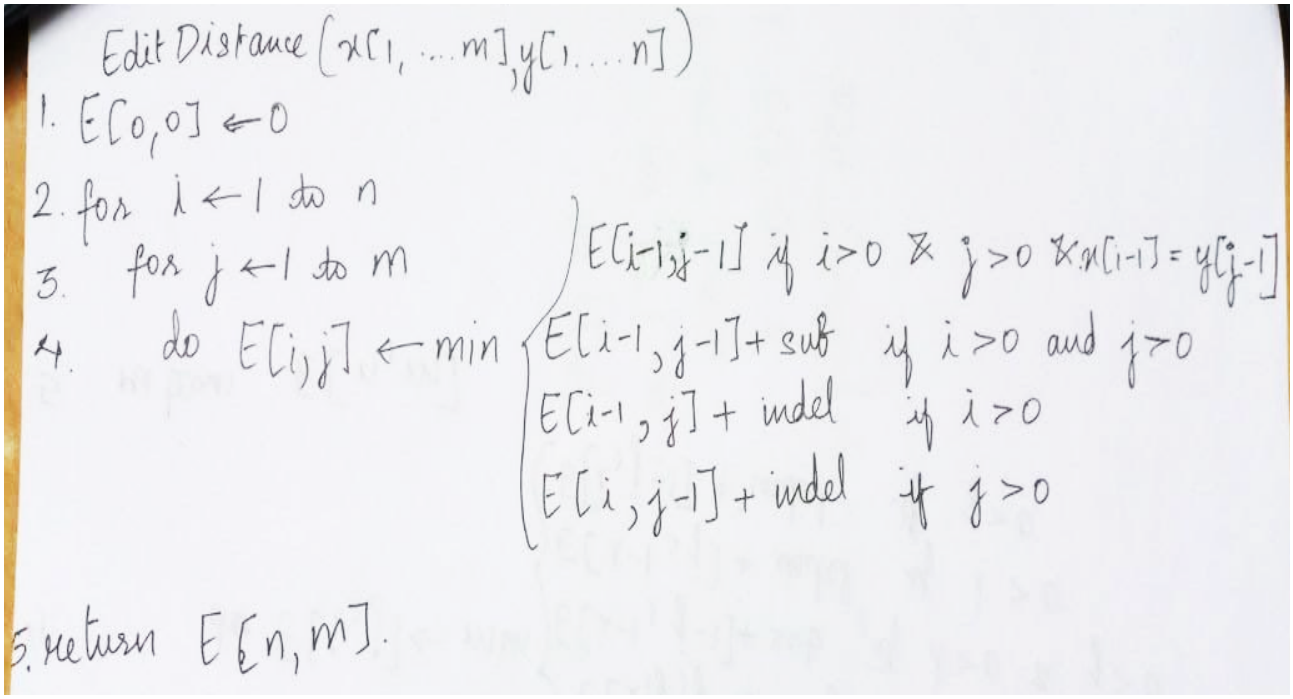
ASSIGNMENT
SHAMBHAVI SUD
1910110365

Edit Distance using Dynamic Programming

I construct matrix E, here every entry is $cost_{x,y}(i,j)$. Such that each value of $cost_{x,y}(i,j)$ depends on $cost_{x,y}(i',j')$ where i'<=i and j'<=j. Using the below Algorithm we calculate the elements of matrix E.



Edit Distance Algorithm Used

<u>Substitution Cost :</u> Calculate the edit distance in between the initial i-1 characters of start string and the first j-1 characters of target string and then do the addition of 1 for replacing the j th character of target with the i th character of start string , if they are different.

Deletion Cost: Calculate the edit distance between the initial i-1 characters of start string & the initial j characters of target string & later increment by 1 for deleting the i th character of start string .

Insertion Cost: Calculate the edit distance between the initial i characters of start and the initial j-1 characters of target string and then increment by 1 for insertion of the j th character of target .

When we visualise this calculation, we see that Dynamic Programming must evaluate the sub-problems in the correct sequence, i.e in bottom - up manner and left to right inside each row. A nested for loop computes the minimal value for each of the sub-problems in order until all elements in matrix E have been calculated. This method is not recursive;but, it employs the results of previous calculations to solve smaller issues.

Time Complexity = $\theta(mn)$
Space Complexity = $\theta(mn)$

Optimization of Edit Distance Algorithm

Space Complexity: As per the problem we were given the string lengths as 600, however had they been 2000, the above Algorithm wouldn't have been useful as it can make 2D matrix of 2000 x 2000 dimension and that would take like a lot of space. Only one row, the upper row, is required to complete a row in a Dynamic Programming array. For example, we only need data from the 9th row to fill the i = 10 rows in the Dynamic Programming array. As a result, we just make a Dynamic Programming array with 2 x start_string length. This technique minimises the complexity of the space, to a linear space complexity of $\theta(m)$.

Time complexity:

Improving upon the data structure used.

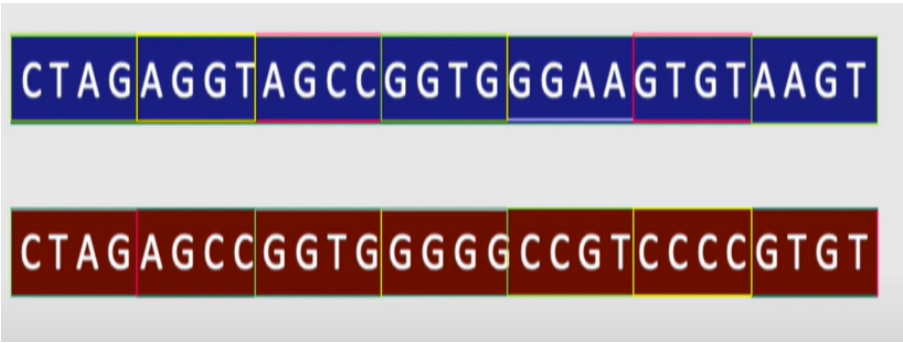
When we implement strings using arrays, we see that insertion and deletion take θ(n) respectively. Edit distance contains 3 operations namely substitute, insert and delta. These can be done in O(1) time using 2 stacks P and Q. I demonstrate the Algorithm below

Substitute with str- **if**(isNotEmpty(Q)) Pop(Q) & Push(str,P)
Delete - if(!isEmpty(Q)) - Pop(Q)
Insert str - Push(P,str)

Each Stack action takes O(1) along with that O(1) is required for each transformation and hence each action takes O(1) time.

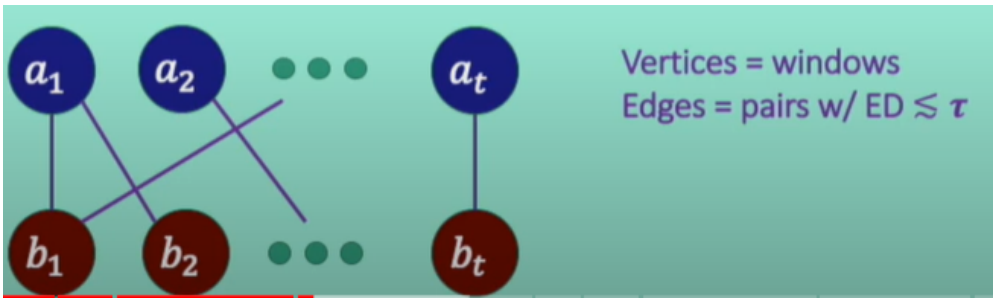
Edit Distance Optimization in nearly Linear Time

1. Divide the strings into windows, into block substring of length d = N/t



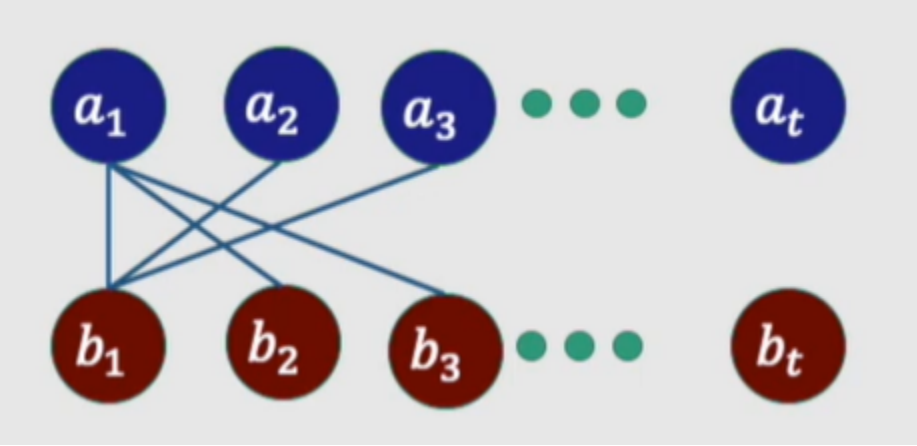
This way we could go onto to Delete windows, Match them, or Insert windows using strings.
Assuming two su
Let b, t be the blocks of windows in strings B, T. Lets assume a graph MΔ = , if edit distance between two windows is less than or equal to D then there exists a edge between them.

Given a dense graph we try to find the edge from one window to another in the following manner. We take a bipartite graph with vertices representing windows.

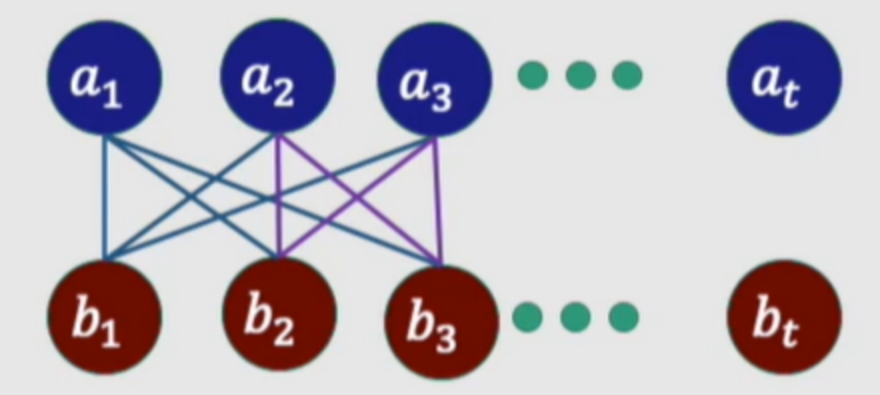


We discover edges in this graph by querying some of the edges and then applying some deduction rules .the idea is we want to discover edges that are approximately within this threshold system . to include edges of distance 3 tau and we also want to find enough edges that we accurately compute the aggregate at a distance.

2. <u>Dense Graphs using Triangular Inequality</u>
   We want to find enough good edges using a bipartite graph and some methods of querying these edges.



Lets take edge a1 and b1 and take time t to find a1's and b1's neighbours respectively. We take all the pairs such that we get a clique.



Triangle inequality:

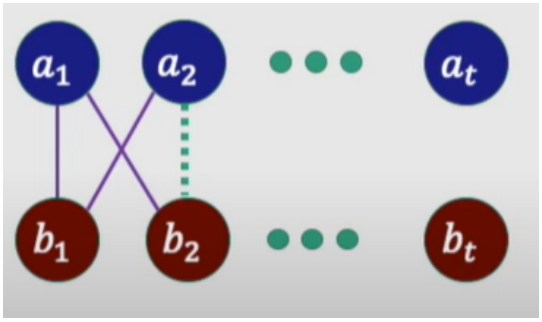
$(u, v) \in \mathcal{N}(a_1) \times \mathcal{N}(b_1)$ $\mathrm{ED}(u, v) \leq 3\tau$

If we assume that each of the t vertices has degree Δ then using t queries we can discover Δ^2 becos the clique is of size Δ because degrees of a1 and b1 are Δ .After doing the match we reach the following assumptions.

Want to discover $t\Delta$ edges
Query per edge $\approx t/\Delta^2$
Total: $t\Delta \cdot t/\Delta^2$

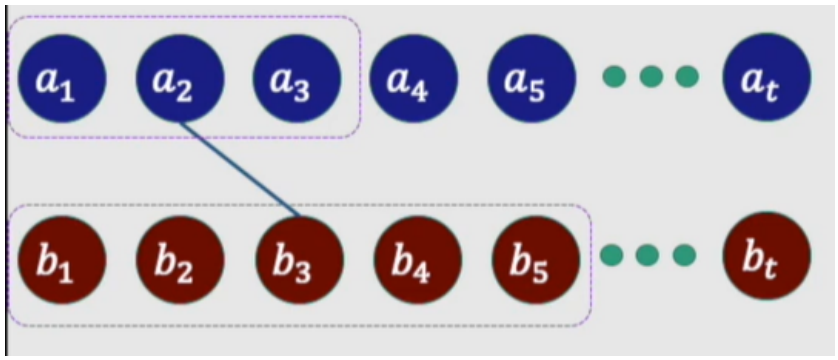
Total: $\approx \frac{t^2}{\Delta}$ queries

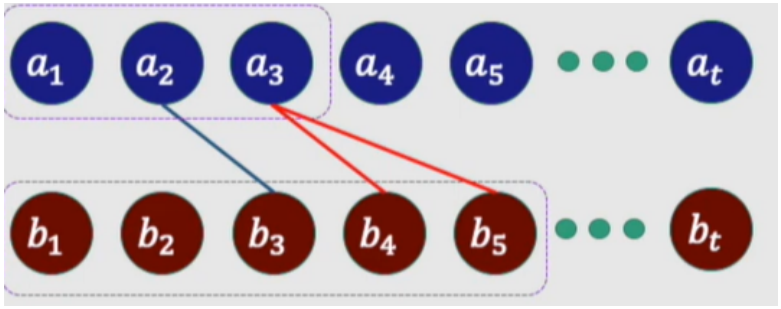
This step uses Triangular Inequality as a1 is connected b1, b1 to a2 and a1to b2 we deduce that a2 is connected to b2. Due to this we lose a factor of 3 in approximation.



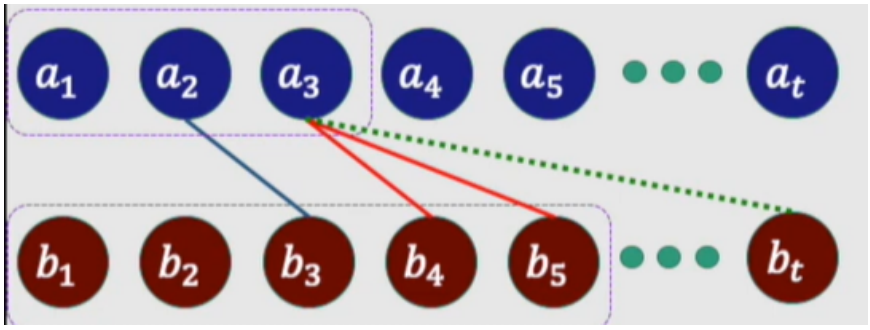
3. Sparse Graphs using “seed & expand ”

Here starting with and edge between a2 and b3 we are going to consider 3 intervals around a2 and b3





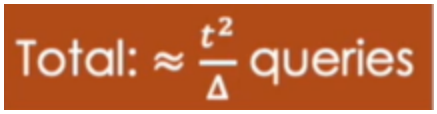
Coming to the major optimization we look at a3 which is just beside a2 and hence we only need to check things that are beside b3.There exists a near optimal matching known as low skew where no pair of edge is going to diverge too much from each other. Ex: a3 and b_t.



If we consider t queries then we can learn everything over an interval of size t/Δ . we get the below result.

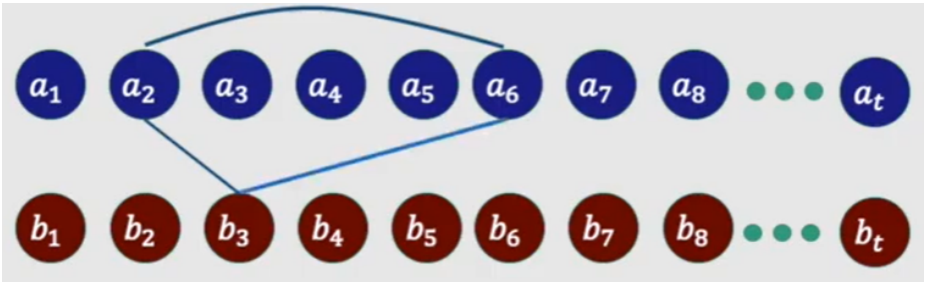
Total: $\approx t\Delta$ queries

So if we multiply the below results we get queries = $t^{1.5}$



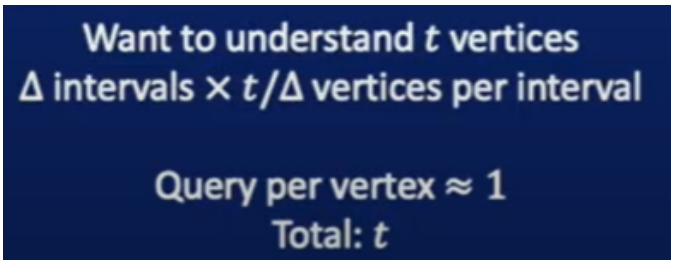
To be able to get near linear time , we need to get number of queries to near linear.

4. Union of disjoint bi-cliques using 1+2
   Here we take a2 and find all its neighbours on both sides. Hence using triangular inequality we find that a6 is also close to b3.



   Next we consider intervals around a6 and b3. The main idea here is that for a6 we only need to consider neighbours that are close to b3. So for things that are close to a6 like a5 and a7 , we will only need to look at things that are close to b3 like b2 and b4.

   Analysis : here we do t queries and learn in intervals of t/Δ. Now because the degree is delta we get delta intervals and t times delta vertices per interval and **we reach a query time per vertex of 1**. SO… we get about t interval queries.



Resources

[1] Constant-factor approximation of near-linear edit distance in near-linear time
[2] Constant factor approximations to edit distance on far input pairs in nearly linear time