

Similarity Search

The String Edit Distance

Nikolaus Augsten
nikolaus.augsten@sbg.ac.at
Department of Computer Sciences
University of Salzburg



WS 2021/22
Version October 26, 2021

Outline

- 1 String Edit Distance
 - Motivation and Definition
 - Brute Force Algorithm
 - Dynamic Programming Algorithm
 - Edit Distance Variants
- 2 Conclusion

Outline

- 1 String Edit Distance
 - Motivation and Definition
 - Brute Force Algorithm
 - Dynamic Programming Algorithm
 - Edit Distance Variants
- 2 Conclusion

Motivation

- How different are
 - `hello` and `hello?`
 - `hello` and `hallo?`
 - `hello` and `hell?`
 - `hello` and `shell?`

What is a String Distance Function?

Definition (String Distance Function)

Given a finite alphabet Σ , a *string distance function*, δ_s , maps each pair of strings $(x, y) \in \Sigma^* \times \Sigma^*$ to a positive real number (including zero).

$$\delta_s : \Sigma^* \times \Sigma^* \rightarrow \mathbb{R}_0^+$$

- Σ^* is the set of all strings over Σ , including the *empty string* ϵ .

Outline

- 1 String Edit Distance
 - Motivation and Definition
 - Brute Force Algorithm
 - Dynamic Programming Algorithm
 - Edit Distance Variants
- 2 Conclusion

The String Edit Distance

Definition (String Edit Distance)

The *string edit distance* between two strings, $\text{ed}(x, y)$, is the minimum number of character insertions, deletions and replacements that transforms x to y .

- Example:
 - `hello` → `hallo`: replace `e` by `a`
 - `hello` → `hell`: delete `o`
 - `hello` → `shell`: delete `o`, insert `s`
- Also called *Levenshtein distance*.¹

¹Levenshtein introduced this distance for signal processing in 1965 [Lev65].

Gap Representation

- *Gap representation* of the string transformation $x \rightarrow y$:
Place string x above string y
 - with a gap in x for every insertion,
 - with a gap in y for every deletion,
 - with different characters in x and y for every replacement.
- Any sequence of edit operations can be represented with gaps.
- Example:


```

h a l l o
s h e l l

```

 - insert `s`
 - replace `a` by `e`
 - delete `o`

Deriving the Recursive Formula

• Example:

```

  h a l l o
  s h e l l

```

- **Given:** Gap representation, $gap(x, y)$, of the shortest edit distance between two strings x and y , such that $gap(x, y) = ed(x, y)$.

• **Claim:**

- If we remove the last column,
- then the remaining columns represent the shortest edit distance, $gap(x', y') = ed(x', y')$, between the remaining substrings, x' and y' .

• **Proof** (by contradiction):

- Last column contributes with $c = 0$ or $c = 1$ to $gap(x, y)$, thus $gap(x, y) = gap(x', y') + c$.
- If we assume $ed(x', y') < gap(x', y')$, then we could find a new gap representation $gap^*(x', y') = ed(x', y') < gap(x', y')$ such that $gap^*(x, y) = gap^*(x', y') + c < gap(x', y') + c = ed(x, y)$. \square

Deriving the Recursive Formula

• Example:

```

  h a l l o
  s h e l l

```

• **Notation:**

- $x[1 \dots i]$ is the substring of the first i characters of x ($x[1 \dots 0] = \varepsilon$)
- $x[i]$ is the i -th character of x

• **Recursive Formula:**

$$\begin{aligned}
 ed(\varepsilon, \varepsilon) &= 0 \\
 ed(x[1..i], \varepsilon) &= i \\
 ed(\varepsilon, y[1..j]) &= j \\
 ed(x[1..i], y[1..j]) &= \min(ed(x[1..i-1], y[1..j-1]) + c, \\
 &\quad ed(x[1..i-1], y[1..j]) + 1, \\
 &\quad ed(x[1..i], y[1..j-1]) + 1)
 \end{aligned}$$

where $c = 0$ if $x[i] = y[j]$, otherwise $c = 1$.

Brute Force Algorithm

```
ed-bf(x, y)
```

```
m = |x|, n = |y|
```

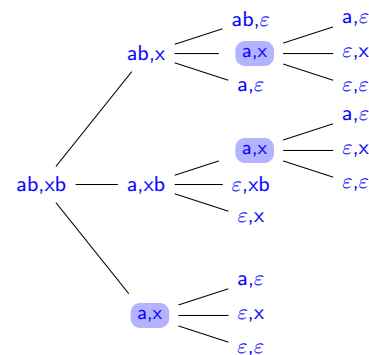
```
if m = 0 then return n
```

```
if n = 0 then return m
```

```
if x[m] = y[n] then c = 0 else c = 1
```

```
return min(ed-bf(x, y[1..n-1]) + 1,
           ed-bf(x[1..m-1], y) + 1,
           ed-bf(x[1..m-1], y[1..n-1]) + c)
```

Brute Force Algorithm

• Recursion tree for $ed\text{-}bf(ab, xb)$:

- Exponential runtime in string length :-)
- **Observation:** Subproblems are computed repeatedly (e.g. $ed\text{-}bf(a, x)$ is computed 3 times)
- **Approach:** Reuse previously computed results!

Outline

- String Edit Distance
 - Motivation and Definition
 - Brute Force Algorithm
 - Dynamic Programming Algorithm
 - Edit Distance Variants

- Conclusion

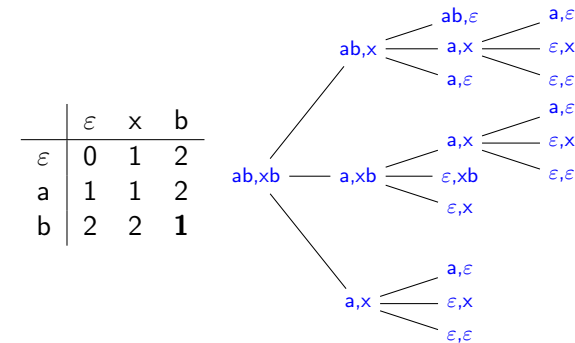
Dynamic Programming Algorithm

- Store distances between all prefixes of x and y
- Use matrix $C_{0..m,0..n}$ with

$$C_{i,j} = \text{ed}(x[1 \dots i], y[1 \dots j])$$

where $x[1..0] = y[1..0] = \varepsilon$.

- Example:



Dynamic Programming Algorithm

 $\text{ed-dyn}(x, y)$

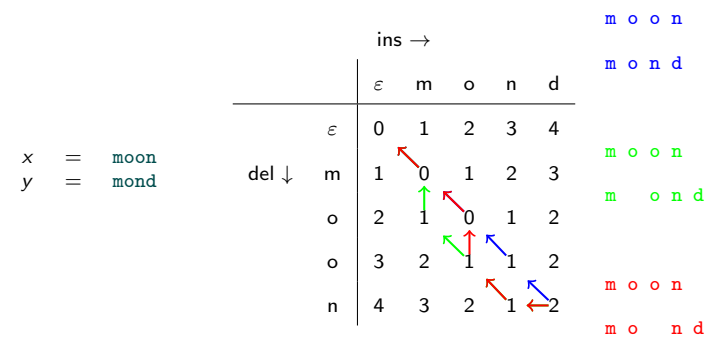
```

C : array[0..|x|][0..|y|]
for i = 0 to |x| do C[i, 0] = i
for j = 1 to |y| do C[0, j] = j
for j = 1 to |y| do
  for i = 1 to |x| do
    if x[i] = y[j] then c = 0 else c = 1
    C[i, j] = min(C[i - 1, j - 1] + c,
                  C[i - 1, j] + 1,
                  C[i, j - 1] + 1)

```

Understanding the Solution

- Example:



- Solution 1:** replace **n** by **d** and (second) **o** by **n** in x
- Solution 2:** insert **d** after **n** and delete (first) **o** in x
- Solution 3:** insert **d** after **n** and delete (second) **o** in x

Dynamic Programming Algorithm – Properties

- Complexity:
 - $O(mn)$ time (nested for-loop)
 - $O(mn)$ space (the $(m+1) \times (n+1)$ -matrix C)
- Improving space complexity (assume $m < n$):
 - we need only the previous column to compute the next column
 - we can forget all other columns
 - $\Rightarrow O(m)$ space complexity

Dynamic Programming Algorithm

$\text{ed-dyn}^+(x, y)$

```

col0 : array[0..|x|]
col1 : array[0..|x|]
for i = 0 to |x| do col0[i] = i
for j = 1 to |y| do
  col1[0] = j
  for i = 1 to |x| do
    if x[i] = y[j] then c = 0 else c = 1
    col1[i] = min(col0[i - 1] + c,
                  col1[i - 1] + 1,
                  col0[i] + 1)

col0 = col1

```

Outline

- 1 String Edit Distance
 - Motivation and Definition
 - Brute Force Algorithm
 - Dynamic Programming Algorithm
 - Edit Distance Variants
- 2 Conclusion

Distance Metric

Definition (Distance Metric)

A distance function δ is a *distance metric* if and only if for any x, y, z the following hold:

- $\delta(x, y) = 0 \Leftrightarrow x = y$ (identity)
- $\delta(x, y) = \delta(y, x)$ (symmetric)
- $\delta(x, y) + \delta(y, z) \geq \delta(x, z)$ (triangle inequality)

Examples:

- the Euclidean distance is a metric
- $d(a, b) = a - b$ is not a metric (not symmetric)

Introducing Weights

- Look at the **edit operations** as a set of **rules with a cost**:

$$\begin{aligned}\alpha(\varepsilon, b) &= \omega_{ins} && \text{(insert)} \\ \alpha(a, \varepsilon) &= \omega_{del} && \text{(delete)} \\ \alpha(a, b) &= \begin{cases} \omega_{rep} & \text{if } a \neq b \\ 0 & \text{if } a = b \end{cases} && \text{(replace)}\end{aligned}$$

where $a, b \in \Sigma$, and $\omega_{ins}, \omega_{del}, \omega_{rep} \in \mathbb{R}_0^+$.

- Edit script**: sequence of rules that transform x to y
- Edit distance**: edit script with minimum cost (adding up costs of single rules)
- Example**: so far we assumed $\omega_{ins} = \omega_{del} = \omega_{rep} = 1$.

Weighted Edit Distance

- Recursive formula with weights**:

$$\begin{aligned}C_{0,0} &= 0 \\ C_{i,j} &= \min(C_{i-1,j-1} + \alpha(x[i], y[j]), \\ &\quad C_{i-1,j} + \alpha(x[i], \varepsilon), \\ &\quad C_{i,j-1} + \alpha(\varepsilon, y[j]))\end{aligned}$$

where $\alpha(a, a) = 0$ for all $a \in \Sigma$, and $C_{-1,j} = C_{i,-1} = \infty$.

- We can easily **adapt** the dynamic programming **algorithm**.

Variants of the Edit Distance

- Unit cost edit distance** (what we did so far):
 - $\omega_{ins} = \omega_{del} = \omega_{rep} = 1$
 - $0 \leq ed(x, y) \leq \max(|x|, |y|)$
 - distance metric
- Hamming distance** [Ham50, SK83]:
 - called also "string matching with k mismatches"
 - allows only replacements
 - $\omega_{rep} = 1, \omega_{ins} = \omega_{del} = \infty$
 - $0 \leq d(x, y) \leq |x|$ if $|x| = |y|$, otherwise $d(x, y) = \infty$
 - distance metric
- Longest Common Subsequence (LCS) distance** [NW70, AG87]:
 - allows only insertions and deletions
 - $\omega_{ins} = \omega_{del} = 1, \omega_{rep} = \infty$
 - $0 \leq d(x, y) \leq |x| + |y|$
 - distance metric
 - $LCS(x, y) = (|x| + |y| - d(x, y))/2$

Allowing Transposition

- Transpositions**
 - switch two adjacent characters
 - can be simulated by delete and insert
 - typos are often transpositions
- New rule for transposition**

$$\alpha(ab, ba) = \omega_{trans}$$

allows us to assign a weight different from $\omega_{ins} + \omega_{del}$

- Recursive formula** that includes transposition:

$$\begin{aligned}C_{0,0} &= 0 \\ C_{i,j} &= \min(C_{i-1,j-1} + \alpha(x[i], y[j]), \\ &\quad C_{i-1,j} + \alpha(x[i], \varepsilon), \\ &\quad C_{i,j-1} + \alpha(\varepsilon, y[j]), \\ &\quad C_{i-2,j-2} + \alpha(x[i-1]x[i], y[j-1]y[j]))\end{aligned}$$

where $\alpha(ab, cd) = \infty$ if $a \neq d$ or $b \neq c$, $\alpha(a, a) = 0$ for all $a \in \Sigma$, and $C_{-1,j} = C_{i,-1} = C_{-2,j} = C_{i,-2} = \infty$.

Example: Edit Distance with Transposition

- **Example:** Compute distance between $x = \text{meal}$ and $y = \text{mael}$ using the edit distance with transposition ($\omega_{ins} = \omega_{del} = \omega_{rep} = \omega_{trans} = 1$)

	ϵ	m	a	e	l
ϵ	0	1	2	3	4
m	1	0	1	2	3
e	2	1	1	1	2
a	3	2	1	1	2
l	4	3	2	2	1

- The value in red results from the transposition of ea to ae.

Text Searching

- **Goal:**
 - search pattern p in text t ($|p| < |t|$)
 - allow k errors
 - match may start at any position of the text
- **Difference** to distance computation:
 - $C_{0,j} = 0$ (instead of $C_{0,j} = j$, as text may start at any position)
 - result: all $C_{m,j} \leq k$ are endpoints of matches

Example: Text Searching

- **Example:**

	ϵ	s	u	r	g	e	r	y
ϵ	0	0	0	0	0	0	0	0
s	1	0	1	1	1	1	1	1
u	2	1	0	1	2	2	2	2
r	3	2	1	0	1	2	2	3
v	4	3	2	1	1	2	3	3
e	5	4	3	2	2	1	2	3
y	6	5	4	3	3	2	2	2

- **Solutions:** 3 matching positions with $k \leq 2$ found.

```
s u r v e y
s u r g e
```

```
s u r v e y
s u r g e r
```

```
s u r v e y
s u r g e r y
```

Summary

- **Edit distance** between two strings: the minimum number of edit operations that transforms one string into the another
- **Dynamic programming algorithm** with $O(mn)$ time and $O(m)$ space complexity, where $m \leq n$ are the string lengths.
- Basic algorithm can easily be **extended** in order to:
 - weight edit operations differently,
 - support transposition,
 - simulate Hamming distance and LCS,
 - search pattern in text with k errors.



Alberto Apostolico and Zvi Galil.

The longest common subsequence problem revisited.
Algorithmica, 2(1):315–336, March 1987.



Richard W. Hamming.

Error detecting and error correcting codes.
Bell System Technical Journal, 26(2):147–160, 1950.



Vladimir I. Levenshtein.

Binary codes capable of correcting spurious insertions and deletions of ones.
Problems of Information Transmission, 1:8–17, 1965.



Saul B. Needleman and Christian D. Wunsch.

A general method applicable to the search for similarities in the amino acid sequence of two proteins.
Journal of Molecular Biology, 48:443–453, 1970.



David Sankoff and Josef B. Kruskal, editors.

Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison.
Addison-Wesley, Reading, MA, 1983.