Edit Distance with Single-Symbol Combinations and Splits

Manolis Christodoulakis\textsuperscript{1} and Gerhard Brey\textsuperscript{2}

\textsuperscript{1}School of Computing & Technology, University of East London
m.christodoulakis@uel.ac.uk

\textsuperscript{2}Centre for Computing in the Humanities, Kings College London
gerhard.brey@kcl.ac.uk

Prague Stringology Conference 2008
NCSE – Our Data Repository

- Nineteenth-Century Serials Edition (NCSE)
  http://www.ncse.ac.uk
- Collaborative project
  - Birkbeck College, London
  - British Library
  - King’s College London (Centre for Computing in the Humanities)
  - Olive Software
- Digital edition of 6 newspapers / periodicals published between 1806–1890
- Microfilmed versions of the newspapers were scanned and OCR’d by Olive Software
OCR Quality

- OCR quality varies widely across corpus
Approximate Pattern Matching

How to distinguish good approximate matches from random ones?

Example

Let the pattern be "Billington".

Matches Edit Distance True match?

- Billmgton: Edit distance 2; must be ✓
- Wellington: Edit distance 2; not really ×

Remarks

$m$ optically resembles $m$; $m$ is essentially a merge of $i$ and $n$. 
The Heart of the Problem

Approximate Pattern Matching

How to distinguish good approximate matches from random ones?

Example

Let the pattern be Billington.
The Heart of the Problem

Approximate Pattern Matching

How to distinguish good approximate matches from random ones?

Example

Let the pattern be Billington.

<table>
<thead>
<tr>
<th>Matches</th>
<th>Edit Distance</th>
<th>True match?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Billmgton</td>
<td>2</td>
<td>Must be ✓</td>
</tr>
</tbody>
</table>

Remarks

$m$ optically resembles $m$; in $m$ is essentially a merge of $i$ and $n$. 

M. Christodoulakis, G. Brey

Edit Distance with Single-Symbol Combinations and Splits
Approximate Pattern Matching

How to distinguish good approximate matches from random ones?

Example

Let the pattern be Billington.

<table>
<thead>
<tr>
<th>Matches</th>
<th>Edit Distance</th>
<th>True match?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Billmgton</td>
<td>2</td>
<td>Must be ✓</td>
</tr>
<tr>
<td>Wellington</td>
<td>2</td>
<td>Not really ✗</td>
</tr>
</tbody>
</table>

Remarks

$m$ optically resembles $m$ is essentially a merge of $i$ and $n$
The Heart of the Problem

Approximate Pattern Matching

How to distinguish good approximate matches from random ones?

Example

Let the pattern be Billington.

<table>
<thead>
<tr>
<th>Matches</th>
<th>Edit Distance</th>
<th>True match?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Billmgton</td>
<td>2</td>
<td>Must be ✓</td>
</tr>
<tr>
<td>Wellington</td>
<td>2</td>
<td>Not really ×</td>
</tr>
</tbody>
</table>

Remarks

- m optically resembles in
- m is essentially a merge of i and n
An enhanced edit-distance algorithm that allows:

- Symbols that look similar to get smaller penalty
- Groups of symbols to be combined and matched against a symbol that looks similar to the combination

We introduce a new edit-distance operation:

**Definition**

The single-symbol combination (hereafter: a combination) operation allows a string, e.g. "in" to be matched against a single symbol, e.g. "m". The dual of a combination is the split operation.
An enhanced edit-distance algorithm that allows:

- symbols that look similar to get smaller penalty
An enhanced edit-distance algorithm that allows:
- symbols that look similar to get smaller penalty
- groups of symbols to be combined and matched against a symbol that looks similar to the combination
An enhanced edit-distance algorithm that allows:
- symbols that look similar to get smaller penalty
- groups of symbols to be combined and matched against a symbol that looks similar to the combination

We introduce a new edit-distance operation:

The single-symbol combination (hereafter: a combination) operation allows a string, e.g. “in” to be matched against a single symbol, e.g. “m”. The dual of a combination is the split operation.
Combination Operation in DP

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>B</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>m</strong></td>
</tr>
<tr>
<td><strong>i</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>l</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>l</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>m</strong></td>
</tr>
<tr>
<td><strong>g</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>t</strong></td>
</tr>
<tr>
<td><strong>n</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>o</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>60</th>
<th>70</th>
<th>80</th>
<th>90</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>B</strong></td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>20</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td>60</td>
<td>70</td>
<td>80</td>
</tr>
<tr>
<td><strong>i</strong></td>
<td>20</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>20</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td>60</td>
<td>70</td>
</tr>
<tr>
<td><strong>l</strong></td>
<td>30</td>
<td>20</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>20</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td>60</td>
</tr>
<tr>
<td><strong>g</strong></td>
<td>40</td>
<td>30</td>
<td>20</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>20</td>
<td>30</td>
<td>40</td>
<td>50</td>
</tr>
<tr>
<td><strong>n</strong></td>
<td>50</td>
<td>40</td>
<td>30</td>
<td>20</td>
<td>10</td>
<td>10</td>
<td>20</td>
<td>30</td>
<td>40</td>
<td>50</td>
</tr>
<tr>
<td><strong>n</strong></td>
<td>60</td>
<td>50</td>
<td>40</td>
<td>30</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[d_{sub} = 10, \ d_{indel} = 10, \ d_{comb} = 1\]
Combination Operation in DP

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Billmington</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>0</strong></td>
<td>0</td>
<td>10</td>
<td>20</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td>60</td>
<td>70</td>
<td>80</td>
<td>90</td>
</tr>
<tr>
<td><strong>1 B</strong></td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>20</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td>60</td>
<td>70</td>
<td>80</td>
</tr>
<tr>
<td><strong>2 i</strong></td>
<td>20</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>20</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td>60</td>
<td>70</td>
</tr>
<tr>
<td><strong>3 l</strong></td>
<td>30</td>
<td>20</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>20</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td>60</td>
</tr>
<tr>
<td><strong>4 l</strong></td>
<td>40</td>
<td>30</td>
<td>20</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>20</td>
<td>30</td>
<td>40</td>
<td>50</td>
</tr>
<tr>
<td><strong>5 i</strong></td>
<td>50</td>
<td>40</td>
<td>30</td>
<td>20</td>
<td>10</td>
<td>10</td>
<td>20</td>
<td>30</td>
<td>40</td>
<td>50</td>
</tr>
<tr>
<td><strong>6 n</strong></td>
<td>60</td>
<td>50</td>
<td>40</td>
<td>30</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>7 g</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>8 t</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>9 o</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>10 n</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

d_{sub} = 10, d_{indel} = 10, d_{comb} = 1

Combination Operation

Find the suffix (if any) of $x[1..i]$ that can be combined into symbol $y[j]$ with the **minimum** cost.
Combination Operation in DP

Combination Operation

Find the suffix (if any) of \(x[1..i]\) that can be combined into symbol \(y[j]\) with the minimum cost.

\[
d_{sub} = 10, \quad d_{indel} = 10, \quad d_{comb} = 1
\]
Combination Operation in DP

**Combination Operation**

Find the suffix (if any) of $x[1..i]$ that can be combined into symbol $y[j]$ with the minimum cost.

**Problem!**

$O(n)$ suffixes per cell!

$d_{sub} = 10$, $d_{indel} = 10$, $d_{comb} = 1$
Combination Operation in DP

Combination Operation

Find the suffix (if any) of $x[1..i]$ that can be combined into symbol $y[j]$ with the minimum cost.

Problem!

$O(n)$ suffixes per cell!

Idea

Preprocess the valid combinations.

$$d_{sub} = 10, \ d_{indel} = 10, \ d_{comb} = 1$$
Combination Lists

Assumptions

For every symbol $\alpha$, a (possibly empty) list is provided, call it $C_\alpha$ (the combination list of $\alpha$), that contains all the strings that resemble $\alpha$. 
Assumptions

For every symbol \( \alpha \), a (possibly empty) list is provided, call it \( C_\alpha \) (the combination list of \( \alpha \)), that contains all the strings that resemble \( \alpha \).

Preprocessing Algorithm

For each symbol \( \alpha \)

1. Build Aho-Corasick automaton, \( T_\alpha \), from the list \( C_\alpha \)
2. Replace the output values of final nodes with the lengths of the strings matched at those nodes
Combination Lists

\( T_m \) built from the combination list \( C_m = \{iii, iin, in, ni, nn, rn, rri\} \)
Incorporating the Combination Lists

\[
\begin{array}{cccccccccc}
  & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
0 & 0 & 10 & 20 & 30 & 40 & 50 & 60 & 70 & 80 & 90 \\
1 B & 10 & 0 & 10 & 20 & 30 & 40 & 50 & 60 & 70 & 80 \\
2 i & 20 & 10 & 0 & 10 & 20 & 30 & 40 & 50 & 60 & 70 \\
3 l & 30 & 20 & 10 & 0 & 10 & 20 & 30 & 40 & 50 & 60 \\
4 l & 40 & 30 & 20 & 10 & 0 & 10 & 20 & 30 & 40 & 50 \\
5 i & 50 & 40 & 30 & 20 & 10 & 10 & 20 & 30 & 40 & 50 \\
6 n & 60 & 50 & 40 & 30 & 20 & & & & & \\
7 g & & & & & & & & & & \\
8 t & & & & & & & & & & \\
9 o & & & & & & & & & & \\
10 n & & & & & & & & & & \\
\end{array}
\]

\[d_{sub} = 10, \quad d_{indel} = 10, \quad d_{comb} = 1\]
Incorporating the Combination Lists

\[ d_{sub} = 10, \quad d_{indel} = 10, \quad d_{comb} = 1 \]

Remark 1

For cell \( D(6, 5) \) we need to spell out “Billin” on \( T_m \)

Remark 2

For cell \( D(7, 5) \) we need to spell out “Billing” on \( T_m \) (only one letter longer than the previous)

Idea

Store a pointer to the node of \( T_m \) where “Billin” ended. When processing \( D(7, 5) \), proceed from that node following the letter ‘g’ only.
**Incorporating the Combination Lists**

\[
d_{sub} = 10, \quad d_{indel} = 10, \quad d_{comb} = 1
\]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Billington</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>10</td>
<td>20</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td>60</td>
<td>70</td>
<td>80</td>
<td>90</td>
</tr>
<tr>
<td>1 B</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>20</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td>60</td>
<td>70</td>
<td>80</td>
</tr>
<tr>
<td>2 i</td>
<td>20</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>20</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td>60</td>
<td>70</td>
</tr>
<tr>
<td>3 l</td>
<td>30</td>
<td>20</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>20</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td>60</td>
</tr>
<tr>
<td>4 l</td>
<td>40</td>
<td>30</td>
<td>20</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>20</td>
<td>30</td>
<td>40</td>
<td>50</td>
</tr>
<tr>
<td>5 i</td>
<td>50</td>
<td>40</td>
<td>30</td>
<td>20</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>20</td>
<td>30</td>
<td>40</td>
</tr>
<tr>
<td>6 n</td>
<td>60</td>
<td>50</td>
<td>40</td>
<td>30</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7 g</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 t</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9 o</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 n</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Remark 1**

For cell \(D(6, 5)\) we need to spell out “Billin” on \(T_m\)

**Remark 2**

For cell \(D(7, 5)\) we need to spell out “Billing” on \(T_m\) (only one letter longer than the previous)
Incorporating the Combination Lists

Remark 1
For cell $D(6, 5)$ we need to spell out “Billin” on $T_m$

Remark 2
For cell $D(7, 5)$ we need to spell out “Billing” on $T_m$ (only one letter longer than the previous)

Idea
Store a pointer to the node of $T_m$ where “Billin” ended. When processing $D(7, 5)$ proceed from that node following the letter ‘g’ only.

$d_{sub} = 10, \ d_{indel} = 10, \ d_{comb} = 1$
Incorporating the Combination Lists

\[ d_{\text{sub}} = 10, \ d_{\text{indel}} = 10, \ d_{\text{comb}} = 1 \]
Incorporating the Combination Lists

\[
d_{\text{sub}} = 10, \quad d_{\text{indel}} = 10, \quad d_{\text{comb}} = 1
\]
Incorporating the Combination Lists

\[ d_{sub} = 10, \quad d_{indel} = 10, \quad d_{comb} = 1 \]
Running Time and Space

Preprocessing Time

\[ O\left(\sum_{\forall \alpha} \ell_\alpha\right) \]
where \( \ell_\alpha = \sum_{x \in C_\alpha} |x| \)
Running Time and Space

### Preprocessing Time

\[ O\left(\sum_{\forall \alpha} \ell_{\alpha}\right) \quad \text{where} \quad \ell_{\alpha} = \sum_{x \in C_{\alpha}} |x| \]

### Edit Distance Algorithm Time

\[ O(mnk) \quad \text{where} \quad k \text{ is an upper bound on } |C_{\alpha}| \text{ over all } \alpha \]
Running Time and Space

Preprocessing Time

\[ O\left(\sum_{\forall \alpha} \ell_\alpha\right) \quad \text{where} \quad \ell_\alpha = \sum_{x \in C_\alpha} |x| \]

Edit Distance Algorithm Time

\[ O(mnk) \quad \text{where} \quad k \text{ is an upper bound on } |C_\alpha| \text{ over all } \alpha \]

Space Requirements

\[ O\left(mn + \sum_{\forall \alpha} \ell_\alpha\right) \]
Open Problems

Current Algorithm

Single-symbol combinations allow a string (e.g. “in”) to be matched against a single symbol (e.g. “m”).
The Problem

The Algorithm

Open Problems

Current Algorithm

Single-symbol combinations allow a string (e.g. “in”) to be matched against a single symbol (e.g. “m”).

Definition (Edit Distance with Re-Combinations)

How about allowing a string (e.g. “in”) to be matched against another string (e.g. “ni”)?

Definition (Edit Distance with Transitive Combinations)

Let “in” ∈ C^m and “ri” ∈ C^n. Write an algorithm that can infer that “iri” resembles “m” despite the fact that “iri” /∈ C^m.
### Open Problems

#### Current Algorithm

**Single-symbol combinations** allow a string (e.g. “in”) to be matched against a *single* symbol (e.g. “m”).

#### Definition (Edit Distance with Re-Combinations)

How about allowing a string (e.g. “in”) to be matched against another string (e.g. “ni”)?

#### Definition (Edit Distance with Transitive Combinations)

Let “in” ∈ $C_m$ and “ri” $C_n$. Write an algorithm that can infer that “iri” resembles “m” despite the fact that “iri” $\notin C_m$. 

---

M. Christodoulakis, G. Brey

*Edit Distance with Single-Symbol Combinations and Splits*