12 Edit distance
Problem: similarity of strings

Edit distance

For two strings $A$ and $B$, compute, as efficiently as possible, the edit distance $D(A,B)$ and a minimal sequence of edit operations which transforms $A$ into $B$.

inf - - - o r m a t i k -
in t e r p o l - a t i o n
Problem: similarity of strings

Approximate string matching

For a given text $T$, a pattern $P$, and a distance $d$, find all substrings $P'$ in $T$ with $D(P,P') \leq d$

Sequence alignment

Find optimal alignments of DNA sequences

```
G A G C A
- - - - - -
C T T G G A T T C T C G G
- - - C A C G T G G - - - - - - - -
```
Edit distance

Given: two strings $A = a_1a_2.... a_m$ and $B = b_1b_2... b_n$

Goal: find minimal cost $D(A,B)$ for a sequence of edit operations to transform $A$ into $B$.

Edit operations:

1. Replace a character in $A$ by a character from $B$
2. Delete a character from $A$
3. Insert a character from $B$
Edit distance

Cost model:

\[ c(a, b) = \begin{cases} 
1 & \text{if } a \neq b \\
0 & \text{if } a = b 
\end{cases} \]

\( a = \epsilon, \ b = \epsilon \) possible

We assume the triangle inequality holds for \( c \):

\[ c(a,c) \leq c(a,b) + c(b,c) \]

\( \rightarrow \) Each character is changed at most once
Edit distances

Trace as representation of edit sequences

\[ A = \begin{array}{ccccccccc}
  b & a & a & c & a & a & b & c \\
\end{array} \]

\[ B = \begin{array}{ccccccccc}
  a & b & a & c & b & c & a & c \\
\end{array} \]

or using indents

\[ A = \begin{array}{ccccccc}
  - & b & a & a & c & a & - & a & b & c \\
\end{array} \]

\[ B = \begin{array}{ccccccc}
  a & b & a & - & c & b & c & a & - & c \\
\end{array} \]

Edit distance (cost): 5

An optimal trace can be divided into two optimal subtraces

\[ \rightarrow \text{dynamic programming can be used} \]
Dynamic programming

- Algorithm design technique often used for optimization problems

- Generally usable for recursive approaches if the same partial solutions are required more than once

- Approach: store partial results in a table

- Advantage: better time complexity, often polynomial instead of exponential
Computation of the edit distance

Let $A_i = a_1...a_i$ and $B_j = b_1....b_j$

$$D_{i,j} = D(A_i, B_j)$$
Computations of the edit distances

- Three possibilities of ending a trace:
  - 1. $a_m$ is replaced by $b_n$:
    \[ D_{m,n} = D_{m-1,n-1} + c(a_m, b_n) \]
  - 2. $a_m$ is deleted:
    \[ D_{m,n} = D_{m-1,n} + 1 \]
  - 3. $b_n$ is inserted:
    \[ D_{m,n} = D_{m,n-1} + 1 \]
Computations of the edit distances

- Three possibilities of ending a trace:

  1. $a_m$ is replaced by $b_n$:
     \[ D_{m,n} = D_{m-1,n-1} + c(a_m, b_n) \]

  2. $a_m$ is deleted:
     \[ D_{m,n} = D_{m-1,n} + 1 \]

  3. $b_n$ is inserted:
     \[ D_{m,n} = D_{m,n-1} + 1 \]
Computations of the edit distances

- Three possibilities of ending a trace:
  - 1. $a_m$ is replaced by $b_n$:
    \[ D_{m,n} = D_{m-1,n-1} + c(a_m, b_n) \]
  - 2. $a_m$ is deleted:
    \[ D_{m,n} = D_{m-1,n} + 1 \]
  - 3. $b_n$ is inserted:
    \[ D_{m,n} = D_{m,n-1} + 1 \]
Computation of the edit distance

- Recurrence relation, if $m,n \geq 1$:

$$D_{m,n} = \min\left\{\begin{array}{c}
D_{m-1,n-1} + c(a_m, b_n), \\
D_{m-1,n} + 1, \\
D_{m,n-1} + 1
\end{array}\right\}$$

- Computation of all $D_{i,j}$ is required, $0 \leq i \leq m$, $0 \leq j \leq n$. 

![Diagram showing the computation of edit distance](diagram.png)
Recurrence relation for the edit distance

Base cases:

\[ D_{0,0} = D(\varepsilon, \varepsilon) = 0 \]
\[ D_{0,j} = D(\varepsilon, B_j) = j \]
\[ D_{i,0} = D(A_i, \varepsilon) = i \]

Recurrence equation:

\[ D_{i,j} = \min \left\{ D_{i-1,j-1} + c(a_i, b_j), D_{i-1,j} + 1, D_{i,j-1} + 1 \right\} \]
Order of computation for the edit distance
Algorithm for the edit distance

Algorithm edit_distance

Input: two strings $A = a_1 \ldots a_m$ and $B = b_1 \ldots b_n$

Output: the matrix $D = (D_{ij})$

1. $D[0,0] := 0$

2. for $i := 1$ to $m$ do $D[i,0] = i$

3. for $j := 1$ to $n$ do $D[0,j] = j$

4. for $i := 1$ to $m$ do

5. for $j := 1$ to $n$ do

6. $D[i,j] := \min(D[i-1,j] + 1,$

7. $D[i,j - 1] + 1,$

8. $D[i-1,j-1] + c(a_i, b_j))$
**Example**

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>b</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>c</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Example

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>b</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>c</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The edit distance matrix for the strings "abaa" and "abc" is shown above. The cells indicate the number of insertions (ins) or deletions (del) needed to transform one string into the other. For example, to transform "abaa" into "abc", we need 4 insertions: first insert "c", then insert "b", then insert "a", and finally insert "c".
### Example

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1_{ins}</td>
<td>2_{ins}</td>
<td>3_{ins}</td>
<td>4_{ins}</td>
</tr>
<tr>
<td>1</td>
<td>b</td>
<td>1_{del}</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>2_{del}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td>3_{del}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>c</td>
<td>4_{del}</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

<table>
<thead>
<tr>
<th>( j )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>( i )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>b</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>c</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
D(\epsilon, 2) + 1 = 3 \\
D(1, 1) + 1 = 2 \\
D(\epsilon, 1) + 0 = 1
\]
### Example

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>b</td>
<td>1_del</td>
<td>1_ins</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>2_del</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td>3_del</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>c</td>
<td>4_del</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- $D(\varepsilon, 2) + 1 = 3$
- $D(1, 1) + 1 = 2$
- $D(\varepsilon, 1) + 0 = 1$
### Example

**Edit distance**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>i</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>b</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>c</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **a**
- **b**
- **a**
- **c**
Example

<table>
<thead>
<tr>
<th>i</th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The table shows the edit distance between two strings. The entries are:

- **ins**: Insertion
- **del**: Deletion
**Example**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>b</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td>3</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>c</td>
<td>4</td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Edit distance**

- **Insertion (ins)**: When a character from the second string is added to the end of the first string.
- **Deletion (del)**: When a character from the first string is removed.

The table above illustrates the edit distance between two strings using dynamic programming.
### Example

Here is an example of how the edit distance is calculated for the strings 'abc' and 'acb'.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1(_{ins})</td>
<td>2(_{ins})</td>
<td>3(_{ins})</td>
<td>4(_{ins})</td>
</tr>
<tr>
<td>1</td>
<td>b</td>
<td>1(_{del})</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>2(_{del})</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td>3(_{del})</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>c</td>
<td>4(_{del})</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
### Example

The table below illustrates the edit distance between two strings, `abac` and `abc`, using the Levenshtein distance algorithm. The table is structured as follows:

- **i** (rows): Represents the characters `a`, `b`, `a`, and `c` from the first string, `abac`.
- **j** (columns): Represents the characters `a`, `b`, `a`, `c` from the second string, `abc`.
- The table entries show the minimum number of operations (insertions, deletions, or substitutions) required to transform the substring at row `i` into the substring at column `j`.

Here, operations are marked with `ins` for insertions, `del` for deletions, and `sub` for substitutions.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>b</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>c</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

### Notes
- A `0` in the table indicates no operations are required for that position.
- An `ins` indicates an insertion was made to match the character at row `i`.
- A `del` indicates a deletion was made to remove the character at row `i`.
- An `sub` indicates a substitution was made to change the character at row `i`.

This table shows the dynamic programming approach to calculate the edit distance between two strings.
### Example

- **Edit Distance**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>1</td>
<td>b</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>c</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
Computation of the edit operations

Algorithm edit_operations (i,j)
Input: matrix D (computed)
1 if i = 0 and j = 0 then return
2 if i ≠ 0 and D[i,j] = D[i −1 , j] + 1
3 then „delete a[i]“
4 edit_operations (i − 1, j)
5 else if j ≠ 0 and D[i,j] = D[i, j −1] + 1
6 then „insert b[j]“
7 edit_operations (i, j − 1)
8 else /* D[i,j] = D[i −1, j −1] + c(a[i], b[j]) */
9 „replace a[i] by b[j]“
10 edit_operations (i − 1, j − 1)

Initial call: edit_operations(m,n)
Trace graph of the edit operations

\[ B = \begin{align*}
    &\text{a} &\text{b} &\text{a} &\text{c} \\
    &0 &1 &2 &3 &4 \\
\end{align*} \]

\[ A = \begin{align*}
    &a &b &c \\
    &0 &1 &1 &1 \\
    &1 &1 &1 &2 \\
    &2 &1 &2 &2 \\
    &3 &2 &2 &3 \\
    &4 &3 &3 &2 \\
\end{align*} \]
Trace graph: All possible traces which transform A into B, directed edges from vertex \((i, j)\) to \((i + 1, j)\), \((i, j + 1)\) and \((i + 1, j + 1)\).

Weights of the edges represent the edit costs.

Costs are monotonically increasing along an optimal path.

Each path from the upper left corner to the lower right corner represents an optimal trace.
Approximate string matching

- **Given:** two strings $P = p_1p_2 \ldots p_m$ (pattern) and $T = t_1t_2 \ldots t_n$ (text)

- **Goal:** an interval $[j', j]$, $1 \leq j' \leq j \leq n$, such that the substring $T_{j', j} = t_{j'} \ldots t_j$ of $T$ is most similar to pattern $P$, i.e. for all other intervals $[k', k]$, $1 \leq k' \leq k \leq n$:

$$D(P, T_{j', j}) \leq D(P, T_{k', k})$$
Approximate string matching

Naïve approach:

\[
\text{for all } 1 \leq j' \leq j \leq n \text{ do}
\]
\[
\quad \text{compute } D(P, T_{j', j})
\]
\[
\quad \text{choose minimum}
\]
Approximate string matching

Consider a related problem:

For each text position $j$ and each pattern position $i$ compute the edit distance of the substring $T_{j,.j}$ of $T$ ending at $j$ which is most similar to $P_i$. 
Approximate string matching

Method:
for all $1 \leq j \leq n$ do
    compute $j'$ such that $D(P, T_{j'}, j)$ is minimal

For $1 \leq i \leq m$ and $0 \leq j \leq n$ let:

$$E_{i,j} = \min_{1 \leq j' \leq j+1} D(P_i, T_{j', j})$$

Optimal trace:

$$P_i = \begin{array}{ccccccccccc}
    b & a & a & c & a & a & b & c \\
\end{array}$$

$$T_{j', j} = \begin{array}{ccccccccccc}
    b & a & c & b & c & a & c \\
\end{array}$$
Approximate string matching

Recurrence relation:

\[
E_{i,j} = \min \left\{ \begin{array}{l}
E_{i-1,j-1} + c(p_i,t_j), \\
E_{i-1,j} + 1, \\
E_{i,j-1} + 1
\end{array} \right\}
\]

Remark:

\( j' \) can be completely different for \( E_{i-1,j-1}, E_{i-1,j} \) and \( E_{i,j-1} \).

A subtrace of an optimal trace is an optimal subtrace.
Approximate string matching

Base cases:

\[ E_{0,0} = E(\varepsilon, \varepsilon) = 0 \]
\[ E_{i,0} = E(P_j, \varepsilon) = i \]

but

\[ E_{0,j} = E(\varepsilon, T_j) = 0 \]

Observation:
The optimal edit sequence from P to \( T_{j'} \) does not start with an insertion of \( t_{j'} \).
Approximate string matching

Dependency graph

$T = \begin{array}{ccccccc}
a & b & b & d & a & d & c & b & c \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}$

$P = \begin{array}{ccccccc}
a & b & b & d & a & d & c & b & c \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}$
Approximate string matching

**Theorem**

If there is a path from $E_{0,j-1}$ to $E_{i,j}$ in the dependency graph, then $T_{j',j}$ is a substring of $T$ ending in $j$ which is most similar to $P_i$ and

$$D(P_i, T_{j',j}) = E_{i,j}$$