CS 463
NATURAL LANGUAGE PROCESSING

Dr. Saleh Haridy

2023-2024

Week 3

Minimum Edit Distance and Alignment

How to compute similarity between two strings?

Spell correction

- The user typed “graffe”

  Which is closest to?
  - giraffe
  - graf
  - graft
  - grail

- Coreference: decide whether two strings refer to the same entity

MAjmaah President Dr.Saleh Almozal
Majmaah University Dr.Saleh Almozal

- these two strings are very similar (differing by only one word

- Also it is needed in Machine Translation, Information Extraction, Speech Recognition

Edit Distance

The minimum edit distance between two strings

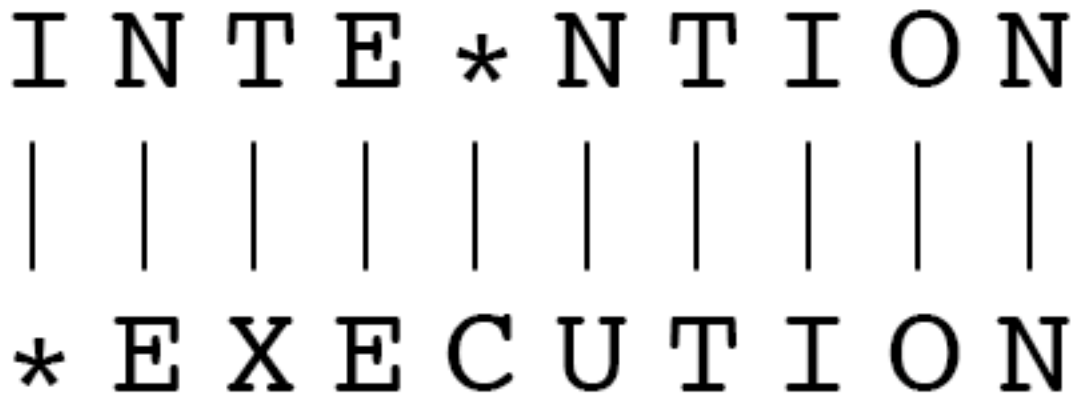
Is the minimum number of editing operations

- Insertion
- Deletion
- Substitution

Needed to transform one into the other

Minimum Edit Distance- example

Two strings and their **alignment**:



Minimum Edit Distance

- How to align two strings?
- Alignment is a correspondence between substrings of the two sequences

```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
d s s   i s
```

If each operation has cost of 1

- Distance between these is 5

If substitutions cost 2 (Levenshtein)

- Distance between them is 8

d for deletion, s for substitution, i for insertion.

Alignment in Computational Biology

Given a sequence of bases

AGGCTATCACCTGACCTCCAGGCCGATGCCC
TAGCTATCACGACCGCGGTCGATTTGCCCGAC

An alignment:

-AGG**CTATCAC**C**T****GACC**T**C**C**A****GG**C**CGA****--TGCCC**---
T**AG-CTATCAC****--****GACC**G**C****--GG**T**CGA**TT**TGCCC**GAC

Given two sequences, align each letter to a letter or gap

Other uses of Edit Distance in NLP

Evaluating Machine Translation and speech recognition

```
Spokesman confirms    senior government adviser was appointed
Spokesman said    the senior           adviser was appointed
            S      I              D
```

Named Entity Extraction and Entity Coreference

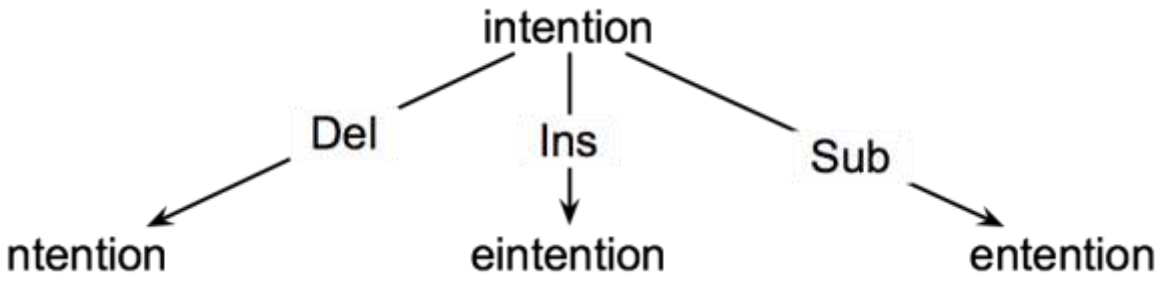
- IBM Inc. announced today
- IBM profits
- Stanford Professor Jennifer Eberhardt announced yesterday
- for Professor Eberhardt…

How to find the Min Edit Distance?

Searching for a path (sequence of edits) from the start string to the final string:

- **Initial state**: the word we're transforming
- **Operators**: insert, delete, substitute
- **Goal state**: the word we're trying to get to
- **Path cost**: what we want to minimize: the number of edits

Intention execution



Minimum Edit as Search

But the space of all edit sequences is huge!

- We can't afford to navigate naïvely
- Lots of distinct paths wind up at the same state.
  - We don't have to keep track of all of them
  - Just the shortest path to each of those revisited states.
- We can do this by using dynamic programming.
- Dynamic programming apply a table-driven method to solve problems by combining solutions to sub-problems.

Defining Min Edit Distance

For two strings

- X of length *n*
- Y of length *m*

We define D(*i,j*)

- the edit distance between X[1..*i*] and Y[1..*j*]
  - i.e., the first *i* characters of X and the first *j* characters of Y
- The edit distance between X and Y is thus D(*n,m*)

Dynamic Programming for Minimum Edit Distance

Dynamic programming: A tabular computation of D(*n,m*)

Solving problems by combining solutions to subproblems.

Bottom-up

- We compute D(i,j) for small *i,j*
- And compute larger D(i,j) based on previously computed smaller values
- i.e., compute D(*i,j*) for all *i* (0 < *i* < n) and *j* (0 < j < m)

$$D[i,j] = \min \begin{cases} D[i-1,j] + \text{del-cost}(source[i]) \\ D[i,j-1] + \text{ins-cost}(target[j]) \\ D[i-1,j-1] + \text{sub-cost}(source[i], target[j]) \end{cases}$$

Defining Min Edit Distance (Levenshtein)

Initialization

$$D(i,0) = i$$

$$D(0,j) = j$$

Recurrence Relation:

For each $i = 1 \ldots M$

For each $j = 1 \ldots N$

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } X(i) \neq Y(j) \\ 0; & \text{if } X(i) = Y(j) \end{cases} \end{cases}$$

Termination:

$D(N,M)$ is distance

Min Edit Distance

```
function MIN-EDIT-DISTANCE(source, target) returns min-distance

  n ← LENGTH(source)
  m ← LENGTH(target)
  Create a distance matrix distance[n+1,m+1]

  # Initialization: the zeroth row and column is the distance from the empty string
      D[0,0] = 0
      for each row i from 1 to n do
          D[i,0] ← D[i-1,0] + del-cost(source[i])
      for each column j from 1 to m do
          D[0,j] ← D[0,j-1] + ins-cost(target[j])

  # Recurrence relation:
  for each row i from 1 to n do
      for each column j from 1 to m do
          D[i,j] ← MIN( D[i−1,j] + del-cost(source[i]),
                        D[i−1,j−1] + sub-cost(source[i], target[j]),
                        D[i,j−1] + ins-cost(target[j]))
  # Termination
  return D[n,m]
```

The Edit Distance Table

$$D[i,j] = \min \begin{cases} D[i-1,j] + \text{del-cost}(source[i]) \\ D[i,j-1] + \text{ins-cost}(target[j]) \\ D[i-1,j-1] + \text{sub-cost}(source[i], target[j]) \end{cases}$$

target

Source

	#	E	X	E	C	U	T	I	O	N
#	**0**	1	2	3	4	5	6	7	8	9
I	1									
N	2									
T	3									
E	4									
N	5									
T	6									
I	7									
O	8									
N	9									

The Edit Distance Table

Src\Tar	#	e	x	e	c	u	t	i	o	n
#	0	1	2	3	4	5	6	7	8	9
i	1	2	3	4	5	6	7	6	7	8
n	2	3	4	5	6	7	8	7	8	7
t	3	4	5	6	7	8	7	8	9	8
e	4	3	4	5	6	7	8	9	10	9
n	5	4	5	6	7	8	9	10	11	10
t	6	5	6	7	8	9	8	9	10	11
i	7	6	7	8	9	10	9	8	9	10
o	8	7	8	9	10	11	10	9	8	9
n	9	8	9	10	11	12	11	10	9	8

Figure 2.18 Computation of minimum edit distance between *intention* and *execution* with the algorithm of Fig. 2.17, using Levenshtein distance with cost of 1 for insertions or deletions, 2 for substitutions.

Computing alignments

Edit distance isn't sufficient

- We often need to **align** each character of the two strings to each other

We do this by keeping a "backtrace"

Every time we enter a cell, remember where we came from

When we reach the end,

- Trace back the path from the upper right corner to read off the alignment

MinEdit with Backtrace

	#	e	x	e	c	u	t	i	o	n
#	0	← 1	← 2	← 3	← 4	← 5	← 6	← 7	← 8	← 9
i	↑ **1**	↖←↑ 2	↖←↑ 3	↖←↑ 4	↖←↑ 5	↖←↑ 6	↖←↑ 7	↖ 6	← 7	← 8
n	↑ 2	↖←↑ **3**	↖←↑ 4	↖←↑ 5	↖←↑ 6	↖←↑ 7	↖←↑ 8	↑ 7	↖←↑ 8	↖ 7
t	↑ 3	↖←↑ 4	↖←↑ **5**	↖←↑ 6	↖←↑ 7	↖←↑ 8	↖ 7	←↑ 8	↖←↑ 9	↑ 8
e	↑ 4	↖ 3	← 4	↖← **5**	← **6**	← 7	←↑ 8	↖←↑ 9	↖←↑ 10	↑ 9
n	↑ 5	↑ 4	↖←↑ 5	↖←↑ 6	↖←↑ 7	↖←↑ **8**	↖←↑ 9	↖←↑ 10	↖←↑ 11	↖↑ 10
t	↑ 6	↑ 5	↖←↑ 6	↖←↑ 7	↖←↑ 8	↖←↑ 9	↖ **8**	← 9	← 10	←↑ 11
i	↑ 7	↑ 6	↖←↑ 7	↖←↑ 8	↖←↑ 9	↖←↑ 10	↑ 9	↖ **8**	← 9	← 10
o	↑ 8	↑ 7	↖←↑ 8	↖←↑ 9	↖←↑ 10	↖←↑ 11	↑ 10	↑ 9	↖ **8**	← 9
n	↑ 9	↑ 8	↖←↑ 9	↖←↑ 10	↖←↑ 11	↖←↑ 12	↑ 11	↑ 10	↑ 9	↖ **8**

$$ptr(i,j) = \begin{cases} \texttt{LEFT} & \text{insertion} \\ \texttt{TOP} & \text{deletion} \\ \texttt{DIAG} & \text{substitution} \end{cases}$$

Adding Backtrace to Minimum Edit Distance

Base conditions:

`D(i,0) = i` `D(0,j) = j`

Termination:

`D(N,M) is distance`

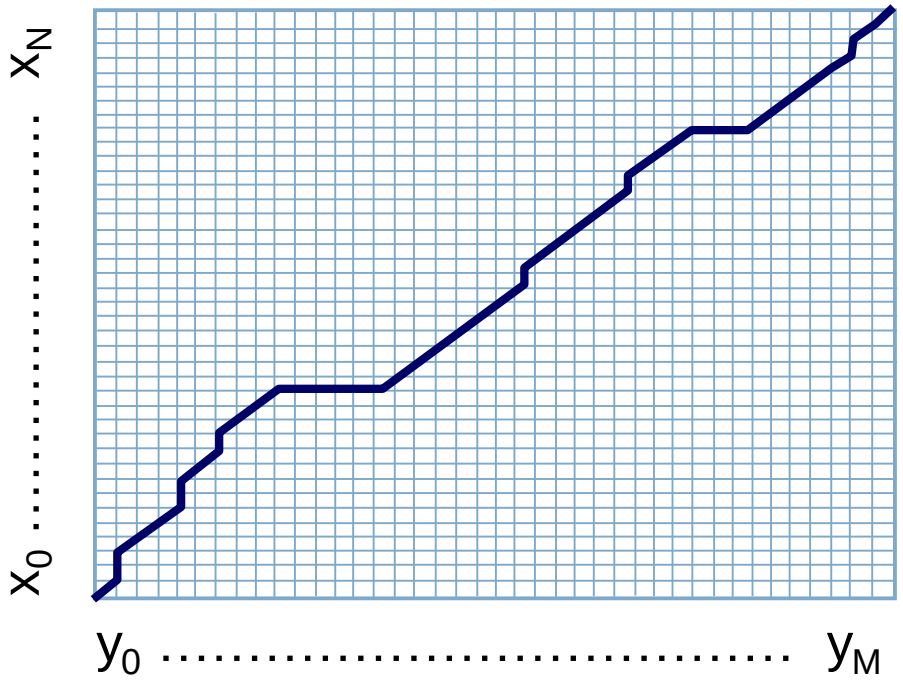
Recurrence Relation:

```
For each  i = 1…M
      For each  j = 1…N
```

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 & \text{deletion} \\ D(i,j-1) + 1 & \text{insertion} \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } X(i) \neq Y(j) \quad \text{substitution} \\ 0; & \text{if } X(i) = Y(j) \end{cases} \end{cases}$$

$$ptr(i,j) = \begin{cases} LEFT & \text{insertion} \\ TOP & \text{deletion} \\ DIAG & \text{substitution} \end{cases}$$

The Distance Matrix



Every non-decreasing path

from (0,0) to (M, N)

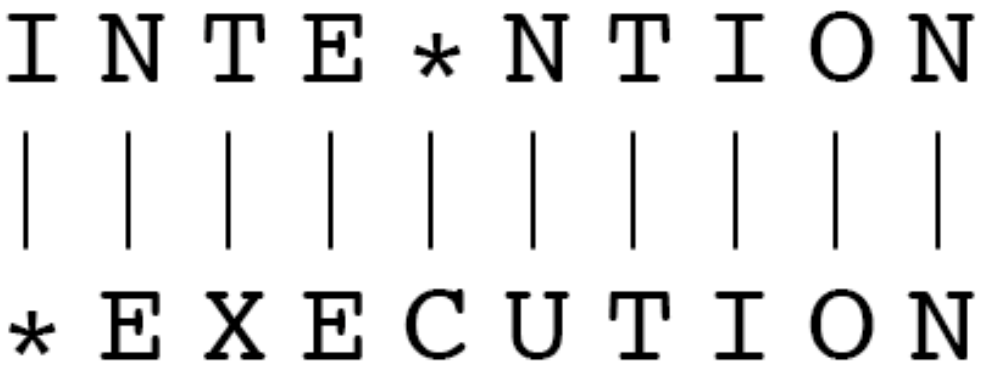
corresponds to
an alignment
of the two sequences

An optimal alignment is composed of optimal subalignments

SLIDE ADAPTED FROM SERAFIM BATZOGLOU WITH PERMISSION

Result of Backtrace

Two strings and their **alignment**:



Example 2

	#	H	E	L	L	O
#		⇐ 1	⇐ 2	⇐ 3	⇐ 4	⇐ 5
S	⇑ **1**	⇑↖⇐ 2	⇑↖⇐ 3	⇑↖⇐ 4	⇑↖⇐ 5	⇑↖⇐ 6
P	⇑ 2	⇑↖⇐ **3**	⇑↖⇐ 4	⇑↖⇐ 5	⇑↖⇐ 6	⇑↖⇐ 7
E	⇑ 3	⇑↖⇐ 4	↖ **3**	⇐ 4	⇐ 5	⇐ 6
L	⇑ 4	⇑↖⇐ 5	⇑ 4	↖ **3**	↖⇐ 4	⇐ 5
L	⇑ 5	⇑↖⇐ 6	⇑ 5	⇑↖ 4	↖ **3**	⇐ **4**

Alignment:

s	p	e	l	l	
	h	e	l	l	o
d	s				i

Performance

Time:

O(nm)

Space:

O(nm)

Backtrace

O(n+m)

Minimum Distance implementation in python

https://giov.dev/2016/01/minimum-edit-distance-in-python.html