Fast search for similar words

Klaus U. Schulz
CIS - LMU Munich

joint work with

Stoyan Mihov
Bulgarian Academy of Science
Problem

Given a large lexicon $D$ and possibly erroneous input tokens, for each input token $v$ efficiently find the „most similar“ entries $w$ of the lexicon.
Problem

Given a large lexicon $D$ and possibly erroneous input tokens, for each input token $v$ efficiently find the "most similar" entries $w$ of the lexicon.

Text correction

Input: text with orthographic errors (e.g., from OCR recognition). For each non-lexical token $v$ find good correction suggestions in the lexicon.
Problem

Given a large lexicon $D$ and possibly erroneous input tokens, for each input token $v$ efficiently find the "most similar" entries $w$ of the lexicon.

Text correction
Input: text with orthographic errors (e.g., from OCR recognition). For each non-lexical token $v$ find good correction suggestions in the lexicon.

Correction of search queries
"Lexicon": list of known search queries
Input: new queries $v$ possibly with misspellings.
If $v$ not known, check if there are similar known queries.
Notions of similarity for words

Levenshtein distance (edit distance)
between two words v, w: minimal number of letter deletions, insertions or substitutions needed to rewrite v into w.

E.g. \(d(\text{chold}, \text{child})=1\), \(d(\text{cold}, \text{child})=2\).

Variants
1. Allow for other “edit operations”: transpositions of letters, merge of two letters, splitting one letter into two,…
2. Assign non-uniform “costs” to the operations depending on the symbols used. E.g. \(\text{cost}(l,i)=0.35\), \(\text{cost}(l,m)=1\), \(\text{cost}(m,i\text{n})=0.22\).
Formalized problem

Given

• large lexicon $D$,
• small bound $n$ for the Levenshtein distance $d$,

for each input token $v$ efficiently find all entries $w$ of $D$ such that $d(v,w) \leq n$.

(Later consider variants of Levenshtein distance)
Structure of talk

1. Efficient solution for „Decision Problem“:
   Given two words $v$, $w$, decide if $d(v,w) \leq n$.

2. Efficient solution for similarity search in very large lexica

3. Refinement with substantially improved efficiency
Decision Problem
Standard solution
Decision Problem
Standard solution

Wagner-Fischer Method:
Compute Levenshtein distance $d(v,w)$ via dynamic programming
Wagner-Fischer Method:
Compute Levenshtein distance $d(v,w)$ via dynamic programming

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>r</th>
<th>k</th>
<th>a</th>
<th>n</th>
<th>s</th>
<th>a</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>n</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>a</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>n</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>a</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>s</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>
**Decision Problem**

**Standard solution**

Wagner-Fischer Method:
Compute Levenshtein distance $d(v,w)$ via dynamic programming

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>r</th>
<th>k</th>
<th>a</th>
<th>n</th>
<th>s</th>
<th>a</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>n</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>a</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>n</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>a</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>s</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>
Wagner-Fischer Method:
Compute Levenshtein distance $d(v,w)$ via dynamic programming

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>r</th>
<th>k</th>
<th>a</th>
<th>n</th>
<th>s</th>
<th>a</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>n</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>a</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>n</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>a</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>s</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>
Decision Problem
Use of nondeterministic “Levenshtein” automata
Decision Problem
Use of nondeterministic “Levenshtein” automata

For each input token v, use a nondeterministic automaton $A_n$ to recognize all words w such that $d(v,w) \leq n$.

E.g. $v = \text{chold}$, $n=2$:
Decision Problem
Use of nondeterministic “Levenshtein” automata

Problems

1. Nondeterminism of automata

2. Need special automaton for each garbled input word v
Triangular areas

Given automaton $A_n$ for each partial input $u$ of length $k$ the „set of active states“ is always subset of $k$-th „triangular area“.
Triangular areas

Empty input
Triangular areas

Input „h“
Triangular areas

Input „hc“
Triangular areas

Input „hch“
Triangular areas

Input „hchi“
Input behaviour
Finding next set of active states

Given set of active states and input symbol $\sigma$, next set of active states only depends on distribution of $\sigma$ in subword of pattern „chold“
Input behaviour
Finding next set of active states

Empty input
Input behaviour
Finding next set of active states

Empty input, next letter h
Input behaviour
Finding next set of active states

Empty input, next letter h

Relevant subword:
Input behaviour
Finding next set of active states

Empty input, next letter h

Relevant subword:
Input behaviour
Finding next set of active states

Empty input, next letter h

Relevant subword:
Input behaviour
Finding next set of active states

Input „h“
Input behaviour
Finding next set of active states

Input „h“, next letter c

Relevant subword:
Input behaviour
Finding next set of active states

Input „h“, next letter c

Relevant subword:
Input behaviour
Finding next set of active states

Input „hc“
Input behaviour
Finding next set of active states

Input „hc“ next letter h

Relevant subword:
Input behaviour
Finding next set of active states

Input „hch“
Input behaviour
Finding next set of active states

Input „hch“, next letter i
Input behaviour
Finding next set of active states

Input "hch", next letter i
Input behaviour
Finding next set of active states

Input „hchi“
Summing up

1. Introduce a new finite set of „universal“ states: subsets of triangle (plus second type of universal final states)

Obtain universal set of states not dependent on input word v.
2. Introduce **universal transition function** dependent only on distribution of „red“ and „green“ horizontal transitions.
2. Introduce **universal transition function** dependent only on distribution of „red“ and „green“ horizontal transitions

Encode red/green by bitvector of symbols 0/1
Summing up

2. Introduce **universal transition function** dependent only on distribution of „red“ and „green“ horizontal transitions

Encode red/green by bitvector of symbols 0/1
Summing up

Obtain a **universal and deterministic automaton** for given bound n.

Given input word v and another word w, each letter σ of w is translated into a bitvektor χ encoding occurrences of σ in relevant subwords of v.

Bitvectors used as input „symbols“ for universal automaton.

Universal automaton accepts sequence of bitvectors iff d(v,w) ≤ n.
Example

Bound n=1
Garbled input word v="chold"
Represented as $chold

Second word w=child
Translated into series

$\chi(c,$cho$)=0100$
$\chi(h,$chol$)=0100$
$\chi(i,$hold$)=0000$
$\chi(l,$old$)=010$
$\chi(d,$ld$)=01$
Example

Bound n=1
Garbled input word v="chold"
Represented as $chold

Second word w=child
Translated into series

\( \chi(c,\$cho)=0100 \)
\( \chi(h,\text{chol})=0100 \)
\( \chi(i,\text{hold})=0000 \)
\( \chi(l,\text{old})=010 \)
\( \chi(d,\text{ld})=01 \)
Example

Bound n=1
Garbled input word v=“chold”
Represented as $chold

Second word w=child
Translated into series

\[ \chi(c,cho)=0100 \]
\[ \chi(h,chol)=0100 \]
\[ \chi(i,hold)=0000 \]
\[ \chi(l,old)=010 \]
\[ \chi(d,ld)=01 \]
Example

Bound n=1
Garbled input word v=“chold“
Represented as $chold$

Second word w=child
Translated into series

\[
\chi(c,$cho$)=0100 \\
\chi(h,chol)=0100 \\
\chi(i,hold)=0000 \\
\chi(l,old)=010 \\
\chi(d,ld)=01
\]
Example

Bound \( n=1 \)
Garbled input word \( v=\text{"chold"} \)
Represented as \( \$\text{chold} \)

Second word \( w=\text{child} \)
Translated into series

\[
\chi(c,\$\text{cho})=0100 \\
\chi(h,\text{chol})=0100 \\
\chi(i,\text{hold})=0000 \\
\chi(l,\text{old})=010 \\
\chi(d,ld)=01
\]
Example

Bound n=1
Garbled input word v="chold"
Represented as $chold

Second word w=child
Translated into series

$\chi(c,$cho)=0100$
$\chi(h,$chol)=0100$
$\chi(i,$hold)=0000$
$\chi(l,$old)=010$
$\chi(d,$ld)=01$
New solution for „Decision Problem“

Universal deterministic Levenshtein automata

- For each distance bound $n$, a universal deterministic Levenshtein automaton is computed offline (once).

- Offers very efficient method to decide for arbitrary input words $v,w$ if $d(v,w) \leq n$.

- Size dependent on bound $n$. In practice: $n < 8$. 
2 Similarity search in large lexicon

Main ideas

Represent lexicon as deterministic finite-state automaton

• Given pattern $v$, start a complete traversal of lexicon automaton but use universal Levenshtein automaton as control:
  • Translate letters of lexicon words into bitvectors using pattern $v$.
  • Failure in the Levenshtein automaton causes failure and backtracking for search
• If in both automata a final state is reached, output lexicon word.
Example

Lexicon: axis, behave, child, church, cold, hate, hold
Example: input “chold”, n=1

Figure 6
The universal deterministic Levenshtein automaton $A^\gamma(1)$. See Example 4 for notation.
Example: input “chold”, n=1

Figure 6
The universal deterministic Levenshtein automaton $A^y(1)$. See Example 4 for notation.
Example: input “chold”, n=1

$\chi(a, $cho$) = 0000$

Figure 6
The universal deterministic Levenshtein automaton $A^V(1)$. See Example 4 for notation.
Example: input “chold”, n=1
Example: input “chold”, n=1

\[ \chi(x, \text{chol}) = 0000 \]
Example: input “chold”, n=1

\[ \chi(x, \text{chol}) = 0000 \]

No transition with 0000

Figure 6
The universal deterministic Levenshtein automaton \( A^\chi(1) \). See Example 4 for notation.
Example: input “chold”, n=1

Figure 6
The universal deterministic Levenshtein automaton $A^n(1)$. See Example 4 for notation.
Example: input “chold”, n=1

\[ \chi(b, \text{cho}) = 0000 \]

Figure 6
The universal deterministic Levenshtein automaton \( A^\chi(1) \). See Example 4 for notation.
Example: input “chold”, n=1

Figure 6
The universal deterministic Levenshtein automaton $A^y(1)$. See Example 4 for notation.
Example: input “chold”, n=1

\[ \chi(e, chol) = 0000 \]
Example: input "chold", n=1

\[ \chi(e, \text{chol}) = 0000 \]

No transition with 0000
Example: input “chold”, n=1
Example: input “chold”, n=1

\[ \chi(c, \text{cho}) = 0100 \]
Example: input “chold”, n=1

Figure 6
The universal deterministic Levenshtein automaton $A^n(1)$. See Example 4 for notation.
Example: input “chold”, n=1

\[ \chi(h,\text{chol})=0100 \]
Example: input “chold”, n=1

Figure 6
The universal deterministic Levenshtein automaton $A^\gamma(1)$. See Example 4 for notation.
Example: input “chold”, n=1

\[ \chi(i, \text{hold}) = 0000 \]

Figure 6
The universal deterministic Levenshtein automaton \( A^\forall(1) \). See Example 4 for notation.
Example: input “chold”, n=1

Figure 6
The universal deterministic Levenshtein automaton $A'(1)$. See Example 4 for notation.
Example: input "chold", n=1

\[ \chi(l, old) = 010 \]
Example: input “chold”, n=1

Figure 6
The universal deterministic Levenshtein automaton $A^\gamma(1)$. See Example 4 for notation.
Example: input “chold”, n=1

$\chi(d, l'd)=01$

Figure 6
The universal deterministic Levenshtein automaton $A^0(1)$. See Example 4 for notation.
Example: input “chold”, n=1

Figure 6
The universal deterministic Levenshtein automaton $A^\gamma(1)$. See Example 4 for notation.
Example: input “chold”, n=1

First output: child
Example: input “chold”, n=1

Figure 6
The universal deterministic Levenshtein automaton $A^Y(1)$. See Example 4 for notation.
Example: input “chold”, n=1

\[ \chi(u, \text{hold}) = 0000 \]
Example: input “chold”, n=1

Figure 6
The universal deterministic Levenshtein automaton $A^\gamma(1)$. See Example 4 for notation.
Example: input “chold”, n=1

χ(r, old) = 000

Figure 6
The universal deterministic Levenshtein automaton A⁺(1). See Example 4 for notation.
Example: input “chold”, n=1

\chi(r,old)=000

No transition with 000
Example: input “chold”, n=1

Figure 6
The universal deterministic Levenshtein automaton $A^\lambda(1)$. See Example 4 for notation.
Example: input “chold”, n=1

$\chi(o, chol) = 0010$

Figure 6
The universal deterministic Levenshtein automaton $A^\gamma(1)$. See Example 4 for notation.
Example: input “chold”, n=1

Figure 6
The universal deterministic Levenshtein automaton $A^\gamma(1)$. See Example 4 for notation.
Example: input “chold”, n=1

\[ \chi(\text{l, hold}) = 0010 \]
Example: input “chold”, n=1

Figure 6
The universal deterministic Levenshtein automaton $A^*(1)$. See Example 4 for notation.
Example: input “chold”, n=1

\[ \chi(d,old)=001 \]
Example: input “chold”, n=1
Example: input “chold”, n=1

2nd output: cold

Figure 6
The universal deterministic Levenshtein automaton $A^V(1)$. See Example 4 for notation.
Example: input “chold”, n=1
Example: input “chold”, $n=1$

$\chi(h,\text{cho})=0010$

Figure 6
The universal deterministic Levenshtein automaton $A^\nu(1)$. See Example 4 for notation.
Example: input "chold", n=1

Figure 6
The universal deterministic Levenshtein automaton $A^Y(1)$. See Example 4 for notation.
Example: input “chold”, n=1

\[ \chi(o, chol) = 0010 \]

Figure 6
The universal deterministic Levenshtein automaton \( A^\gamma(1) \). See Example 4 for notation.
Example: input “chold”, n=1
Example: input “chold”, n=1

\[ \chi(l, hold) = 0010 \]
Example: input “chold”, n=1

Figure 6
The universal deterministic Levenshtein automaton $A^\gamma(1)$. See Example 4 for notation.
Example: input “chold”, n=1

\[ \chi(d,old)=001 \]
Example: input “chold”, n=1
Example: input “chold”, n=1

3rd output: hold

Figure 6
The universal deterministic Levenshtein automaton $A^\gamma(1)$. See Example 4 for notation.
Example: input “chold”, n=1
Example: input “chold”, n=1

\[ \chi(a, \text{hold}) = 0000 \]

No transition with 0000
Example: input “chold”, n=1

No transition with 0000

Figure 6
The universal deterministic Levenshtein automaton $A^Y(1)$. See Example 4 for notation.
Example: input “chold”, n=1

Figure 6
The universal deterministic Levenshtein automaton $A^v(1)$. See Example 4 for notation.
Evaluation
Evaluation

Three lexica BL (Bulgarian), GL (German), TL (Book Titles)

<table>
<thead>
<tr>
<th></th>
<th>BL</th>
<th>GL</th>
<th>TL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of words</td>
<td>956,339</td>
<td>3,871,605</td>
<td>1,200,073</td>
</tr>
<tr>
<td>Automaton states</td>
<td>39,339</td>
<td>4,068,189</td>
<td>29,103,779</td>
</tr>
<tr>
<td>Automaton transitions</td>
<td>102,585</td>
<td>6,954,377</td>
<td>30,252,173</td>
</tr>
<tr>
<td>Size (Bytes)</td>
<td>1,191,548</td>
<td>90,206,665</td>
<td>475,615,320</td>
</tr>
</tbody>
</table>
## Evaluation

<table>
<thead>
<tr>
<th>Length</th>
<th>(CT1)</th>
<th>(NC1)</th>
<th>(CT2)</th>
<th>(NC2)</th>
<th>(CT3)</th>
<th>(NC3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.107</td>
<td>12.03</td>
<td>0.974</td>
<td>285.4</td>
<td>4.589</td>
<td>2983.2</td>
</tr>
<tr>
<td>4</td>
<td>0.098</td>
<td>8.326</td>
<td>1.048</td>
<td>192.1</td>
<td>5.087</td>
<td>2426.6</td>
</tr>
<tr>
<td>5</td>
<td>0.085</td>
<td>5.187</td>
<td>1.086</td>
<td>105.0</td>
<td>5.424</td>
<td>1466.5</td>
</tr>
<tr>
<td>6</td>
<td>0.079</td>
<td>4.087</td>
<td>0.964</td>
<td>63.29</td>
<td>5.454</td>
<td>822.77</td>
</tr>
<tr>
<td>7</td>
<td>0.079</td>
<td>3.408</td>
<td>0.853</td>
<td>40.95</td>
<td>5.426</td>
<td>466.86</td>
</tr>
<tr>
<td>8</td>
<td>0.081</td>
<td>3.099</td>
<td>0.809</td>
<td>30.35</td>
<td>5.101</td>
<td>294.84</td>
</tr>
<tr>
<td>9</td>
<td>0.083</td>
<td>2.707</td>
<td>0.824</td>
<td>22.36</td>
<td>4.631</td>
<td>187.12</td>
</tr>
<tr>
<td>10</td>
<td>0.088</td>
<td>2.330</td>
<td>0.794</td>
<td>16.83</td>
<td>4.410</td>
<td>121.73</td>
</tr>
<tr>
<td>11</td>
<td>0.088</td>
<td>1.981</td>
<td>0.821</td>
<td>12.74</td>
<td>4.311</td>
<td>81.090</td>
</tr>
<tr>
<td>12</td>
<td>0.088</td>
<td>1.633</td>
<td>0.831</td>
<td>9.252</td>
<td>4.277</td>
<td>51.591</td>
</tr>
<tr>
<td>13</td>
<td>0.089</td>
<td>1.337</td>
<td>0.824</td>
<td>6.593</td>
<td>4.262</td>
<td>31.405</td>
</tr>
<tr>
<td>14</td>
<td>0.089</td>
<td>1.129</td>
<td>0.844</td>
<td>4.824</td>
<td>4.251</td>
<td>19.187</td>
</tr>
<tr>
<td>15</td>
<td>0.089</td>
<td>0.970</td>
<td>0.816</td>
<td>3.748</td>
<td>4.205</td>
<td>12.337</td>
</tr>
<tr>
<td>16</td>
<td>0.087</td>
<td>0.848</td>
<td>0.829</td>
<td>3.094</td>
<td>4.191</td>
<td>9.1752</td>
</tr>
<tr>
<td>17</td>
<td>0.086</td>
<td>0.880</td>
<td>0.805</td>
<td>2.970</td>
<td>4.138</td>
<td>8.1250</td>
</tr>
<tr>
<td>18</td>
<td>0.087</td>
<td>0.809</td>
<td>0.786</td>
<td>2.717</td>
<td>4.117</td>
<td>7.1701</td>
</tr>
<tr>
<td>19</td>
<td>0.087</td>
<td>0.810</td>
<td>0.792</td>
<td>2.646</td>
<td>4.078</td>
<td>6.6544</td>
</tr>
<tr>
<td>20</td>
<td>0.091</td>
<td>0.765</td>
<td>0.795</td>
<td>2.364</td>
<td>4.107</td>
<td>5.7686</td>
</tr>
</tbody>
</table>

**Table 1**
Evaluation results for the basic correction algorithm, Bulgarian dictionary BL, standard Levenshtein distance, and distance bounds $k = 1, 2, 3$. Times in milliseconds.
Evaluation

<table>
<thead>
<tr>
<th>Length</th>
<th>(CT1)</th>
<th>(NC1)</th>
<th>(CT2)</th>
<th>(NC2)</th>
<th>(CT3)</th>
<th>(NC3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 – 14</td>
<td>0.225</td>
<td>0.201</td>
<td>4.140</td>
<td>0.686</td>
<td>23.59</td>
<td>2.345</td>
</tr>
<tr>
<td>15 – 24</td>
<td>0.170</td>
<td>0.605</td>
<td>3.210</td>
<td>1.407</td>
<td>19.66</td>
<td>3.824</td>
</tr>
<tr>
<td>25 – 34</td>
<td>0.249</td>
<td>0.492</td>
<td>4.334</td>
<td>0.938</td>
<td>24.58</td>
<td>1.558</td>
</tr>
<tr>
<td>35 – 44</td>
<td>0.264</td>
<td>0.449</td>
<td>4.316</td>
<td>0.781</td>
<td>24.06</td>
<td>1.187</td>
</tr>
<tr>
<td>45 – 54</td>
<td>0.241</td>
<td>0.518</td>
<td>3.577</td>
<td>0.969</td>
<td>20.18</td>
<td>1.563</td>
</tr>
<tr>
<td>55 – 64</td>
<td>0.233</td>
<td>0.444</td>
<td>3.463</td>
<td>0.644</td>
<td>19.03</td>
<td>0.737</td>
</tr>
</tbody>
</table>

Table 2
Evaluation results for the basic correction algorithm, German dictionary GL, standard Levenshtein distance, and distance bounds $k = 1, 2, 3$. Times in milliseconds.
Evaluation

<table>
<thead>
<tr>
<th>Length</th>
<th>(CT1)</th>
<th>(NC1)</th>
<th>(CT2)</th>
<th>(NC2)</th>
<th>(CT3)</th>
<th>(NC3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 – 14</td>
<td>0.294</td>
<td>0.537</td>
<td>3.885</td>
<td>2.731</td>
<td>19.31</td>
<td>24.67</td>
</tr>
<tr>
<td>15 – 24</td>
<td>0.308</td>
<td>0.451</td>
<td>4.024</td>
<td>0.872</td>
<td>19.50</td>
<td>1.703</td>
</tr>
<tr>
<td>25 – 34</td>
<td>0.321</td>
<td>0.416</td>
<td>4.160</td>
<td>0.644</td>
<td>19.98</td>
<td>0.884</td>
</tr>
<tr>
<td>35 – 44</td>
<td>0.330</td>
<td>0.412</td>
<td>4.225</td>
<td>0.628</td>
<td>20.20</td>
<td>0.844</td>
</tr>
<tr>
<td>45 – 54</td>
<td>0.338</td>
<td>0.414</td>
<td>4.300</td>
<td>0.636</td>
<td>20.44</td>
<td>0.857</td>
</tr>
<tr>
<td>55 – 64</td>
<td>0.344</td>
<td>0.347</td>
<td>4.340</td>
<td>0.433</td>
<td>20.61</td>
<td>0.449</td>
</tr>
</tbody>
</table>

Table 3
Evaluation results for the basic correction algorithm, title dictionary TL, standard Levenshtein distance, and distance bounds $k = 1, 2, 3$. Times in milliseconds.
3. Improving efficiency
3. Improving efficiency

Structure of lexicon automaton
3. Improving efficiency

Structure of lexicon automaton

Enourmous branching close to start
3. Improving efficiency

Structure of lexicon automaton

Enormous branching close to start  Almost no branching deeper inside
3. Improving efficiency

Structure of lexicon automaton

Even for bound n=1, have to visit all transitions and translate labels

Enormous branching close to start  Almost no branching deeper inside
3. Improving efficiency

Main idea, here for n=1:

Input word v

Split v in two subwords v1, v2

If v contains at most 1 error, distinguish two cases:

1. v1 contains 0 errors, v2 at most 1 error
2. v2 contains 0 errors, v1 one error

Subsearch for Case 1: Use standard lexicon automaton as before
Subsearch for Case 2: Use automaton for lexicon of reversed words
(backward lexicon)
3. Improving efficiency

Main idea, here for n=1:

Input word v

Split v in two subwords v1, v2

If v contains at most 1 error, distinguish two cases:

1. v1 contains 0 errors, v2 at most 1 error
2. v2 contains 0 errors, v1 one error

Subsearch for Case 1: Use standard lexicon automaton as before
Subsearch for Case 2: Use automaton for lexicon of reversed words (backward lexicon)

Both cases: no search/backtracking needed in „dense“ part of automaton
3. Improving efficiency

Subsearch 1

Forward lexicon automaton

\[ v1 \text{ 0 errors} \quad v2 \text{ at most 1 error} \]

forward
3. Improving efficiency

Subsearch 2

Backward lexicon automaton

v1
1 error

v2
0 errors

backward
Evaluation

<table>
<thead>
<tr>
<th>Length</th>
<th>(CT1)</th>
<th>Speed-up 1</th>
<th>(CT2)</th>
<th>Speed-up 2</th>
<th>(CT3)</th>
<th>Speed-up 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.031</td>
<td>3.45</td>
<td>0.876</td>
<td>1.11</td>
<td>6.466</td>
<td>0.71</td>
</tr>
<tr>
<td>4</td>
<td>0.027</td>
<td>3.63</td>
<td>0.477</td>
<td>2.2</td>
<td>4.398</td>
<td>1.16</td>
</tr>
<tr>
<td>5</td>
<td>0.018</td>
<td>4.72</td>
<td>0.450</td>
<td>2.41</td>
<td>2.629</td>
<td>2.06</td>
</tr>
<tr>
<td>6</td>
<td>0.016</td>
<td>4.94</td>
<td>0.269</td>
<td>3.58</td>
<td>2.058</td>
<td>2.65</td>
</tr>
<tr>
<td>7</td>
<td>0.011</td>
<td>7.18</td>
<td>0.251</td>
<td>3.4</td>
<td>1.327</td>
<td>4.09</td>
</tr>
<tr>
<td>8</td>
<td>0.012</td>
<td>6.75</td>
<td>0.196</td>
<td>4.13</td>
<td>1.239</td>
<td>4.12</td>
</tr>
<tr>
<td>9</td>
<td>0.009</td>
<td>9.22</td>
<td>0.177</td>
<td>4.66</td>
<td>0.828</td>
<td>5.59</td>
</tr>
<tr>
<td>10</td>
<td>0.010</td>
<td>8.8</td>
<td>0.159</td>
<td>4.99</td>
<td>0.827</td>
<td>5.33</td>
</tr>
<tr>
<td>11</td>
<td>0.008</td>
<td>11.0</td>
<td>0.147</td>
<td>5.59</td>
<td>0.603</td>
<td>7.15</td>
</tr>
<tr>
<td>12</td>
<td>0.008</td>
<td>11.0</td>
<td>0.142</td>
<td>5.85</td>
<td>0.658</td>
<td>6.5</td>
</tr>
<tr>
<td>13</td>
<td>0.006</td>
<td>14.8</td>
<td>0.128</td>
<td>6.44</td>
<td>0.457</td>
<td>9.33</td>
</tr>
<tr>
<td>14</td>
<td>0.006</td>
<td>14.8</td>
<td>0.123</td>
<td>6.86</td>
<td>0.458</td>
<td>9.28</td>
</tr>
<tr>
<td>15</td>
<td>0.005</td>
<td>17.8</td>
<td>0.112</td>
<td>7.29</td>
<td>0.321</td>
<td>13.1</td>
</tr>
<tr>
<td>16</td>
<td>0.005</td>
<td>17.2</td>
<td>0.111</td>
<td>7.47</td>
<td>0.320</td>
<td>13.1</td>
</tr>
<tr>
<td>17</td>
<td>0.005</td>
<td>17.4</td>
<td>0.108</td>
<td>7.45</td>
<td>0.283</td>
<td>14.6</td>
</tr>
<tr>
<td>18</td>
<td>0.005</td>
<td>17.4</td>
<td>0.108</td>
<td>7.28</td>
<td>0.280</td>
<td>14.7</td>
</tr>
<tr>
<td>19</td>
<td>0.004</td>
<td>21.8</td>
<td>0.103</td>
<td>7.69</td>
<td>0.269</td>
<td>15.2</td>
</tr>
<tr>
<td>20</td>
<td>—</td>
<td>—</td>
<td>0.105</td>
<td>7.57</td>
<td>0.274</td>
<td>15.0</td>
</tr>
</tbody>
</table>

Table 4
Evaluation results using the backwards dictionary filtering method, Bulgarian dictionary BL, distance bounds $k = 1, 2, 3$. Times in milliseconds and speed-up factors (ratio of times) w.r.t. basic algorithm.
## Evaluation

<table>
<thead>
<tr>
<th>Length</th>
<th>(CT1)</th>
<th>Speed-up 1</th>
<th>(CT2)</th>
<th>Speed-up 2</th>
<th>(CT3)</th>
<th>Speed-up 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 – 14</td>
<td>0.007</td>
<td>32.1</td>
<td>0.220</td>
<td>18.8</td>
<td>0.665</td>
<td>35.5</td>
</tr>
<tr>
<td>15 – 24</td>
<td>0.010</td>
<td>17.0</td>
<td>0.175</td>
<td>18.3</td>
<td>0.601</td>
<td>32.7</td>
</tr>
<tr>
<td>25 – 34</td>
<td>0.009</td>
<td>27.7</td>
<td>0.221</td>
<td>19.6</td>
<td>0.657</td>
<td>37.4</td>
</tr>
<tr>
<td>35 – 44</td>
<td>0.007</td>
<td>37.7</td>
<td>0.220</td>
<td>19.6</td>
<td>0.590</td>
<td>40.8</td>
</tr>
<tr>
<td>45 – 54</td>
<td>—</td>
<td>—</td>
<td>0.201</td>
<td>17.8</td>
<td>0.452</td>
<td>44.6</td>
</tr>
<tr>
<td>55 – 64</td>
<td>—</td>
<td>—</td>
<td>0.195</td>
<td>17.8</td>
<td>0.390</td>
<td>48.8</td>
</tr>
</tbody>
</table>

Table 5
Evaluation results using the backwards dictionary filtering method, German dictionary GL, distance bounds \( k = 1, 2, 3 \). Times in milliseconds and speed-up factors (ratio of times) w.r.t. basic algorithm.
## Evaluation

<table>
<thead>
<tr>
<th>Length</th>
<th>(CT1)</th>
<th>Speed-up 1</th>
<th>(CT2)</th>
<th>Speed-up 2</th>
<th>(CT3)</th>
<th>Speed-up 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 − 14</td>
<td>0.032</td>
<td>9.19</td>
<td>0.391</td>
<td>9.94</td>
<td>1.543</td>
<td>12.5</td>
</tr>
<tr>
<td>15 − 24</td>
<td>0.019</td>
<td>16.2</td>
<td>0.247</td>
<td>16.3</td>
<td>0.636</td>
<td>30.7</td>
</tr>
<tr>
<td>25 − 34</td>
<td>0.028</td>
<td>11.5</td>
<td>0.260</td>
<td>16.0</td>
<td>0.660</td>
<td>30.3</td>
</tr>
<tr>
<td>35 − 44</td>
<td>0.029</td>
<td>11.4</td>
<td>0.295</td>
<td>14.3</td>
<td>0.704</td>
<td>28.7</td>
</tr>
<tr>
<td>45 − 54</td>
<td>0.037</td>
<td>9.14</td>
<td>0.332</td>
<td>13.0</td>
<td>0.759</td>
<td>26.9</td>
</tr>
<tr>
<td>55 − 64</td>
<td>0.038</td>
<td>9.05</td>
<td>0.343</td>
<td>12.7</td>
<td>0.814</td>
<td>25.3</td>
</tr>
</tbody>
</table>

Table 6
Evaluation results using the backwards dictionary filtering method, title dictionary TL, distance bounds $k = 1, 2, 3$. Times in milliseconds and speed-up factors (ratio of times) w.r.t. basic algorithm.
Generalizations of results

(With Petar Mitankin, Bulg. Ac. Of Science)

Similar results for many variants of Levenshtein distance:

• Adding transpositions as edit operations
• Adding merges and splits as edit operations
• Adding more complex edit operations (III-> in)
• Restricting edit operations to specific symbols or strings
When looking for algorithmic solutions for problems around sequences or strings we tend to only consider a „left-to-right“ search strategy.

Sometimes much better algorithms are obtained when going beyond conventional „left-to-right“.