Minimum Edit Distance

Application

- Spell correction
  - The user typed “graffe”

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

- Computational Biology
  - Align two sequences of nucleotides

    AGGCTATCACCTGACCTCCAGGCCGATGCCC
    TAGCTATCACGACCGCGGTCGATTTGCCCGAC

  - Resulting alignment:

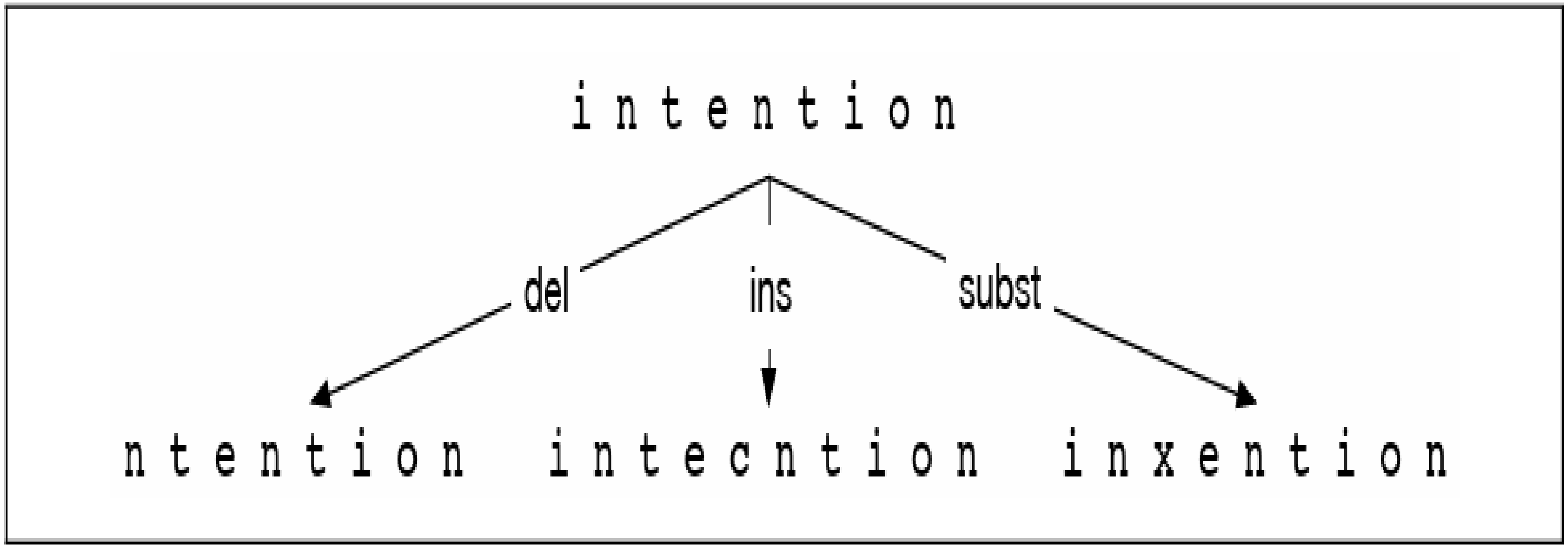
    -AGGCTATCACCTGACCTCCAGGCCGA--TGCCC---
    TAG-CTATCAC--GACCGC--GGTCGATTTGCCCGAC

- Also for 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

Meaning of Edit Operation



Minimum Edit Distance

- Two strings and their **alignment**:

```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
```

```
i n t e n t i o n
                    ← delete i
n t e n t i o n
                    ← substitute n by e
e t e n t i o n
                    ← substitute t by x
e x e n t i o n
                    ← insert u
e x e n u t i o n
                    ← substitute n by c
e x e c u t i o n
```

Example

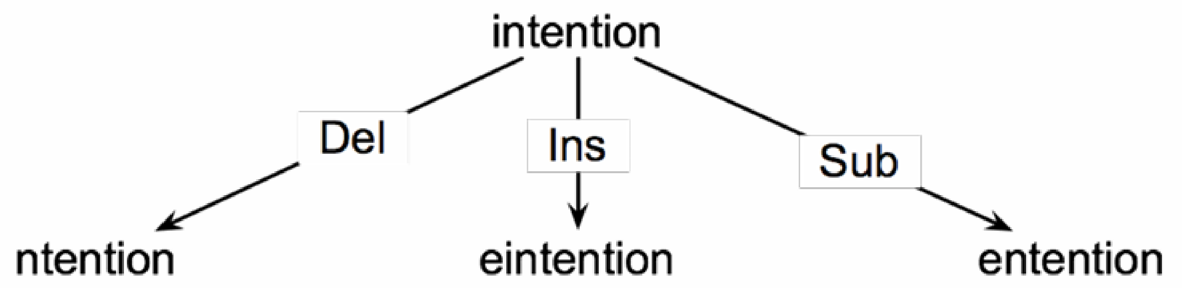
Minimum Edit Distance

```
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

How to find Minimum 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



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 revisted states.

Defining Minimum 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 Distan

- **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)

Defining Minimum Edit Distance (LAVENSHTEIN)

- Initialization

  D(i,0) = i

  D(0,j) = j

- Recurrence Relation:

  For each i = 1…M

  For each j = 1…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

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

$$D[i,j] = \min \begin{cases} D[i-1,j] + 1 \\ D[i,j-1] + 1 \\ D[i-1,j-1] + \begin{cases} 2; & \text{if } source[i] \neq target[j] \\ 0; & \text{if } source[i] = target[j] \end{cases} \end{cases}$$

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

  n ← LENGTH(source)
  m ← LENGTH(target)
  Create a distance matrix D[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]
```

Source string = n Target = m j

i

Src/Tar	#	E	X	E	C	U	T	I	O	N
#										
I										
N										
T										
E										
N										
T										
I										
O										
N										

11×11

← INSERT ↑ DELETE

Src/Tar	#	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									

11×11

INSERT DELETE

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									
T	3									
E	4									
N	5									
T	6									
I	7									
O	8									
N	9									

11×11

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.

	#	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 2.19 When entering a value in each cell, we mark which of the three neighboring cells we came from with up to three arrows. After the table is full we compute an **alignment** (minimum edit path) by using a **backtrace**, starting at the **8** in the lower-right corner and following the arrows back. The sequence of bold cells represents one possible minimum cost alignment between the two strings. Diagram design after Gusfield (1997).

THANK YOU