Advanced algorithms
Master1 WI/MLDM

Project-2015-2016

Different Algorithm and Optimization Approaches for the Solution of Edit Distance Problem

By

Amine Amyar, Emin Avci

Abstract

In this article, we study the behavior of different programming methods (algorithms and optimization approaches) for the edit distance problem; such as recursive, dynamic programming, divide & conquer etc. We show that those different algorithms try to solve the problem in different approaches and different running times with same inputs.

The edit distance between two strings word1 and word2 is defined to be the minimum number of character(operation) inserts, deletes and replaces needed to convert word1 to word2. Given a text string word1 of length n, and a pattern string word2 of length m, informally, the string edit distance matching problem is to compute the smallest edit distance between word2 and substrings of word1.

The problem of finding the edit distance between two sequencesare important problems with applications in many domains like virus scanners, security kernels, natural language translation and genome sequence alignment.

INTRODUCTION

The edit distance between two strings is the number of insertions, deletions, and substitutions needed to transform one string into the other. This distance is of key importance in several fields such as computational biology and text processing, and consequently computational problems involving the edit distance were studied quite.

Calculation of edit distances between two sequences and aligning the two sequences based on their edit distances is a very compute intensive process. This problem becomes n-fold when the number of sequences on which these operations have to be done are huge.

We have tried to implement a solution in 6 different algorithms as mentioned below:

- The recursive version
- The classical algorithm based on dynamic programming
- Dynamic programming and divide and conquer approaches
- A branch-and-bound version of the recursive approach
- An approximated version based on a greedy approach
- An approximated randomized version
- An approximated version of the classical dynamic programming approach where one fills only a stripe of size k around the diagonal of the matrix.
- k-means algorithm

In each of algorithm implementations for the solution of Edit Distance problem, we observed different approaches and different behavior. We observed that some algorithms (such as recursive) was doing some steps which should not be done in an efficient algorithm. However we tried to

make other implementation (such as dynamic programming, divide & conquer etc.) to have more efficient solutions.

1. **Recursive Version**

In our algorithm of recursive version, we keep recursing until we hit an empty string, taking one character off each time. If we call the lengths of the stringsword1 and word2 as m and n respectively, then the depth of the tree is min(m, n).

At every node in the call tree except the leaves, you recurse exactly three times, so in the limit the average branching factor is 3. That gives us a call tree of Θ(3^min(m, n)) nodes. The worst case occurs when m = n, so we can call that Θ(3^n).

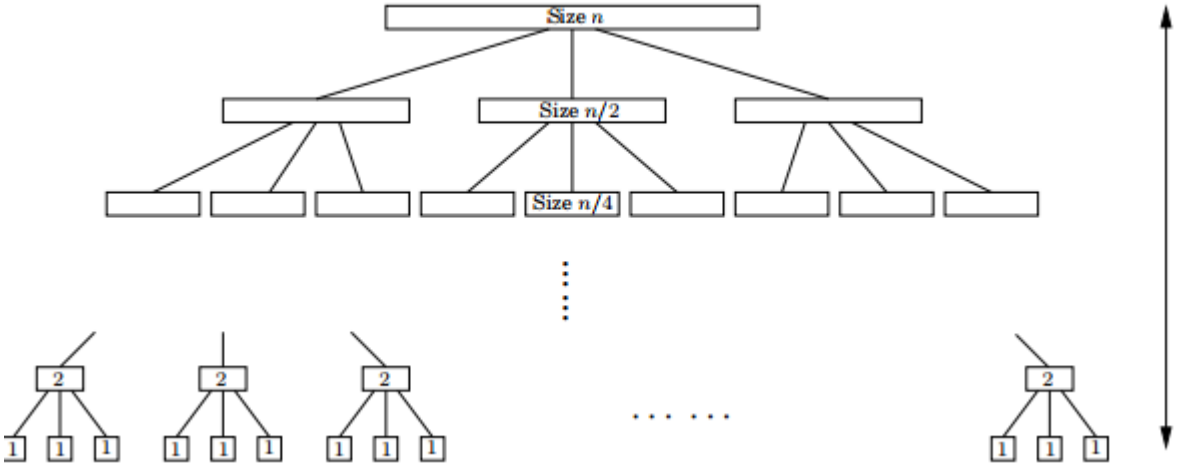
This algorithm was done by both of us, and it took 2 hours to implement it.

2. **The classical algorithm based on dynamic programming**

Time complexity: O(n*m).
Space complexity: O(n*m).

We used two functions to implement this algorithm, the first one initializes the matrix, and the second one fill it. In each iteration, we calculate the insertion, deletion and substitution. We compare the value of them and we choose the one with the lowest cost. The result is the value you reach in the last line, the last column.

3. **Dynamic programming and divide and conquer approaches**



In divide and conquer algorithm we try to solve sub problem of our main problem. Eash sub-problem is working for delete, replace and insert cost.

Time complexity : O(n^2)

4. **A branch-and-bound version of the recursive approach**

We have to have branch for each solution of our problem and we define a bound you to take this solution as a optimal solution or not. And we continue those steps until we reach a optimal solution.
Time complexity : O(n^2)

5. **An approximated version based on a greedy approach**

   We initialize a variable k at the difference between the length of two words word1 and word2. Then for each two characters in the same position of word1 and word2 we make a comparison, if character 1 is different from character 2 we increase k by 1.

   Time complexity: O(n), with n is the length of the biggest word

6. **An approximated randomized version**

   We choose a random number between the length of word1 and word2 plus the length of the smallest word.
   Time complexity: O(1).

7. **An approximated version of the classical dynamic programming approach where one fills only a stripe of size k around the diagonal of the matrix.**

   Two iterations used for the implementation of this algorithm. One from 1 until the length of the biggest word, and the other from the current diagonal until k.

   Time complexity = Space complexity = O(tall*k), with tall is the length of the biggest word. In the worst case the complexity will be equal to (n*m) as the classical dynamic.

8. **k-means algorithm**

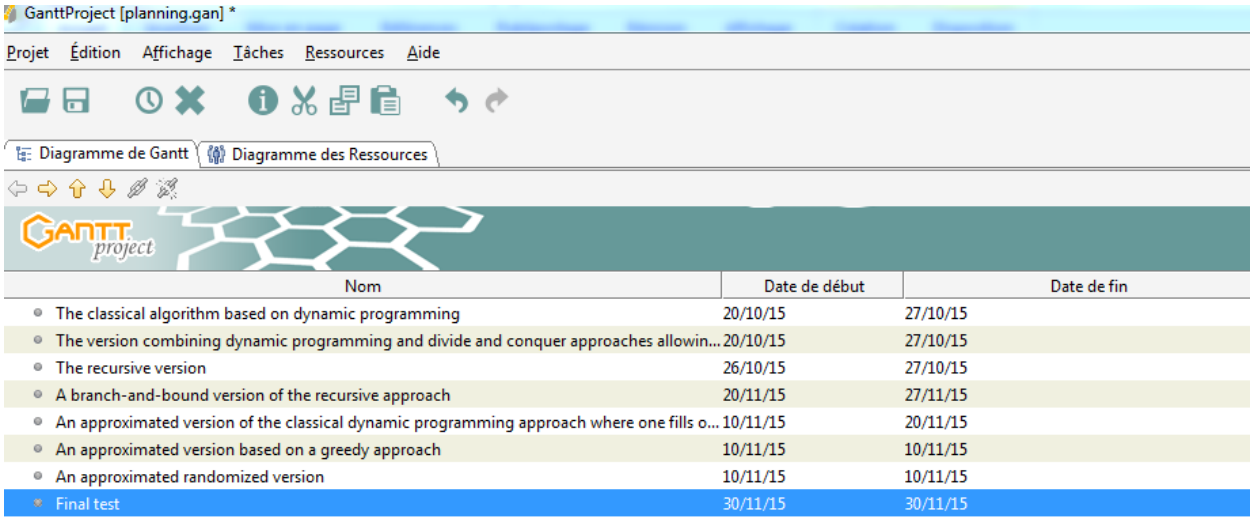
As part of this project, the database is a result of a string of characters, we ask the user to enter the number of Cluster wanted, in our case k. Then we choose randomly K chain of characters as the center. Afterwards, for each word in the database, we calculate the distance between the word and the center K uses the edit distance algorithm. We define an S threshold, if the distance is > S we consider the string as an outline.

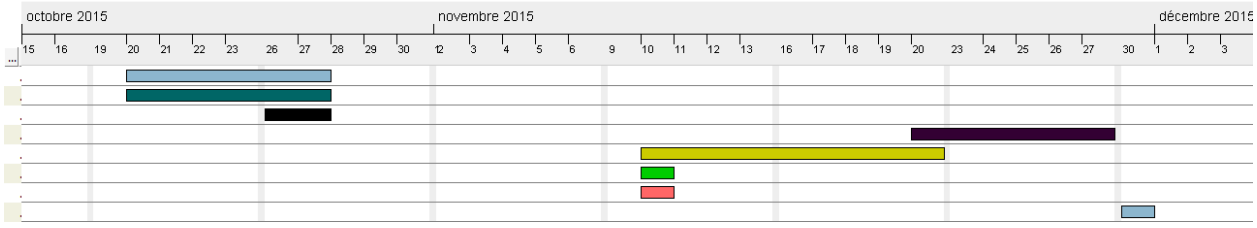
The new centroid is calculated from the new clusters, one does it again until another working two identical successive iteration.

The milestone of the project

Tasks to achieve	**Member to do it**	**Time deserved (unit test included , and comparing efficiency of algo after each task completed)**
The classical algorithm based on dynamic programming (see exercises and lecture slides).	AMYAR Amine	- 1 days - 2 hours
The version combining dynamic programming and divide and conquer approaches allowing one to solve the problem with a linear space complexity (see exercise sheet on dynamic programming).	AVCI Emin	- 2 days - 3 hours.
The recursive version (having an exponential time complexity).	AMYAR Amine AVCI Emin	- 1 days - 2 hours.
A branch-and-bound version of the recursive approach (see lecture slides).	AVCI Emin	- 2 days - 3 hours.
An approximated version of the classical dynamic programming approach where one fills only a stripe of size k around the diagonal of the matrix. (Different band values must be evaluated).	AMYAR Amine	- 1 day - 1 hour.
An approximated version based on a greedy approach (to be defined, cf lecture notes).	AMYAR Amine	- 1 days - 3 hour.
An approximated randomized version (to be defined, try to find an approach able to	AMYAR Amine	- 1 days - 1 hour.

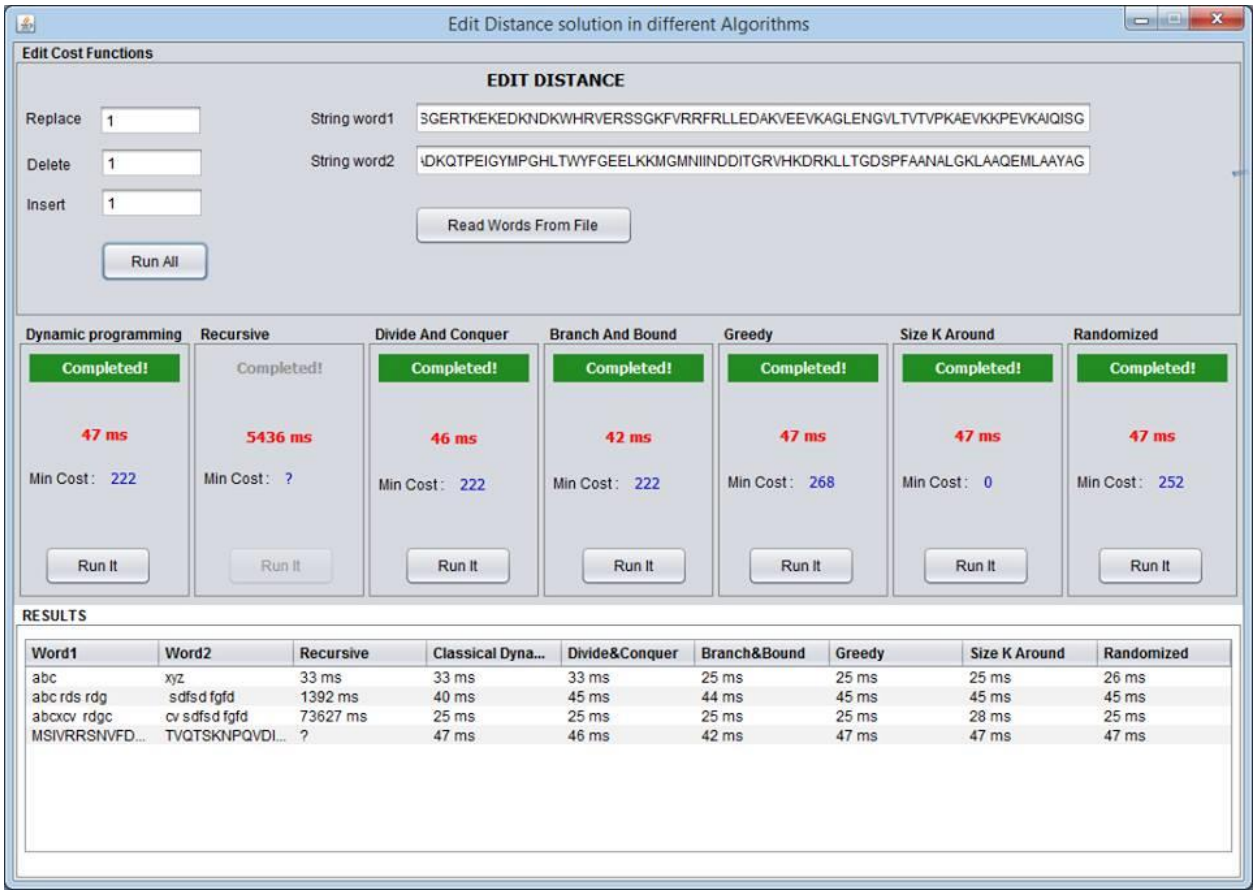
approximate the solution by using some random draws).		
Final test	AMYAR Amine AVCI Emin	- 2 days - 5 hours per day -





Elements used for testing the program:

- A string generator: we created a book generated randomly using script python.


Edit Distance solution in different Algorithms
Edit Cost Functions
EDIT DISTANCE
Replace
1
Delete
1
Insert
1
Run All
String word1
3GERTKEKEDKNDKWHRVERSSGKFVRRFRLLEDAKVEEVKAGLENGVLTVTVPKAEVKKPEVKAIQISG
String word2
\DKQTPEIGYMPGHLTWYFGEELKKMGMNIINDDITGRVHKDRKLLTGDSPFAANALGKLAAQEMLAAYAG
Read Words From File
Dynamic programming
Completed!
47 ms
Min Cost : 222
Run It
Recursive
Completed!
5436 ms
Min Cost : ?
Run It
Divide And Conquer
Completed!
46 ms
Min Cost : 222
Run It
Branch And Bound
Completed!
42 ms
Min Cost : 222
Run It
Greedy
Completed!
47 ms
Min Cost : 268
Run It
Size K Around
Completed!
47 ms
Min Cost : 0
Run It
Randomized
Completed!
47 ms
Min Cost : 252
Run It
RESULTS
Word1
Word2
Recursive
Classical Dyna...
Divide&Conquer
Branch&Bound
Greedy
Size K Around
Randomized
abc
xyz
33 ms
33 ms
33 ms
25 ms
25 ms
25 ms
26 ms
abc rds rdg
sdfsd fgfd
1392 ms
40 ms
45 ms
44 ms
45 ms
45 ms
45 ms
abcxcv rdgc
cv sdfsd fgfd
73627 ms
25 ms
25 ms
25 ms
25 ms
28 ms
25 ms
MSIVRRSNVFD...
TVQTSKNPQVDI...
?
47 ms
46 ms
42 ms
47 ms
47 ms
47 ms

Test Date and Result

Word1 length	Word2 length	Recursive	Dynamic	Divide And Conquer	Branch & Bound	Greedy	Size K Around	Randomized
41	45	31500	22	31	22	22	21	32
10	14	206	39	38	37	23	37	38
3	3	23	23	22	21	23	22	22
947	2177	335000	33	24	2472	24	23	23
14	17	16000	25	26	25	25	25	24
6	6	25	25	26	25	26	26	26
1730	2597	489500	25	144	7425	26	25	26
8	11	27	33	27	32	27	27	33
1301	2285	396000	31	130	4112	31	28	28
5	6	32	33	33	32	32	32	33
7	17	79	34	32	32	32	32	33
30	49	25000	32	33	32	32	32	32
137	310	150000	34	33	33	33	22	33
1686	2537	425000	77	74	6688	38	36	38
1618	2504	412772	80	151	6345	82	80	87
624	923	212120	87	86	413	87	27	87
982	1708	288900	87	86	1658	87	87	87