Advanced Programming

Edit distance

Ramaseshan Ramachandran

MINIMUM EDIT DISTANCE

In computational linguistics, edit distance is a way to quantify how similar/dissimilar two strings are. This (dis)similarity is found by counting the minimum number of operations required to transform one string into the other. The count is known as Levenshtein minimum edit distance.

APPLICATIONS

- Retrieving similar words
- Spelling correction
- Speech recognition
- OCR error corrections
- Computational Biology - alignment of DNA sequence
- Identification of duplicate invoices

MINIMUM EDIT DISTANCE - OPERATIONS

- Two strings $S1$ and $S2$ are given
- $S1$ is the input word with wrong spelling and $S2$ is the Correct word
- Operations allowed to transform $S1$ to $S2$ - insert, delete and substitute/replace

EXAMPLES

S1 S2	W M	H U	I Z	Z Z	Z Z	I I	N N	G G
Operations	r	r	r	n	n	n	n	n
Cost of operation	1	1	1	0	0	0	0	0

Cost of operation = 3

S1 S2	W Q	H U	I I	Z Z	Z Z	I E	N R	G S
Operations	r	r	n	n	n	r	r	r
Cost of operation	1	1	0	0	0	1	1	1

Cost of operation = 5

S1 S2	C K	A A	R R	N N	A A	I T	I A	C K	 A
Operations	r	r	n	n	n	r	r	r	i
Cost of operation	1	0	0	0	0	0	1	1	1

Cost of operation = 4

FIND SIMILARITY BETWEEN DNA SEQUENCE

A	A	G	T	C	T	T	A	T	A	C	A	G	G	C
A	T	G	A	C	T	A	T	A	G	G	G	C	A	

A	A	G	T	C	T	T	A	T	A	C	A	G	G	C	-
	X		X								X				
A	T	G	A	C	T	-	A	T	A	G	G	-	G	C	A
	S		S			D					S				I

X between the pair	Transformation is necessary
– in the first string	Insertion operation is required
– in the second string	Deletion operation is needed

How about these two strings?

AGGCTATCACCTGACCTCCAGGCCGATGCCC

TAGCTATCACGACCGCGGTCGATTTGCCCGAC

ANOTHER EXAMPLE

S1 S2	E E	X X	I C	T I	E T	M E	E M	N E	T N	 T
Operations										
Cost of operation										

Cost of operation =

IMPORTANCE OF ALIGNMENT

S1 S2	E E	X X	I C	T I	E T	* E	M M	E E	N N	T T
Operations										
Cost of operation										

Cost of operation =

IMPORTANCE OF ALIGNMENT

S1	E	X	I	*	T	E	M	E	N	T
S2	E	X	C	I	T	E	M	E	N	T
Operations										
Cost of operation										

Cost of operation =

S1	E	X	*	I	T	E	M	E	N	T
S2	E	X	C	I	T	E	M	E	N	T
Operations										
Cost of operation										

Cost of operation =

SEARCH SPACE

Searching for sequence of edit operation (path in a graph) from the S_1 to the S_2:

- Start State: $S_1 \rightarrow S_2$
- Operators: Insert, delete and substitute
- Final State: S_2
- Cost: Number of edit operations
- Search space of all edit sequences is huge
- Several distinct path to reach the final state
- Do we need to remember all of them?
- Can we just find the shortest path?

PROPERTIES OF EDIT DISTANCE

- If two strings are equal, then Levenshtein distance is zero or $d(x, x) = \forall x \in M$, where M is the metric space
- The simplest operation of the replacement.
- If two strings differ in k^{th} position, then $d_e = D(x_k, y_k)$
- Penalty for multiple replacements $= \sum_{k=1}^{r} D(x_k, y_k)$
- If the deletion/insertion of an entry $x_{\hat{k}}$ gives the required string, then the penalty is $D(x_{\hat{k}}, y_{\hat{k}})$
- Penalty for multiple deletions $= \sum_{k=1}^{r} D(x_{\hat{k}}, y_{\hat{k}})$
- The cost of the operation $= \rho_r \sum_{k=1}^{r} D(x_k, y_k) + \rho_d \sum_{k=1}^{d} D(x_{\hat{k}d}, y_{\hat{k}d}) + \rho_i \sum_{k=1}^{i} D(x_{\hat{k}i}, y_{\hat{k}i})$ where ρ_s, ρ_d, ρ_i are the penalties for replacement, deletion and substitution operations, respectively

DYNAMIC PROGRAMMING

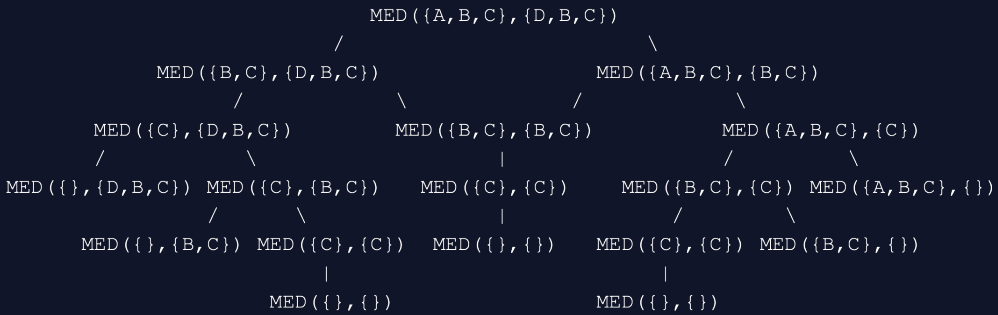
- **Breaking Down Complexity**
- Efficient mechanism to find optimal solutions which contains a lot of overlapping sub-problems
- **Optimal Substructure**

  Each sub-problem has an optimal solution that contributes to the global optimal solution
- **Memoization**

  Sub-problems results are stored to avoid redundant calculations
- **Bottom-Up Approach**

  Recombines solutions to sub-problems to form the final solution

SUBPROBLEMS



EDIT DISTANCE COMPUTATION

A tabular computation of $D(m,n)$ is achieved by computing minimum edit distance, $D(i,j)$ for small i,j i.e., fill up element $D(i,j)$ for $0 < i < m, 0 < j < n$ using two rules

$$\textbf{Rule 1:} \text{if } s_j = t_j \text{ then } D(i,j) = D(i-1,j-1) \tag{1}$$

$$\textbf{Rule 2:} \text{if } s_j \neq t_j \text{ then } D(i,j) = min \begin{cases} D(i-1,j)+1 & \text{insert} \\ D(i,j-1)+1 & \text{delete} \\ D(i-1,j-1)+1 & \text{replace} \end{cases} \tag{2}$$

DP - EDIT DISTANCE

	-	T	A	G	G	G	C	A
-	0	1	2	3	4	5	6	7
A	1							
C	2							
A	3							
G	4							
G	5							
C	6							

DP-SOLUTION

	-	T	A	G	G	G	C	A
-	0	1	2	3	4	5	6	7
A	1	1	1	2	3	4	5	6
C	2	2	2	2	3	4	4	5
A	3	3	2	3	3	4	5	4
G	4	4	3	2	3	3	4	5
G	5	5	4	3	2	3	4	5
C	6	6	5	4	3	4	3	4

DYNAMIC PROGRAMING - APPLICATION OF RULES

Application of Delete operation

		P	L	A	Y
	0	1	2	3	4
P	1	0	1	2	3
L	2	1	0	1	2
Y	3	2	1	1	1

Application of Insert operation

		P	L	Y
	0	1	2	3
P	1	0	1	2
L	2	1	0	1
A	3	2	1	1
Y	4	3	2	1

Application of replace operation

		P	L	O	Y
	0	1	2	3	4
P	1	0	1	2	3
L	2	1	0	1	2
A	3	2	1	1	2
Y	4	3	2	2	1

DYNAMIC PROGRAMMING -EXERCISE - CORRECT THIS TABLE

		E	X	I	T	E	M	E	N	T
	0	1	2	3	4	5	6	7	8	9
E	1	0	1	2	3	4	5	6	7	8
X	2	1	0	1	2	3	4	5	6	7
C	3	2	1	1	2	3	4	5	6	7
I	4	3	2	1	2	3	4	5	6	7
T	5	4	3	2	1	2	3	4	5	6
E	6	5	4	3	2	1	2	3	4	5
M	7	6	5	4	3	2	1	2	3	4
E	8	7	6	5	4	3	2	1	2	3
N	9	8	7	6	5	4	3	2	1	2
T	10	9	8	7	6	5	4	3	2	1

What is the minimum edit distance?

what is the minimum distance for the strings below:

How about these two strings?

AGGCTATCACCTGACCTCCAGGCCGATGCCC

TAGCTATCACGACCGCGGTCGATTTGCCCGAC

- Base condition - $D(i,0) = i$ and $D(0,j) = j$
- Termination $D(m,n)$

for $i = 1 \ldots m$

for $j = 1 \ldots n$

$$Di,j = min \begin{cases} D(i-1,j) + 1 \text{deletion} \\ D(i,j-1) + 1 \text{insertion} \\ D(i-1,j-1) + 1; \text{ if } X(i) \neq Y(j) \\ D(i-1,j-1) + 0; \text{ if } X(i) = Y(j) \end{cases}$$