Fast String Correction with Levenshtein-Automata

Klaus U. Schulz
CIS
University of Munich
schulz@cis.uni-muenchen.de

Stoyan Mihov
Linguistic Modelling Laboratory
LPDP – Bulgarian Academy of Sciences
stoyan@lml.bas.bg

Abstract

The Levenshtein-distance between two words is the minimal number of insertions, deletions or substitutions that are needed to transform one word into the other. We describe a family of finite state automata that can be used to recognize for an arbitrary input word W the set of all words V where the Levenshtein-distance between V and W does not exceed a certain bound k. The automata can be computed in time linear in the length of W. Given an input word W and an electronic dictionary that is implemented in the form of a trie or a finite state automaton, Levenshtein-automata for W can be used to control search in the lexicon in such a way that exactly the lexical words V are generated where the Levenshtein-distance between V and W does not exceed the given bound. The construction leads to efficient methods for correcting corrupted input words of unrestricted text using very large electronic dictionaries.

Keywords: Spelling correction, Levenshtein-distance, optical character recognition, very large electronic dictionaries.

1 Introduction and Motivation

Text editing systems and systems for optical character recognition have to offer adequate possibilities for correcting garbled words. Due to its relevance for fundamental applications, the problem of how to correct misspelled strings of a text has been considered by many authors (e.g., [Bla60, RE71, Ull77, AFW83, SHC83, Sri85, TIAY90, Kuk92, ZD95, DHH$^+$97]). While some more recent work tries to use the sentence or document context for correcting errors and resolving ambiguities (e.g., [Hul92, KEW91, Hon95]), most contributions suggest methods for correcting isolated words of a text using statistical or lexical information.

Since the correction power of statistical methods is very limited, recent work has concentrated on lexical correction techniques. If an electronic dictionary is available that covers – by and large – the possible input words, a simple procedure may be used for detecting and correcting errors. Given an input word W, it is first checked if the word is in the dictionary. In the negative case, the words of the dictionary that are most similar to W are good suggestions for correction.

Similarity between two words can be measured in several ways. Most popular are (dis)similarity measures based on n-gram distances [AFW83, Ukk92, KST92, KST94] or on variants of the Levenshtein-distance [Lev66, WF74, WBR95, SKS96, OL97]. In this paper, we take the Levenshtein-distance as a basis.

The standard algorithm for computing the Levenshtein-distance between two words by Wagner and Fisher [WF74] uses a dynamic programming scheme that leads to quadratic time complexity. Even with more sophisticated algorithms (cf. [Ukk85, Mye99]) it is not realistic to compute the Levenshtein-distance between the input word W and each of the words in the dictionary, already for dictionaries of a modest size. Even if there are linear algorithms for computing n-gram distances [Ukk92], the situation is basically the same for these metrics.

Several solutions have been proposed to overcome this problem. In most approaches, the correction is divided in two phases. In a first step, a filtering technique is used to select a small number of dictionary words that is guaranteed to contain all interesting corrections of the input string. The emphasis is rather on efficiency than on optimal precision, which means that in general the "list of candidates" also contains words that do not represent reasonable corrections. In the second step, for each candidate the distance to the garbled input word is computed, using a fine-graded measure. Candidates are ranked according to this distance and the best candidates are suggested as correction words. Similarity keys [Sin90, Kuk92, dBdBT95, ZD95] offer one way to select a list of correction candidates in the first step. The dictionary is partitioned into several classes, using appropriate features of the words. The corresponding features of the input word W are used to determine a small number of classes that contain all interesting correction candidates. In other approaches, a special index structure is added to the dictionary. The index may be used, for example, to retrieve the set of all dictionary words that contain a given n-gramm of the input word W. All words where the number of common n-grams with W exceed a certain threshold are selected in the first step [OM88, KST92, ZD95].

More recently, techniques from automata theory have been used to tackle the problem. Bunke [Bun93] has shown that for any given word W the columns of the table computed in the Fisher-Wagner algorithm can be compiled into a deterministic finite state automaton. For any word V the automaton may be used to compute the Levenshtein-distance between V and W in time linear in the length $|V|$ of V. Given a dictionary of words $W_1, \ldots, W_n$, a similar automaton can be given that computes the Levenshtein-distance between V and each of the words W_i in time $O(|V|)$. The problem with the approach is that the size of the automaton is exponential in the sum of the length of the words in the dictionary. Hence the method can only be used for very small dictionaries.

Oflazer [Ofl96] suggested another method that can easily deal with very large lexicons, in fact even with infinite dictionaries of agglutinating languages. The set of all dictionary words is treated as a regular language over the alphabet of letters. As a prerequisite, a deterministic finite state automaton recognizing this language has to be given.[1] Faced with an input word W, Oflazer starts an exhaustive traversal of the dictionary automaton. At each step, the prefix of all letters that are consumed on the path from the initial state to the current state is maintained. A variant of the Wagner-Fisher algorithm is used to control the walk through the automaton in such a way that only prefixes are generated that potentially lead to a correction candidate V where the Levenshtein-distance between V and W does not exceed a fixed bound k. Each dictionary word V within the given distance to W is added to the output list. Oflazer shows that for $k = 1, 2, 3$ the control mechanism helps to avoid the inspection of most of the states of the dictionary automaton. The method leads to a very efficient generation of an appropriate list of correction candidates, even for very large — or infinite — dictionaries.

[1] For finite dictionaries, an efficient algorithm for computing the minimal deterministic finite state automaton for the dictionary has been described in [Mih98, DMWW00].

The correction procedure that we suggest in this paper can be considered as a variant of Oflazer's approach. We also assume that the dictionary is represented as a deterministic finite state automaton. However, we completely avoid the computation of the Levenshtein-distance during the traversal of the automaton. Given the input word W and a bound k, we first compute a finite state automaton A that accepts exactly all words V where the Levenshtein-distance between V and W does not exceed k. A is called a Levenshtein-automaton for W. Levenshtein-automaton and dictionary automaton are then traversed in parallel. In this way, each move in the dictionary automaton is controlled by the Levenshtein-automaton (and vice versa) instead of actually computing any distance. We obtain the intersection of the languages of the two automata as our list of correction candidates.

Our main mathematical results show that appropriate Levenshtein-automata for arbitrary input words can be generated in a uniform and efficient way. We first show that for bounds $k = 1, 2$ the minimal deterministic Levenshtein-automaton of degree k for input W can be computed in linear time and space in $|W|$. For larger bounds, a similar construction is possible, but it might lead to an inacceptable size of the automaton. This discussion is part of future work.

We also introduce a class of non-deterministic Levenshtein-automata that can be computed in linear time and space in $|W|$, for arbitrary bound k. In the non-deterministic case, the structure of the automaton only depends on the length of the input W, hence it is possible to compute for each length a "template automaton" where transitions are labeled with variables. For a given input, we just have to instantiate the appropriate template to receive the Levenshtein-automaton.

Our preliminary experimental results show that lexical string correction with Levenshtein-automata is very fast even for large dictionaries. We used a Bulgarian dictionary with 869200 word entries for correction of garbled words. Given an input word W we computed a deterministic Levensthein automaton for W to retrieve the set of all words V from the lexicon where the Levenshtein-distance between V and W does not exceed bound 1 (tests for larger bounds are in preparation). The average correction time for generating all corrections was below one millisecond (details s.b.), including the time for computing the Levensthein automaton for W and for writing the output. We expect that the method leads to efficient correction even when using dictionaries with several millions of entries. Such dictionaries are needed, for example,

- for correcting OCR-output in a multi-lingual application context, or
- for correcting german texts with composite nouns.

Another possible application area of Levenshtein-automata is approximative string search. Deterministic Levenshtein-automata for a given input word W can be used to efficiently search large files for approximative matches of W.

The paper is structured as follows. Section 2 gives some general technical background. In Section 3 we formally define Levensthein-automata and we describe the string correction method sketched above in more detail. Section 4 describes the construction of the minimal deterministic Levenshtein-automaton of an input word W for distance bounds $k = 1, 2$. Section 5 treats non-deterministic Levenshtein-automata for arbitrary bounds k. Section 6 explains in more detail how to use pre-computed templates for Levenshtein-automata. In Section 7 preliminary experimental results are added. We finish with a short conclusion where we mention some side results of our work and comment on future work.

2 Formal Preliminaries

In the first section, we recall some simple definitions and facts that will be needed later.

Background from formal language theory

We assume that the reader is familiar with the basic notions of formal language theory as described, e.g., in [HU79, Koz97]. As usual, finite state automata (FSA) are treated as tuples of the form $A = \langle \Sigma, Q, q_0, F, \Delta \rangle$ where Σ is the input alphabet, Q is the set of states, $q_0 \in Q$ is the initial state, F is the set of final states, and $\Delta \subseteq Q \times \Sigma^\varepsilon \times Q$ is the transition relation. Here "ε" denotes the empty word and $\Sigma^\varepsilon := \Sigma \cup \{\varepsilon\}$. If $P \subseteq Q$ is a set of states of the FSA A, the ε-closure of P is the set $P \cup \{q \in Q \mid \exists p \in P \text{ s.th. } \langle p, \varepsilon, q \rangle \in \Delta^*\}$. Here $\Delta^* \subseteq Q \times \Sigma^* \times Q$ denotes the generalized transition relation.

The length of a word W is denoted $|W|$. Regular languages over Σ are defined as usual. If A is a FSA, $\mathcal{L}(A)$ denotes the language accepted by A. The concatenation of two languages $\mathcal{L}_1$ and $\mathcal{L}_2$ over Σ is denoted $\mathcal{L}_1 \circ \mathcal{L}_2$. Two words V and W are called *isomorphic* iff V can be obtained from W by a permutation of the alphabet Σ. The notion of isomorphism carries over to automata in the obvious sense.

A finite state automaton A is *deterministic* if the transition relation is a function $\delta : Q \times \Sigma \to Q$. It is wellknown that for each regular language $\mathcal{L}$ there exists a deterministic FSA accepting $\mathcal{L}$ whose number of states is minimal among all deterministic FSA accepting $\mathcal{L}$. The automaton is unique up to isomorphism and called the minimal deterministic FSA for $\mathcal{L}$.

Using some additional notions, a simple characterization of the minimal deterministic FSA can be given. Let $\delta^* : Q \times \Sigma^* \to Q$ denote the generalized transition function, which is defined as usual. A state $q \in Q$ is *accessible* iff there exists a word $U \in \Sigma^*$ such that $\delta^*(q_0, U) = q$. For $q \in Q$ let $\mathcal{L}(q) := \{U \in \Sigma^* \mid \delta^*(q, U) \in F\}$ denote the language of all words that lead from q to a final state. Two states $p, q \in Q$ are called *equivalent* iff $\mathcal{L}(p) = \mathcal{L}(q)$.

Lemma 2.1 *A deterministic FSA $A = \langle \Sigma, Q, q_0, F, \delta \rangle$ is the minimal deterministic FSA for the regular language $\mathcal{L}$ iff $\mathcal{L} = \mathcal{L}(A)$, each state is accessible in A, and if distinct states of A are always non-equivalent.*

The Levenshtein-distance between two words

The Levenshtein-distance between two words is based on the notion of a primitive edit operation. In this paper we mostly consider the standard Levenshtein-distance. Here the primitive operations are

- the subsitution of a symbol by another symbol,
- the deletion of a symbol, and
- the insertion of a symbol.

Obviously, given two words W and V over the alphabet Σ, it is always possible to rewrite W into V, using primitive edit operations.

Definition 2.2 Let V, W be words over the alphabet Σ. The *Levenshein-distance* between V and W is the minimal number of edit operations (substitutions, deletions, or insertions) that are needed to transform V into W.

With $d_L(V,W)$ we denote the Levenshtein-distance between V and W. It can be computed using the following simple dynamic programming scheme (cf. [WF74]):

$$d_L(\varepsilon, W) = |W|$$

$$d_L(V, \varepsilon) = |V|$$

$$d_L(aV, bW) = \begin{cases} d_L(V,W) & \text{if } a = b \\ 1 + \min(d_L(V,W), d_L(aV,W), d_L(V,bW)) & \text{if } a \neq b \end{cases}$$

for $V, W \in \Sigma^*$ and $a, b \in \Sigma$. The following simple observation follows immediately.

Lemma 2.3 *Let $W = UW'$ and $V = UV'$. Then $d_L(V,W) = d_L(V',W')$.*

Definition 2.4 Let W be a word over the alphabet Σ. With $\mathcal{L}_{Lev}(k, W)$ we denote the set of all words V such that $d_L(W,V) \leq k$.

3 String correction with Levenshtein-automata

As indicated in the introduction, we face a situation where we use an eletronic dictionary for detecting and correcting misspelled words. Given any input word W, it is first checked if W is a word of the lexicon. In the negative case, the lexicon is used to generate a list of candidate corrections. The words V of the lexicon that are most similar to W are considered to be good correction candidates. Dissimilarity is measured in terms of the Levenshtein-distance between W and V.

We assume that the dictionary is implemented in the form of a deterministic FSA or a trie. The trie can be considered as a finite state automaton as well. The language of the automaton represents the set of all correct words. We assume that the automaton has the form $A_D = \langle \Sigma, Q^D, q_0^D, F^D, \delta^D \rangle$. A_D will be called the *dictionary automaton* in the sequel.

We may now introduce the central concepts of this paper.

Definition 3.1 Let W be a word over the alphabet Σ, let $k \geq 1$ be a natural number. A finite state automaton A is a *Levenshtein-automaton of degree k for W* iff $\mathcal{L}(A) = \{V \in \Sigma^* \mid d_L(V,W) \leq k\}$.

The correction method suggested in this paper uses Levenshtein-automata. In order to generate a list of correction candidates for a garbled word W, we select a number k and compute a Levenshtein-automaton $A_W = \langle \Sigma, Q^W, q_0^W, F^W, \delta^W \rangle$ of degree k for W. Using a standard backtracking procedure, we traverse the two automata A_W and A_D in parallel. For traversal, we give two algorithms. The first algorithm refers to the case where A (as A_D) is deterministic:

```
push (<ε, q0^D, q0^W>);
while not empty(stack) do begin
```

```
    pop (<V, q^D, q^W>);
    for a in Σ do begin
        q_1^D := δ^D(q^D, a);
        q_1^W := δ^W(q^W, a);
        V_1 := concat(V, a);
        if (q_1^D <> NIL) and (q_1^W <> NIL) then begin
            push(<V_1, q_1^D, q_1^D>);
            if (q_1^D ∈ F^D) and (q_1^W ∈ F^W) then output(V_1);
        end;
    end;
end;
```

Starting with the pair of initial states $\langle q_0^D, q_0^W \rangle$ and the empty word ε, each step of the traversal adds a new letter $a \in \Sigma$ to the actual word and leads from a pair of states $\langle q^D, q^W \rangle \in Q^D \times Q^W$ to $\langle \delta^D(q^D, a), \delta^W(q^W, a) \rangle$. We proceed as long as both components are distinct from the empty failure state[2].

If the Levenshtein automaton is a non-deterministic automaton we can proceed as follows:

```
push (<ε, q_0^D, ε-closure({q_0^W})>);
while not empty(stack) do begin
    pop (<V, q^D, Q^W>);
    for a in Σ do begin
        q_1^D := δ^D(q^D, a);
        Q_1^W := ε-closure(∪_{q^W ∈ Q^W} δ^W(q^W, a));
        V_1 := concat(V, a);
        if (q_1^D <> NIL) and (Q_1^W ≠ ∅) then begin
            push(<V_1, q_1^D, Q_1^W>);
            if (q_1^D ∈ F^D) and (Q_1^W ∩ F^W ≠ ∅) then output(V_1);
        end;
    end;
end;
```

If the Levenshtein-automaton does not have ε-moves the ε-closure is not necessary.

It is trivial that in both situations the list of all output words is $\mathcal{L}(A_D) \cap \mathcal{L}(A_W)$, hence it contains exactly the "grammatical" words in $\mathcal{L}_{Lev}(k, W)$. With a good choice of k, we obtain an appropriate set of correction candidates for the input W.

4 Computation of deterministic Levenshtein-Automata

In this section we show how to compute Levenshtein-automata in order to apply the first correction method described above. The parallel traversal of the dictionary automaton and the Levenshtein-automaton of the input word is simpler and more efficient if the the Levenshtein-automaton is deterministic. For this reason we first look at deterministic Levenshtein-automata. Before we describe appropriate automata of degree 1 and 2 we introduce some general notions. In the sequel, Σ denotes the background alphabet

Definition 4.1 Let $w_1 \cdots w_m \in \Sigma^m$ be a sequence of characters. The *profil* of

[2] A *failure state* is a state q whose language $\mathcal{L}(q)$ is empty.

$w_1 \cdots w_m$ is the sequence of naturals $(n_1 \cdots n_m)$ obtained in the following way. Define $n_1 := 1$. Assume that $n_1, \ldots, n_k$ are defined for some $1 \leq k < m$. If $w_{k+1} \in \{w_1, \ldots, w_k\}$, say, $w_{k+1} = w_i$ (where $1 \leq i \leq k$), then $n_{k+1} := n_i$. In the other case we define $n_{k+1} := \max\{n_i \mid 1 \leq i \leq k\} + 1$.

We write $Pr(W)$ for the profile of W.

Example 4.2 We have

$$\begin{aligned} Pr(aachen) &= (112345) \\ Pr(odd) &= (122) \\ Pr(even) &= (1213) \end{aligned}$$

Let V and W denote two words of the same length. It should be clear that for any degree k we can use isomorphic deterministic Levenshtein-automata whenever $Pr(V) = Pr(W)$. In this section we shall see that the structure of the minimal deterministic Levenshtein-automaton for an input word W depends — in a sense to be made precise — only on local subprofiles of the input word.

Definition 4.3 Let $W = w_1 \cdots w_m$ be a word of length m, let $k \geq 1$. The *k-profile sequence* of W is the sequence of profiles

$$Pr(w_1 \cdots w_k)Pr(w_2 \cdots w_{k+1}) \cdots Pr(w_{m-k+1} \cdots w_m)$$

for $k \leq m$. For $k > m$, the k-profile sequence of W is $Pr(w_1 \cdots w_m)$.

The following example shows that the k-profile sequences of two words can be identical even for non-isomorphic words.

Example 4.4 The 3-profile sequence of *butter* is $(1,2,3), (1,2,2), (1,1,2), (1,2,3)$. The 3-profile sequence of *setter* is the same sequence.

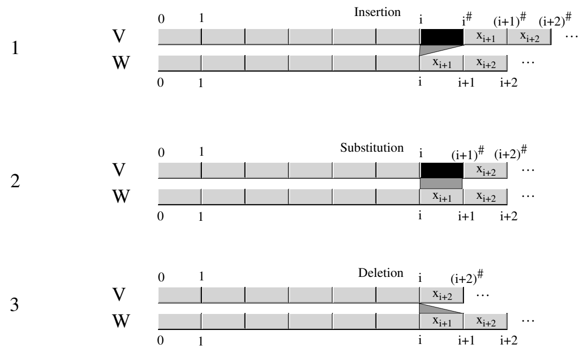
Lemma 4.5 *Let V and W be two words of length m, let $1 \leq k_1 \leq k_2$. If the k_2-profile sequences of V and W are identical, then also the k_1-profile sequences of V and W are identical.*

4.1 The minimal deterministic Levenshtein-automaton of degree 1

We now describe a deterministic Levenshtein-automaton $LEV_1(W)$ of degree 1 for an arbitrary input wird W. We shall prove that $LEV_1(W)$ is the minimal deterministic automaton for the language $\mathcal{L}_{Lev}(1, W)$, modulo deletion of inaccessible states. The generic description of $LEV_1(W)$ can be used to generate, given the input word W, the minimal deterministic finite state automaton for $\mathcal{L}_{Lev}(1, W)$ in linear time and space in $|W|$. We shall see that the structure of $LEV_1(W)$ is completely determined by the 3-profile sequence of W.

Positions

Before we introduce the automaton we consider the three possible ways how a word V in Levenshtein-distance 1 from W can deviate. Suppose that the first i letters of V and W coincide and that letter $y_{i+1} \neq x_{i+1}$ where x_{i+1} and y_{i+1} respectively denote the next letters of W and V. Assume, to avoid a subcase analysis, that $i \leq n-2$ and that x_{i+1} is distinct from the following letter x_{i+2} of W. If V is obtained from W by an insertion (cf. situation 1 below), then the $(i+2)$-th letter of V is x_{i+1}, which is the letter following position i in W. We use a symbol $i^\sharp$ to denote the position of V reached after reading y_{i+1}. The expression $i^\sharp$ will be called a *failure position*, or briefly a *position*. The exponent "$\sharp$" indicates that one editing operation has occurred. The base number i gives the parallel position in W. In a similar way, the positions of V are denoted when we have a subsitution or a deletion.



Elementary transitions

When actually scanning an unknown word V in Levenshtein-distance 1 from W, after reading the first letter where we find a deviation from W — which is, say, the $(i+1)$-th letter y of V — we cannot say which edit operation is used to transform W into V. Hence there may be several possibilities for the actual position. We introduce a state that collects all possible alternatives. The step where we read a letter of V and assign to the current position of V a set of possible successor positions will be called an *elementary transition*. Figure 1 gives a complete survey of all elementary transitions. Depending on the position in V and the profil of W several cases have to be distinguished. Bold lines represent "default transitions" which are always used for symbols that are distinct from the other transitions labels. For convenience, we also introduce the empty set as a target set. Note that the given set of elementary transitions specifies for any position $\pi \in \{i \mid 0 \leq i \leq n\} \cup \{i^\sharp \mid 0 \leq i \leq n\}$ and any symbol $y \in \Sigma$ a unique set of positions, denoted as $\delta'(\pi, y)$.

The automaton

We may now describe the automaton $LEV_1(W)$. Recall that Σ denotes the general background alphabet.

Definition 4.6 Let $W = x_1 \cdots x_n$ where $n \geq 1$. Then $LEV_1(W)$ is the deterministic finite state automaton $\langle \Sigma, Q, q_0, F, \delta \rangle$ where

1. the set of *states* Q contains $\emptyset$, furthermore all sets of the form

$$\begin{array}{ll} \{i\} & (0 \leq i \leq n), \\ \{i^\sharp\} & (0 \leq i \leq n), \end{array}$$

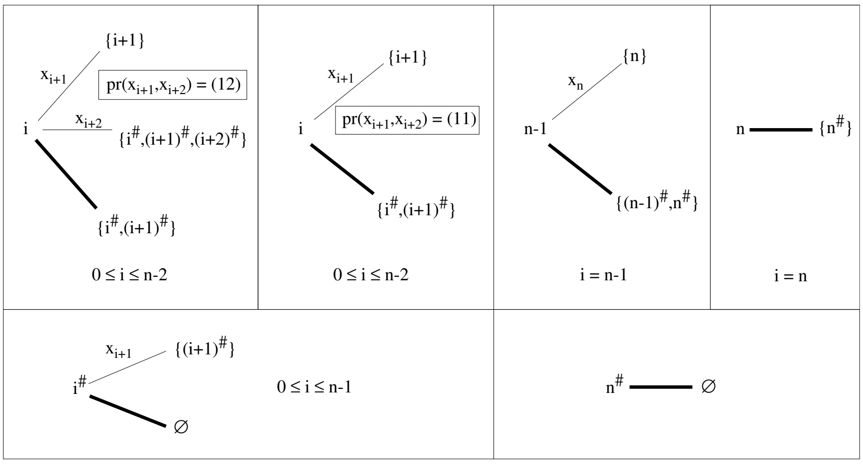


Figure 1: Elementary transitions for input word $W = x_1 \cdots x_n$ (degree 1).

$$\begin{array}{rl} \{i^\sharp, (i+1)^\sharp\} & \quad (0 \leq i \leq n-1), \\ \{i^\sharp, (i+2)^\sharp\} & \quad (0 \leq i \leq n-2), \\ \{i^\sharp, (i+1)^\sharp, (i+2)^\sharp\} & \quad (0 \leq i \leq n-2), \end{array}$$

2. the *initial state* is $q_0 := \{0\}$,
3. the set F of *final states* contains $\{n\}$, $\{n-1\}$, and all states M such that $n^\sharp \in M$,
4. the *transition function* δ is defined in the following way: for any symbol $y \in \Sigma$ and any state M, we define $\delta(M, y) := \bigcup_{\pi \in M} \delta'(\pi, y)$.

It is straightforward to verify that the transition function is well-defined in the sense that $\delta(M, y) \in Q$ for all $M \in Q$ and $y \in \Sigma$. It follows that $LEV_1(W)$ is a deterministic finite state automaton. In the sequel, let $LEV_1(W) = \langle \Sigma, Q, q_0, F, \delta \rangle$ as above.

Proposition 4.7 *The following properties hold:*

1. $\mathcal{L}(\emptyset) = \emptyset$,
2. *for all* $0 \leq i \leq n$*:* $\mathcal{L}(\{i^\sharp\}) = \mathcal{L}_{Lev}(0, x_{i+1} \cdots x_n) = \{x_{i+1} \cdots x_n\}$,
3. *for all states of the form* $M \cup N \subseteq \{0^\sharp, \ldots n^\sharp\}$ *and all* $y \in \Sigma$*:*
$$\delta(M \cup N, y) = \delta(M, y) \cup \delta(N, y),$$
4. *for all states of the form* $M \cup N \subseteq \{0^\sharp, \ldots n^\sharp\}$ *and all* $V \in \Sigma^*$*:*
$$\delta^*(M \cup N, V) = \delta^*(M, V) \cup \delta^*(N, V),$$
5. *for all states of the form* $M \subseteq \{0^\sharp, \ldots n^\sharp\}$*:*
$$\mathcal{L}(M) = \bigcup_{i^\sharp \in M} \mathcal{L}(\{i^\sharp\}) = \{x_{i+1} \cdots x_n \mid i^\sharp \in M\},$$

6. for all $0 \le i \le n$*:* $\mathcal{L}(\{i\}) = \mathcal{L}_{Lev}(1, x_{i+1} \cdots x_n)$.

Proof. Part 1 is trivial. Part 2 is an immediate consequence from the definition of elementary transitions from positions of the form $i^\sharp$ and of the definition of the set of final states F.

Proof of Part 3. We have

$$\begin{aligned}
\delta(M \cup N, y) &= \bigcup_{i^\sharp \in M \cup N} \delta'(i^\sharp, y) \\
&= \bigcup_{i^\sharp \in M} \delta'(i^\sharp, y) \cup \bigcup_{i^\sharp \in N} \delta'(i^\sharp, y) \\
&= \delta(M, y) \cup \delta(N, y).
\end{aligned}$$

Proof of Part 4. Follows from Part 3 by a trivial induction on the length of V.

Proof of Part 5. Using Part 4 and the definition of the set of final states we obtain

$$\begin{aligned}
V \in \mathcal{L}(M) &\Leftrightarrow \delta^*(M, V) \in F \\
&\Leftrightarrow \bigcup_{i^\sharp \in M} \delta^*(\{i^\sharp\}, V) \in F \\
&\Leftrightarrow n^\sharp \in \bigcup_{i^\sharp \in M} \delta^*(\{i^\sharp\}, V) \\
&\Leftrightarrow \exists i^\sharp \in M \colon \delta^*(\{i^\sharp\}, V) \in F \\
&\Leftrightarrow V \in \bigcup_{i^\sharp \in M} \mathcal{L}(\{i^\sharp\}).
\end{aligned}$$

The second equation of Part 5 follows using Part 2.

Proof of Part 6. Let W_i denote the suffix $x_{i+1} \cdots x_n$ of W $(0 \le i \le n)$. We first show that $\mathcal{L}_{Lev}(1, W_i) \subseteq \mathcal{L}(\{i\})$. Let $V \in \mathcal{L}_{Lev}(1, W_i)$. For $V = W_i$ we have $\delta^*(\{i\}, V) = \{n\}$, hence $V \in \mathcal{L}(\{i\})$. For $i < n$, assume that V is obtained from W_i by deleting the last symbol x_n. We have $\delta^*(\{i\}, V) = \{n-1\} \in F$, hence $V \in \mathcal{L}(\{i\})$. In the remaining case we may assume that V is obtained from W_i by an insertion, a substitution, or by a deletion of a symbol $y \ne x_n$.

If V is obtained from W_i by inserting, say, symbol y after x_j (where $i \le j \le n$), we may assume that $y \ne x_{j+1}$. Starting from state $\{i\}$ and consuming the letters $x_{i+1}, \ldots x_j, y$ we reach states $\{i+1\}, \ldots \{j\}, M$ where M contains $j^\sharp$. Parts 2 and 5 show that $\{x_{j+1} \cdots x_n\} \in \mathcal{L}(M)$. Hence $V = x_{i+1} \cdots x_j y x_{j+1} \cdots x_n \in \mathcal{L}(\{i\})$.

Next assume that V is obtained from W_i by substituting a symbol y for x_j (where $i < j \le n$). Starting from state $\{i\}$ and consuming the letters $x_{i+1}, \ldots x_{j-1}, y$ we reach states $\{i+1\}, \ldots \{j-1\}, M$ where M contains $j^\sharp$. Parts 2 and 5 show that $\{x_{j+1} \cdots x_n\} \in \mathcal{L}(M)$. Hence $V = x_{i+1} \cdots x_{j-1} y x_{j+1} \cdots x_n \in \mathcal{L}(\{i\})$.

If V is obtained from W_i by deletion of a symbol x_j (where $j \ge i+1$), we may assume that $j < n$ and $x_j \ne x_{j+1}$. Hence $Pr(x_j, x_{j+1}) = (1, 2)$. Starting from state $\{i\}$ and consuming the letters $x_{i+1}, \ldots x_{j-1}, x_{j+1}$ we reach states $\{i+1\}, \ldots \{j-1\}, M$ where M contains $(j+1)^\sharp$. Parts 2 and 5 show that $\{x_{j+2} \cdots x_n\} \in \mathcal{L}(M)$. Hence $V = x_{i+1} \cdots x_{j-1} x_{j+1} \cdots x_n \in \mathcal{L}(\{i\})$.

Conversely, assume that V is accepted in $LEV_1(W_i)$. If V is accepted (starting from state $\{i\}$) on a sequence $\{i+1\}, \ldots, \{k\}$, then $k \in \{n, n-1\}$ and either

$V = x_{i+1} \cdots x_n = W_i$ or $V = x_{i+1} \cdots x_{n-1}$. The Levenshtein-distance between V and W_i is either 0 or 1. Assume now that V is accepted (starting from $\{i\}$) on a path $\{i+1\}, \ldots, \{j\}, M, \ldots$ where $M \neq \{j+1\}$. Then V has a prefix $x_{i+1} \cdots x_j y$ where $j \leq n$ and $y \neq x_{i+1}$ if $i < n$. The definition of the elementary transitions shows that M can only have the form

$$\begin{array}{rl} \{j^\sharp\} & \\ \{j^\sharp, (j+1)^\sharp\} & \quad \text{(for } j \leq n-1\text{, or)} \\ \{j^\sharp, (j+1)^\sharp, (j+2)^\sharp\} & \quad \text{(for } j \leq n-2,\ y = x_{j+2} \text{ and } Pr(x_{j+1}x_{j+1}) = (1,2)). \end{array}$$

From Part 5 we obtain

$$\begin{array}{rcl} L(\{j^\sharp\}) & = & \{x_{j+1} \cdots x_n\} \\ L(\{j^\sharp, (j+1)^\sharp\}) & = & \{x_{j+1} \cdots x_n, x_{j+2} \cdots x_n\}, \\ L(\{j^\sharp, (j+1)^\sharp, (j+2)^\sharp\}) & = & \{x_{j+1} \cdots x_n, x_{j+2} \cdots x_n, x_{j+3} \cdots x_n\}. \end{array}$$

It follows that

$$\begin{array}{rcl} V & = & x_{i+1} \cdots x_j y x_{j+1} \cdots x_n, \\ V & = & x_{i+1} \cdots x_j y x_{j+2} \cdots x_n, \text{ or} \\ V & = & x_{i+1} \cdots x_j x_{j+2} x_{j+3} \cdots x_n. \end{array}$$

Hence V is obtained from W_i by one insertion, substitution or deletion. □

Theorem 4.8 *Modulo deletion of inaccessible states, $LEV_1(W)$ is the minimal deterministic Levenshtein-automaton of degree 1 for W.*

Proof. Part 6 of Proposition 4.7 shows that $\mathcal{L}(LEV_1(W)) = \mathcal{L}(\{0\}) = \mathcal{L}_{Lev}(1, W)$, hence $LEV_1(W)$ is a deterministic Levenshtein-automaton of degree 1 for W. It follows immediately from Parts 2 and 6 of Proposition 4.7 that distinct states of $LEV_1(W)$ are non-equivalent. By Lemma 2.1, $LEV_1(W)$ is the minimal deterministic Levenshtein automaton of degree 1 for W, modulo deletion of inaccessible states. □

Tables 1 and 2 give a direct description of all transitions of $LEV_1(W)$, omitting transitions to the failure state "$\emptyset$". Obviously we may use the tables to generate $LEV_1(W)$ in time $O(|W|)$. Since the set of accessible states of a deterministic finite state automaton can be computed in time linear in the size of the automaton we receive the following corollary.

Theorem 4.9 *For any word W, the minimal deterministic finite state automaton for the language $\mathcal{L}_{Lev}(1, W)$ can be generated in time and space $O(|W|)$.*

Remark 4.10 The description of the transition function of $LEV_1(W)$ in Tables 1 and 2 only refers to subprofiles of W of type $Pr(w_i w_{i+1} w_{i+2})$, $Pr(w_i w_{i+1})$, and $Pr(w_i w_{i+2})$. Hence it is easy to see that the structure of $LEV_1(W)$ is completely determined by the 3-profile sequence of W. Cf. Lemma 4.5.

Remark 4.11 Lookin at the details of the transition function it becomes obvious that depending on the 3-profile sequence of W some types of states cannot be reached from the initial state and are not needed. For example, if the 3-profile sequence of W has only sequences of the form $(1, 2, 3)$, then states of type $\{i^\sharp, (i+2)^\sharp\}$ cannot be reached.

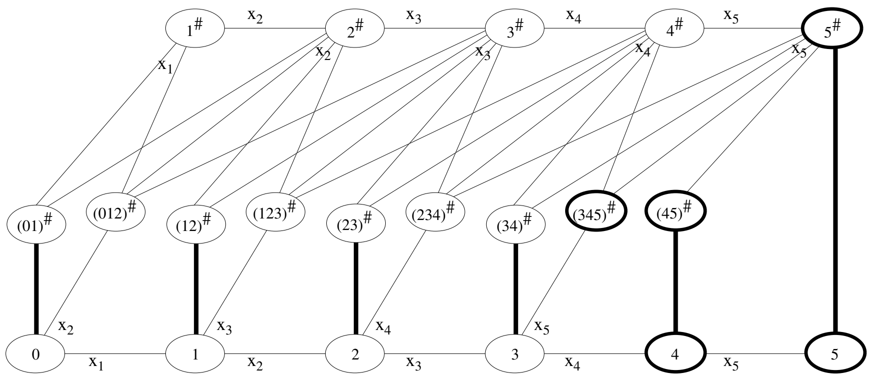


Figure 2: Deterministic Levenshtein-automaton $LEV_1(W)$ for words W of length 5 with 3-profile sequence $(1, 2, 3), (1, 2, 3), (1, 2, 3)$.

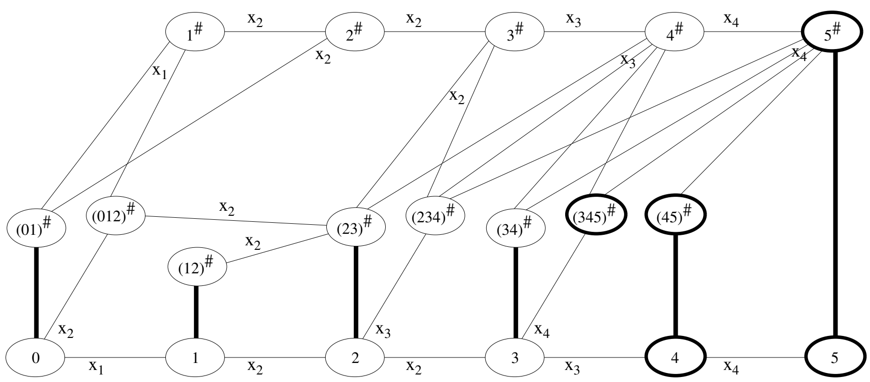


Figure 3: Deterministic Levenshtein-automaton $LEV_1(W)$ for words W of length 5 with 3-profile sequence $(1, 2, 2), (1, 1, 2), (1, 2, 3)$.

$0 \leq i \leq n-3$	
$\{i\}$	for $Pr(x_{i+1}, x_{i+2}) = (1,2)$: $x_{i+1} \mapsto \{i+1\}$ $x_{i+2} \mapsto \{i^\sharp, (i+1)^\sharp, (i+2)\sharp\}$ $df \mapsto \{i^\sharp, (i+1)^\sharp\}$ for $Pr(x_{i+1}, x_{i+2}) = (1,1)$: $x_{i+1} \mapsto \{i+1\}$ $df \mapsto \{i^\sharp, (i+1)^\sharp\}$
$\{i^\sharp\}$	$x_{i+1} \mapsto \{(i+1)^\sharp\}$
$\{i^\sharp, (i+1)^\sharp\}$	for $Pr(x_{i+1}, x_{i+2}) = (1,2)$: $x_{i+1} \mapsto \{(i+1)^\sharp\}$ $x_{i+2} \mapsto \{(i+2)^\sharp\}$ for $Pr(x_{i+1}, x_{i+2}) = (1,1)$: $x_{i+1} \mapsto \{(i+1)^\sharp, (i+2)^\sharp\}$
$\{i^\sharp, (i+2)^\sharp\}$	for $Pr(x_{i+1}, x_{i+3}) = (1,2)$: $x_{i+1} \mapsto \{(i+1)^\sharp\}$ $x_{i+3} \mapsto \{(i+3)^\sharp\}$ for $Pr(x_{i+1}, x_{i+3}) = (1,1)$: $x_{i+1} \mapsto \{(i+1)^\sharp, (i+3)^\sharp\}$
$\{i^\sharp, (i+1)^\sharp, (i+2)^\sharp\}$	for $Pr(x_{i+1}, x_{i+2}, x_{i+3}) = (1,2,3)$: $x_{i+1} \mapsto \{(i+1)^\sharp\}$ $x_{i+2} \mapsto \{(i+2)^\sharp\}$ $x_{i+3} \mapsto \{(i+3)^\sharp\}$ for $Pr(x_{i+1}, x_{i+2}, x_{i+3}) = (1,1,2)$: $x_{i+1} \mapsto \{(i+1)^\sharp, (i+2)^\sharp\}$ $x_{i+3} \mapsto \{(i+3)^\sharp\}$ for $Pr(x_{i+1}, x_{i+2}, x_{i+3}) = (1,2,2)$: $x_{i+1} \mapsto \{(i+1)^\sharp\}$ $x_{i+2} \mapsto \{(i+2)^\sharp, (i+3)^\sharp\}$ for $Pr(x_{i+1}, x_{i+2}, x_{i+3}) = (1,2,1)$: $x_{i+1} \mapsto \{(i+1)^\sharp, (i+3)^\sharp\}$ $x_{i+2} \mapsto \{(i+2)^\sharp\}$ for $Pr(x_{i+1}, x_{i+2}, x_{i+3}) = (1,1,1)$: $x_{i+1} \mapsto \{(i+1)^\sharp, (i+2)^\sharp, (i+3)^\sharp\}$

Table 1: Transitions of $LEV_1(W)$ for $0 \leq i \leq n-3$.

In Figure 2 we give the automaton $LEV_1(W)$ for each word of the form $x_1x_2x_3x_4x_5$ with 3-profile sequence $(1,2,3),(1,2,3),(1,2,3)$. Similarly Figure 3 describes the structure of $LEV_1(W)$ for each word of the form $x_1x_2x_2x_3x_4$ with 3-profile sequence $(1,2,2),(1,1,2),(1,2,3)$.

4.2 The minimal deterministic Levenshtein-automaton of degree 2

The construction of the previous subsection can be generalized. We now describe a deterministic Levenshtein-automaton $LEV_2(W)$ of degree 2 for an arbitrary input word W. Again we prove that $LEV_2(W)$ is the minimal deterministic automaton for the language $\mathcal{L}_{Lev}(2,W)$, modulo deletion of inaccessible states. The generic description of $LEV_2(W)$ can be used to generate, given the input word W, the

$i = n-2$	
$\{i\}$	as above
$\{i^\sharp\}$	as above
$\{i^\sharp, (i+1)^\sharp\}$	as above
$\{i^\sharp, (i+2)^\sharp\}$	$x_{i+1} \mapsto \{(i+1)^\sharp\}$
$\{i^\sharp, (i+1)^\sharp, (i+2)^\sharp\}$	for $Pr(x_{i+1}, x_{i+2}) = (1,2)$: $x_{i+1} \mapsto \{(i+1)^\sharp\}$ $x_{i+2} \mapsto \{(i+2)^\sharp\}$ for $Pr(x_{i+1}, x_{i+2}) = (1,1)$: $x_{i+1} \mapsto \{(i+1)^\sharp, (i+2)^\sharp\}$
$i = n-1$	
$\{i\}$	$x_{i+1} \mapsto \{i+1\}$ $df \mapsto \{i^\sharp, (i+1)^\sharp\}$
$\{i^\sharp\}$	$x_{i+1} \mapsto \{(i+1)^\sharp\}$
$\{i^\sharp, (i+1)^\sharp\}$	$x_{i+1} \mapsto \{(i+1)^\sharp\}$
$i = n$	
$\{i\}$	$df \mapsto \{i^\sharp\}$

Table 2: Transitions of $LEV_1(W)$ for $n-2 \le i \le n$.

minimal deterministic finite state automaton for $\mathcal{L}_{Lev}(2, W)$ in linear time and space in $|W|$. The structure of $LEV_2(W)$ is completely determined by the 5-profile sequence of W.

Positions

Let $W = x_1 \cdots x_n$. As before, we use positions of the form i and $i^\sharp$ $(0 \le i \le n)$. In addition we introduce positions of the form $i^{\sharp\sharp}$ $(0 \le i \le n)$. The base number i gives the actual parallel position of W, the marker "$\sharp\sharp$" indicates that two edit operations have been used.

A remark is in order before we describe the elementary transitions and the automaton. As before, states are formalized as sets of positions. The transitions will be organized such a way that the following properties hold (cf. Proposition 4.7):

$$\begin{aligned}
\mathcal{L}(\{i\}) &= \mathcal{L}_{Lev}(2, w_{i+1} \cdots w_n) \qquad (0 \le i \le n),\\
\mathcal{L}(\{i^\sharp\}) &= \mathcal{L}_{Lev}(1, w_{i+1} \cdots w_n) \qquad (0 \le i \le n),\\
\mathcal{L}(\{i^{\sharp\sharp}\}) &= \mathcal{L}_{Lev}(0, w_{i+1} \cdots w_n) \qquad (0 \le i \le n),
\end{aligned}$$

In general we have $\mathcal{L}(M) = \bigcup_{\pi \in M} \mathcal{L}(\{\pi\})$. This leads to the following

Subsumption Principle: In any state M,

(S-1) each position of the form i subsumes all positions of the form $(i-1)^\sharp, i^\sharp$, as well as $(i+1)^\sharp, (i-2)^{\sharp\sharp}, (i-1)^{\sharp\sharp}, \ldots, (i+2)^{\sharp\sharp}$,

(S-2) each position of the form $i^\sharp$ subsumes all positions of the form $(i-1)^{\sharp\sharp}, i^{\sharp\sharp}, (i+1)^{\sharp\sharp}$.

Obviously we may erase all positions of M that are subsumed by another position of M. This explains the choice of states below. At various places we shall consider

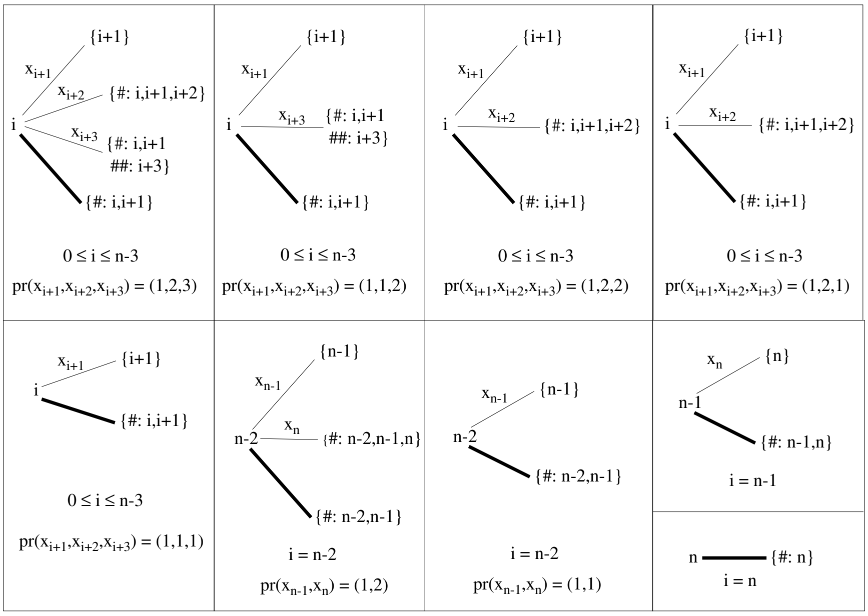


Figure 4: Elementary transitions from positions $0, \ldots, n$ for input word $W = x_1 \cdots x_n$.

the union of two states M and N. We write $M \sqcup N$ for the set that is obtained from $M \cup N$ by omission of states that are subsumed by other states in the above sense.

Elementary transitions

Figure 4 describes all elementary transitions from positions $0, \ldots, n$ for input $W = x_1 \cdots x_n$. The elementary transitions from positions $0^\sharp, \ldots, n^\sharp$ and $0^{\sharp\sharp}, \ldots, n^{\sharp\sharp}$ can be obtained from the elementary transitions for $LEV_1(W)$ by "raising": positions of the form i and $i^\sharp$ are respectively replaced by positions $i^\sharp$ and $i^{\sharp\sharp}$. As before, the elementary transitions define a mapping δ' that assigns to each pair $\langle \pi, y \rangle$ — where π is a position and $y \in \Sigma$ — a set of positions.

The automaton

Definition 4.12 Let $W = x_1 \cdots x_n$ where $n \geq 1$. Then $LEV_2(W)$ is the deterministic finite state automaton $\langle \Sigma, Q, q_0, F, \delta \rangle$ where

1. the set of *states* Q contains $\emptyset$, furthermore all sets of the form

$$
\begin{array}{rl}
\{i\} & (0 \leq i \leq n) \\
\{i^\sharp\} & (0 \leq i \leq n) \\
\{i^\sharp, (i+2)^{\sharp\sharp}\} & (0 \leq i \leq n-2) \\
\{i^\sharp, (i+3)^{\sharp\sharp}\} & (0 \leq i \leq n-3) \\
\{i^\sharp, (i-2)^{\sharp\sharp}\} & (2 \leq i \leq n) \\
\{i^\sharp, (i-3)^{\sharp\sharp}\} & (3 \leq i \leq n)
\end{array}
$$

$$
\begin{array}{rl}
\{i^\sharp, (i+2)^{\sharp\sharp}, (i+3)^{\sharp\sharp}\} & (0 \le i \le n-3) \\
\{i^\sharp, (i-2)^{\sharp\sharp}, (i-3)^{\sharp\sharp}\} & (3 \le i \le n) \\
\{i^\sharp, (i-2)^{\sharp\sharp}, (i+2)^{\sharp\sharp}\} & (2 \le i \le n-2) \\
\{i^\sharp, (i+1)^\sharp\} & (0 \le i \le n-1) \\
\{i^\sharp, (i+1)^\sharp, (i+3)^{\sharp\sharp}\} & (0 \le i \le n-3) \\
\{i^\sharp, (i+1)^\sharp, (i-2)^{\sharp\sharp}\} & (2 \le i \le n-1) \\
\{i^\sharp, (i+2)^\sharp\} & (0 \le i \le n-2) \\
\{i^\sharp, (i+1)^\sharp, (i+2)^\sharp\} & (0 \le i \le n-2) \\
\{i^{\sharp\sharp}\} & (0 \le i \le n) \\
\{i^{\sharp\sharp}, (i+1)^{\sharp\sharp}\} & (0 \le i \le n-1) \\
\{i^{\sharp\sharp}, (i+2)^{\sharp\sharp}\} & (0 \le i \le n-2) \\
\{i^{\sharp\sharp}, (i+1)^{\sharp\sharp}, (i+2)^{\sharp\sharp}\} & (0 \le i \le n-2) \\
\{i^{\sharp\sharp}, (i+3)^{\sharp\sharp}\} & (0 \le i \le n-3) \\
\{i^{\sharp\sharp}, (i+1)^{\sharp\sharp}, (i+3)^{\sharp\sharp}\} & (0 \le i \le n-3) \\
\{i^{\sharp\sharp}, (i+2)^{\sharp\sharp}, (i+3)^{\sharp\sharp}\} & (0 \le i \le n-3) \\
\{i^{\sharp\sharp}, (i+1)^{\sharp\sharp}, (i+2)^{\sharp\sharp}, (i+3)^{\sharp\sharp}\} & (0 \le i \le n-3) \\
\{i^{\sharp\sharp}, (i+4)^{\sharp\sharp}\} & (0 \le i \le n-4) \\
\{i^{\sharp\sharp}, (i+1)^{\sharp\sharp}, (i+4)^{\sharp\sharp}\} & (0 \le i \le n-4) \\
\{i^{\sharp\sharp}, (i+2)^{\sharp\sharp}, (i+4)^{\sharp\sharp}\} & (0 \le i \le n-4) \\
\{i^{\sharp\sharp}, (i+3)^{\sharp\sharp}, (i+4)^{\sharp\sharp}\} & (0 \le i \le n-4) \\
\{i^{\sharp\sharp}, (i+1)^{\sharp\sharp}, (i+2)^{\sharp\sharp}, (i+4)^{\sharp\sharp}\} & (0 \le i \le n-4) \\
\{i^{\sharp\sharp}, (i+2)^{\sharp\sharp}, (i+3)^{\sharp\sharp}, (i+4)^{\sharp\sharp}\} & (0 \le i \le n-4) \\
\{i^{\sharp\sharp}, (i+1)^{\sharp\sharp}, (i+3)^{\sharp\sharp}, (i+4)^{\sharp\sharp}\} & (0 \le i \le n-4) \\
\{i^{\sharp\sharp}, (i+1)^{\sharp\sharp}, (i+2)^{\sharp\sharp}, (i+3)^{\sharp\sharp}, (i+4)^{\sharp\sharp}\} & (0 \le i \le n-4)
\end{array}
$$

2. the *initial state* is $q_0 := \{0\}$,
3. the set F of *final states* contains $\{n\}$, $\{n-1\}$, $\{n-2\}$, and all states M that contain a position $f \in \{n^{\sharp\sharp}, n^\sharp, (n-1)^\sharp\}$,
4. the *transition function* δ is defined in the following way: for any symbol $y \in \Sigma$ and any state M, $\delta(M, y) := \bigsqcup_{\pi \in M} \delta'(\pi, y)$.

It is not difficult to see that due to the erasure of subsumed positions in fact δ assigns to each state $M \in Q$ and each $y \in \Sigma$ a state $M \in Q$. It follows easily that $LEV_2(W)$ is a deterministic finite state automaton. In the sequel, let $LEV_2(W) = \langle \Sigma, Q, q_0, F, \delta \rangle$ as above.

Proposition 4.13 *The following properties hold:*

1. $\mathcal{L}(\emptyset) = \emptyset$,
2. *for all* $0 \le i \le n$:

$$\mathcal{L}(\{i^{\sharp\sharp}\}) = \mathcal{L}_{Lev}(0, x_{i+1} \cdots x_n) \text{ and } \mathcal{L}(\{i^\sharp\}) = \mathcal{L}_{Lev}(1, x_{i+1} \cdots x_n),$$

3. for all states of the form $M \sqcup N \subseteq Q \setminus \{0, \ldots, n\}$ and all $y \in \Sigma$:

$$\delta(M \sqcup N, y) = \delta(M, y) \sqcup \delta(N, y),$$

4. for all states of the form $M \sqcup N \subseteq Q \setminus \{0, \ldots, n\}$ and all $V \in \Sigma^*$:

$$\delta^*(M \sqcup N, V) = \delta^*(M, V) \sqcup \delta^*(N, V),$$

5. for all states $M \subseteq Q \setminus \{0, \ldots, n\}$: $\mathcal{L}(M) = \bigcup_{\pi \in M} \mathcal{L}(\{\pi\})$,

6. for all $0 \leq i \leq n$: $\mathcal{L}(\{i\}) = \mathcal{L}_{Lev}(2, x_{i+1} \cdots x_n)$.

Proof. Part 1 is trivial. Part 2 follows from Parts 2 and 6 of Proposition 4.7 and from the fact that the underlying elementary transitions are obtained from the corresponding elementary transitions of $LEV_1(W)$ by raising of positions.

Proof of Part 3. We have

$$\begin{aligned} \delta(M \sqcup N, y) &= \bigsqcup_{\pi \in M \sqcup N} \delta'(\pi, y) \\ &= \bigsqcup_{\pi \in M} \delta'(\pi, y) \sqcup \bigsqcup_{\pi \in N} \delta'(\pi, y) \\ &= \delta(M, y) \sqcup \delta(N, y). \end{aligned}$$

Proof of Part 4. Follows from Part 3 by a trivial induction on the length of V.

Proof of Part 5. Using Part 4 we obtain

$$\begin{aligned} V \in \mathcal{L}(M) &\Leftrightarrow \delta^*(M, V) \in F \\ &\Leftrightarrow \bigsqcup_{\pi \in M} \delta^*(\{\pi\}, V) \in F \\ &\Leftrightarrow \exists f \in \{n^{\sharp\sharp}, n^{\sharp}, (n-1)^{\sharp}\}\colon f \in \bigsqcup_{\pi \in M} \delta^*(\{\pi\}, V) \\ &\overset{*}{\Leftrightarrow} \exists f \in \{n^{\sharp\sharp}, n^{\sharp}, (n-1)^{\sharp}\}\colon f \in \bigcup_{\pi \in M} \delta^*(\{\pi\}, V) \\ &\Leftrightarrow \exists f \in \{n^{\sharp\sharp}, n^{\sharp}, (n-1)^{\sharp}\}, \exists \pi \in M\colon f \in \delta^*(\{\pi\}, V) \\ &\Leftrightarrow \exists \pi \in M\colon \delta^*(\{\pi\}, V) \in F \\ &\Leftrightarrow \exists \pi \in M\colon V \in \mathcal{L}(\{\pi\}) \\ &\Leftrightarrow V \in \bigcup_{\pi \in M} \mathcal{L}(\{\pi\}). \end{aligned}$$

The marked equivalence follows from the fact that by choice of M the position $n^{\sharp\sharp}$ is the only element of $\{n^{\sharp\sharp}, n^{\sharp}, (n-1)^{\sharp}\}$ that can be subsumed by another element of this set. However, $n^{\sharp\sharp}$ is only subsumed by $n^{\sharp}$ and $(n-1)^{\sharp}$.

Proof of Part 6. Let W_i denote the suffix $x_{i+1} \cdots x_n$ of W $(0 \leq i \leq n)$.

We first show that $\mathcal{L}_{Lev}(2, W_i) \subseteq \mathcal{L}(\{i\})$. Let $V \in \mathcal{L}_{Lev}(2, W_i)$. For $V = W_i$ we have $\delta^*(\{i\}, V) = \{n\}$, hence $V \in \mathcal{L}(\{i\})$. For $i < n$, assume that V is obtained from W_i by deleting either the last symbol x_n or the last two symbols $x_{n-1}x_n$. We have $\delta^*(\{i\}, V) = \{n-1\} \in F$ or $\delta^*(\{i\}, V) = \{n-2\} \in F$, hence $V \in \mathcal{L}(\{i\})$. We now look at the remaining cases. Here there exists a position j such that $V = x_{i+1} \cdots x_j y V'$ where $y \neq x_{j+1}$. We have $yV' \in \mathcal{L}_{Lev}(2, x_{j+1} \cdots x_n)$ by Lemma 2.3.

If the occurrence of y is an insertion before x_{j+1} (where $i \leq j \leq n$), then $V' \in \mathcal{L}_{Lev}(1, x_{j+1} \cdots x_n)$. Starting from state $\{i\}$ and consuming the letters $x_{i+1}, \ldots x_j, y$ we reach states $\{i+1\}, \ldots \{j\}, M$ where M contains $j^\sharp$ (cf. elementary transitions). Parts 2 and 5 show that $V' \in \mathcal{L}(M)$. Hence $V \in \mathcal{L}(\{i\})$.

If the occurrence of y substitutes x_{j+1} (where $i \leq j < n$), then V' belongs to $\mathcal{L}_{Lev}(1, x_{j+2} \cdots x_n)$. Starting from state $\{i\}$ and consuming the letters $x_{i+1}, \ldots x_j, y$ we reach states $\{i+1\}, \ldots \{j\}, M$ where M contains $(j+1)^\sharp$. Parts 2 and 5 show that $V' \in \mathcal{L}(M)$. Hence $V \in \mathcal{L}(\{i\})$.

In the remaining cases, symbol x_{j+1} has been deleted and yV' belongs to $\mathcal{L}_{Lev}(1, x_{j+2} \cdots x_n)$. We do not have an additional insertion before x_{j+1} since the two operations could be modelled by one substitution. This shows in particular that we may assume that $j \leq n-2$. We have to distinguish several cases.

Case 1: $y = x_{j+2}$. Here $x_{j+1} \neq x_{j+2}$. Hence $Pr(x_{j+1}, x_{j+2}, x_{j+3}) = (1, 2, \_)$ for $j \leq n-3$ and $Pr(x_{j+1}, x_{j+2}) = (1, 2)$, for $j = n-2$. Now $V' \in \mathcal{L}_{Lev}(1, x_{j+3} \cdots x_n)$, by Lemma 2.3. Starting from state $\{i\}$ and consuming the letters $x_{i+1}, \ldots x_j, x_{j+2}$ we reach states $\{i+1\}, \ldots \{j\}, M$ where M contains $(j+2)^\sharp$. Parts 2 and 5 show that $V' \in \mathcal{L}(M)$. Hence $V \in \mathcal{L}(\{i\})$.

Case 2: $y \neq x_{j+2}$. The occurrence of y cannot be an insertion before x_{j+2}, as we remarked above.

Assume that y is a substitute for x_{j+2}. In this case $V' = x_{j+3} \cdots x_n$. Note that $x_{j+3} \cdots x_n \in \mathcal{L}_{Lev}(1, x_{j+2} \cdots x_n)$. Starting from state $\{i\}$ and consuming the letters $x_{i+1}, \ldots x_j, y$ we reach states $\{i+1\}, \ldots \{j\}, M$ where M contains $(j+1)^\sharp$. Parts 2 and 5 show that $x_{j+3} \cdots x_n \in \mathcal{L}(M)$. Hence $V \in \mathcal{L}(\{i\})$.

The last explanation for y is that $yV' = x_{j+3} \cdots x_n$, which means that x_{j+2} has been erased as well. Recall that $x_{j+3} = y \notin \{x_{j+1}, x_{j+2}\}$. Hence

$$Pr(x_{j+1}, x_{j+2}, x_{j+3}) \in \{(1,1,2), (1,2,3)\}.$$

Starting from state $\{i\}$ and consuming the letters $x_{i+1}, \ldots x_j, y$ we reach states $\{i+1\}, \ldots \{j\}, M$ where M contains $(j+3)^{\sharp\sharp}$. Parts 2 and 5 show that $V' = x_{j+4} \cdots x_n \in \mathcal{L}(M)$. Hence $V \in \mathcal{L}(\{i\})$.

It remains to prove that $\mathcal{L}(\{i\}) \subseteq \mathcal{L}_{Lev}(2, W_i)$. Let $V \in \mathcal{L}(\{i\})$. If V is accepted - starting from $\{i\}$ - on a path $\{i+1\}, \{i+2\}, \ldots, \{k\}$, then $k \in \{n, n-1, n-2\}$, $V \in \{x_{i+1} \cdots x_n, x_{i+1} \cdots x_{n-1}, x_{i+1} \cdots x_{n-2}\}$ and $V \in \mathcal{L}_{Lev}(2, W_i)$. In the other case, starting from state $\{i\}$ and consuming the letters of V we reach states $\{i+1\}, \ldots \{j\}, M$ where $M \notin \{i, \ldots, n\}$. Using Parts 2 and 5 and the definition of elementary transitions it is now routine to check that $V \in \mathcal{L}_{Lev}(2, W_i)$. To treat one example, consider the case $j \leq n-3$, $V = x_{i+1} \cdots x_j x_{j+2} V'$, $Pr(x_{j+1}, x_{j+2}, x_{j+3}) = (1, 2, 3)$. Here $M = \{j^\sharp, (j+1)^\sharp, (j+2)^\sharp\}$. Parts 3 and 5 show

$$\mathcal{L}(M) = \mathcal{L}_{Lev}(1, x_{j+1} \cdots x_n) \cup \mathcal{L}_{Lev}(1, x_{j+2} \cdots x_n) \cup \mathcal{L}_{Lev}(1, x_{j+3} \cdots x_n).$$

It is trivial to check that $V \in \{x_{i+1} \cdots x_j x_{j+2}\} \circ L(M) \subseteq \mathcal{L}_{Lev}(2, x_{i+1} \cdots x_n)$. $\square$

Theorem 4.14 *Modulo deletion of inaccessible states, $LEV_2(W)$ is the minimal deterministic Levenshtein-automaton of degree 2 for W.*

Proof. Part 6 of Proposition 4.13 shows that

$$\mathcal{L}(LEV_2(W)) = \mathcal{L}(\{0\}) = \mathcal{L}_{Lev}(2, W),$$

hence $LEV_2(W)$ is a deterministic Levenshtein-automaton of degree 2 for W. It follows immediately from Parts 2 and 6 of Proposition 4.13 that distinct states of $LEV_2(W)$ are non-equivalent. By Lemma 2.1, $LEV_2(W)$ is the minimal deterministic Levenshtein-automaton of degree 2 for W modulo deletion of inaccessible states. $\square$

Obviously is possible to give a complete table of all transitions of $LEV_2(W)$, similary as for $LEV_1(W)$.

Theorem 4.15 *For any word W, the minimal deterministic finite state automaton for the language $\mathcal{L}_{Lev}(2, W)$ can be generated in time and space $O(|W|)$.*

The structure of $LEV_2(W)$ is completely determined by the 5-profile sequence of W.

5 Computation of weakly non-deterministic Levenshtein-automata

Since the number of states of deterministic Levenshtein-automata of higher degrees seems to be problematic we now introduce a class of non-deterministic Levenshtein-automata where treatment of large distances of simpler. The automata to be described below do not have any ε-moves, and in many cases for each state there exists at most one letter of the alphabet that licenses two distinct transitions. Because of these characteristics the automata will be called weakly non-deterministic. Instead of formally defining weakly non-deterministic Levenshtein-automata for an input word W, we give graphical descriptions. In each case it is trivial to derive from the picture the set of (final) states, the initial state, and the transition relation. It is also trivial to see that the automata can be generated in linear time and space in the length of the input word W. The proofs that the given automata are in fact Levenshtein-automata for the input word W are straightforward and omitted.

Figure 5 gives a weakly non-deterministic Levenshtein-automaton $Lev_1(W)$ of degree 1 for an arbitrary input word W of length 5. The generalization to input words of arbitrary length n are obvious. The automaton for an input word of length n has $3n + 2$ states.

In order to introduce weakly non-deterministic Levenshtein-automata of degree 2 we need the following definition.

Definition 5.1 Two edit operations Op_1, Op_2 on the word $W = w_1 \cdots w_n$ are *neighboured* in the following cases:

1. Op_1 is deletion of w_i and Op_2 is deletion of w_{i+1} (deletion-deletion),
2. Op_1 is deletion of w_i and Op_2 is substitution of w_{i+1} (deletion-substitution),
3. Op_1 is substitution of w_i and Op_2 is deletion of w_{i+1} (substitution-deletion),
4. Op_1 is substitution of w_i and Op_2 is substitution of w_{i+1} (substitution-substitution),
5. Op_1 is an insertion before of w_i and Op_2 is a substitution of w_i, (insertion-substitution),

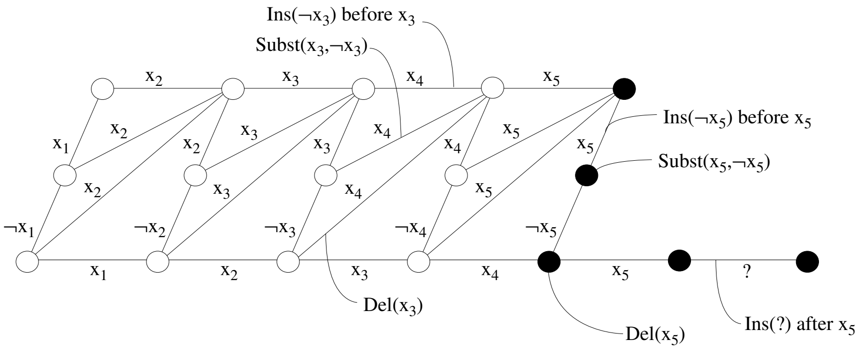


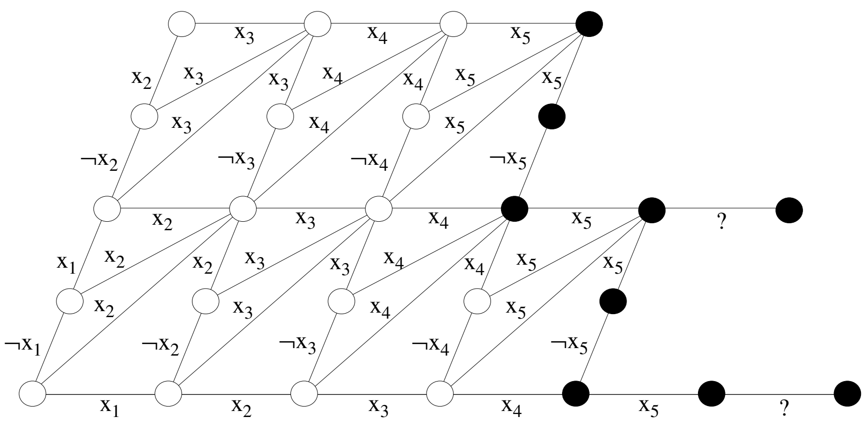
Figure 5: The weakly non-deterministic Levenshtein-automaton $Lev_1(x_1x_2x_3x_4x_5)$.

6. Op_1 is a substitution of w_i and Op_2 is an insertion before w_i, (substitution-insertion),
7. Op_1 is an insertion before of w_i and Op_2 is a second insertion before w_i, or both are insertions after w_n (insertion-insertion),

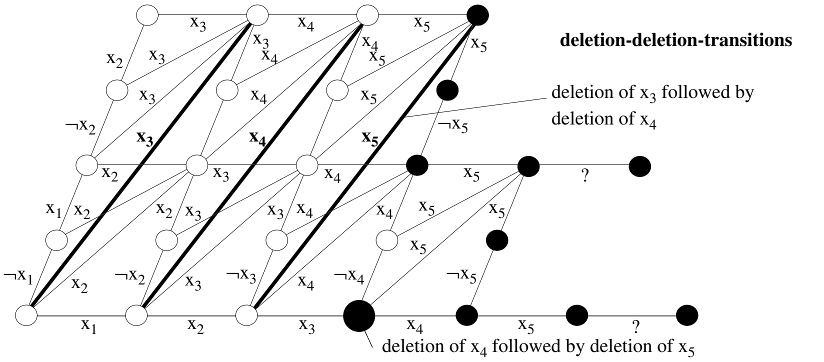
Here $1 \leq i < n$ in cases 1-4 and $1 \leq i \leq n$ in cases 5 and 6.

Remark 5.2 We do not consider insertions before w_i followed by a deletion of w_i since we obtain the same result using a substitution of w_i. For similar reasons we disregard deletion of w_i followed by an insertion before w_{i+1}. If weights are used that depend on the edited symbols it could make sense to extend the above list.

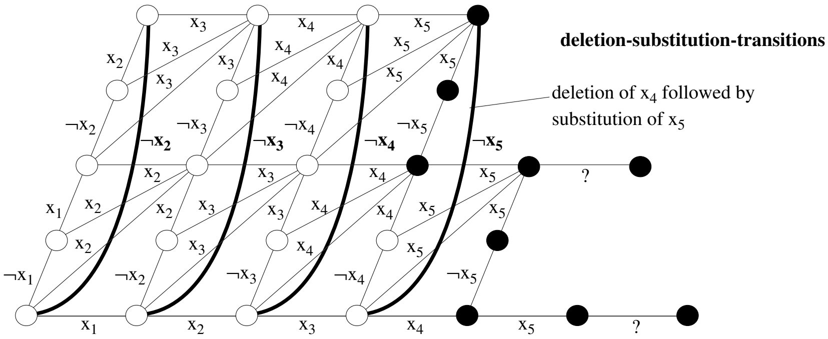
Levenshtein-automata of degree 2 will first be illustrated by ignoring neighboured edit operations. In this case, the automaton for the input word $x_1x_2x_3x_4x_5$ is basically a straightforward combination of two Levenshtein-automata of degree 1:



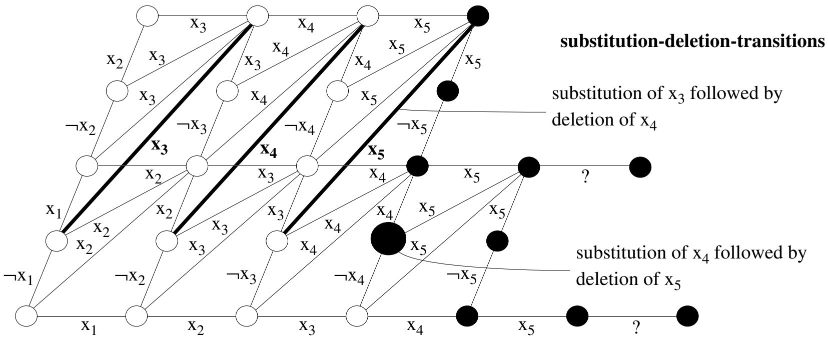
We shall now show how this automaton has to be extended in order to take neighboured edit operations into account. First, we consider the deletion-deletion case. Here we have to add some transition, furthermore an additional state is treated as a final state now:



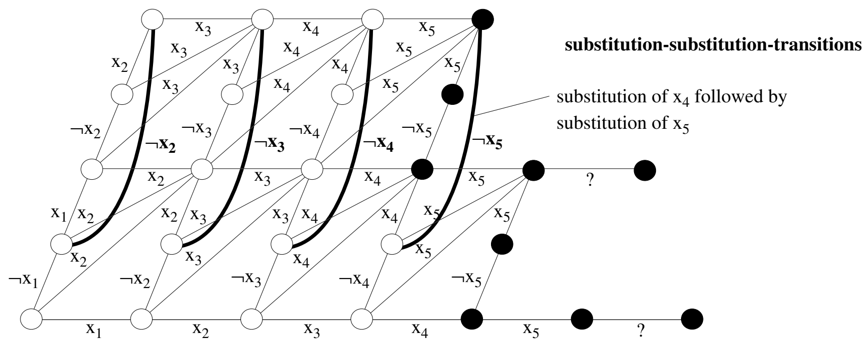
In order to cover deletion-substitution case, four additional transitions are needed:



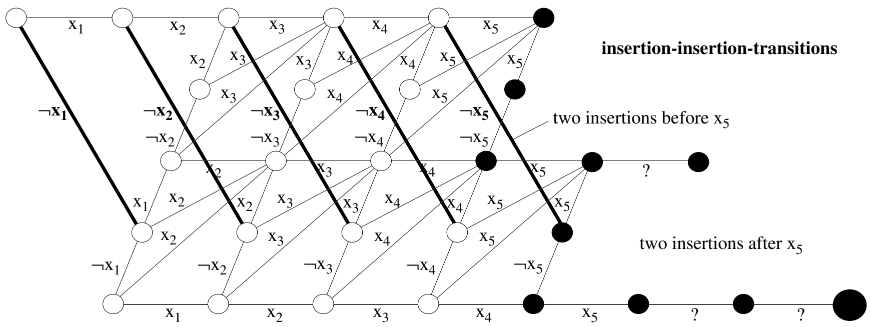
For the substitution-deletion case, three new transitions are needed, an additional state is marked as a final state.



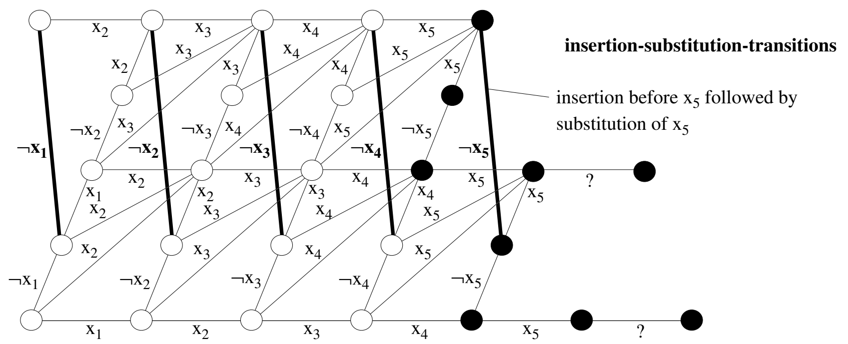
In order to cope with two subsequent substitutions we can use four additional transitions.



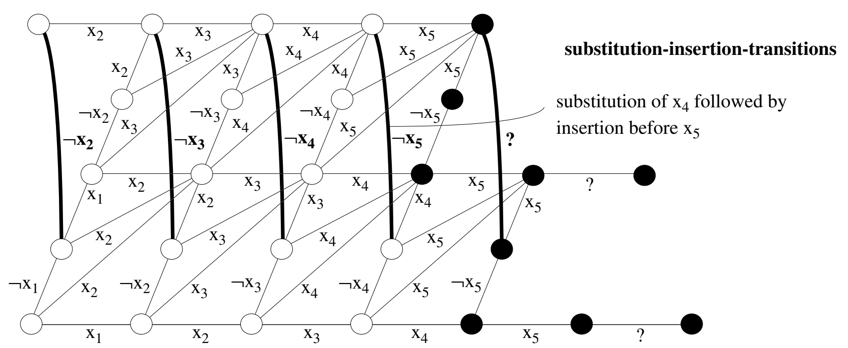
In order to allow for double insertions, three new states have to be added. One state is a final state that represents two insertions after x_5.



Insertion-substitutions can be handled in the following way:



Substitution-insertion sequences follow a similar scheme:



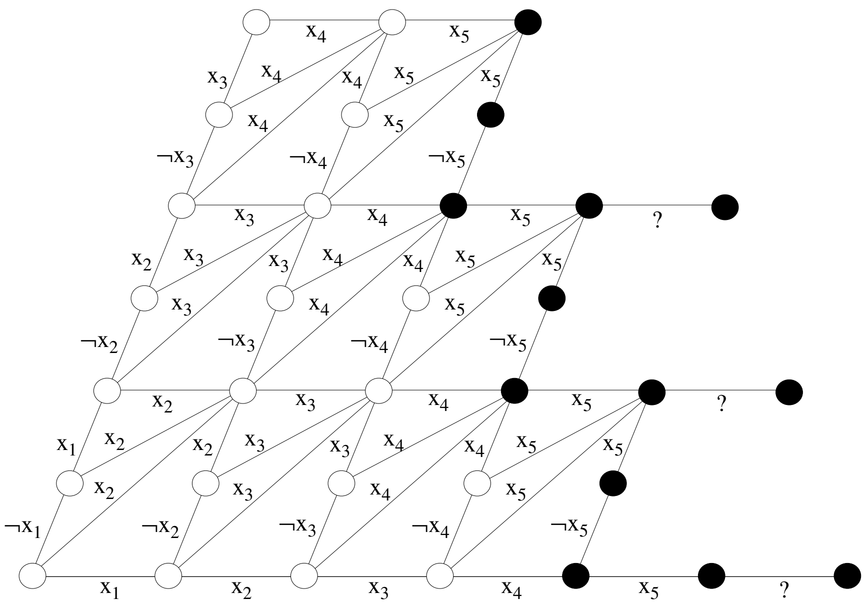


Figure 6: Weakly non-deterministic Levenshtein-automaton of degree 3 without neighboured edit operations.

The construction of Levenshtein-automata of degree $k \geq 3$ follows the same ideas. We may use the Levenshtein-automaton of degree k without neighboured editing as a starting point, and add transitions and (final) states that take neighboured edit operations into account. Note that we may now have k neighboured edit operations. We do not go into details. The weakly non-deterministic Levenshtein-automaton of degree 3 without neighboured edit operations is described in Figure 6.

6 Templates for Levenshtein-automata

From the description of the non-deterministic Levenshtein-automata it should be obvious that their structure only depends on the length of the input word. This shows that it is not necessary to compute the Levenshtein-automaton of an input word W at runtime. We may pre-compute for each length n a Levenshtein-automaton for a "generic" input word $x_1 \cdots x_n$ in an offline computation. Such an automaton will be called a *template automaton* in the sequel. Given a concrete input word (e.g., "aachen") we compute its length, select the template of appropriate length and instantiate the transitions labels, mapping variables to letters (e.g., $x_1 \mapsto a, x_2 \mapsto a \ldots, x_6 \mapsto n$).

When using deterministic Levenshtein-automata, the use of templates is also possible. However, the structure of deterministic Levenshtein-automata depends on the profile sequences, as we have seen before. The second line of the following table gives the number of distinct 3-profile sequences for words of length n (line 1), which characterizes the number of distinct minimal deterministic Levenshtein-automata of degree 1. The third line gives the number of distinct 5 profile sequences, which characterizes the number of distinct minimal deterministic Levenshtein-automata of degree 2.

1	2	3	4	5	6	7	8	9
1	2	5	13	34	89	233	610	1597
1	2	5	15	52	188	696	2606	9800

This shows that for small length, the use of deterministic Levenshtein-templates is possible. For larger length, there are serious limitations. Since many profile sequences will probably never occur in practice, an ultimate judgement of the possibilities for the use of templates (instead of an actual computation of Levenshtein-automata) cannot be given here.

7 Preliminary experimental results

First experimental results were made using a Bulgarian lexicon with 869200 word entries. The algorithm for computing, given an input W, the automaton $LEV_1(W)$ and the algorithm for computing all dictionary words in Levenshtein-distance ≤ 1 to W described in Section 3 were implemented in C on a 500 MHz Pentium III machine under Linux. We did not use template automata. As a first test set we used all the prefixes of length $3, 4, \ldots, 20$ of all dictionary words as input "words" and computed the correction candidates. The results are given in Table 3 and show that the average time needed to correct one input string remains below one millisecond, regardless of the length of the input (column 5).

The third column represents the average time that was needed to construct $LEV_1(W)$. A remarkable point is that the construction time is almost independent from the length of the input. This seems to indicate that most of the time is needed for initialization of the data structures that are used by the algorithm. Possibly this offers some room for further optimizations. Also the average time that is needed for the simultaneous traversal of $LEV_1(W)$ and the dictionary automaton (column 4) does not grow drastically when using long input words. A partial explanation is that the average number of corrections decreases for long input strings (column 6). The times given in columns 4, 5 include the time needed to output corrections.

8 Conclusion

We introduced a new method for correcting garbled words using an electronic dictionary that is represented as a deterministic finite state automaton. Our method can be used for very large electronic dictionaries. It is similar to Oflazer's method [Ofl96], but it avoids the computation of Levenshtein-distances. Instead, a suitable Levenshtein-automaton for the given input word is used to control lexical search. Our main mathematical results show that appropriate Levenshtein-automata can be computed in linear time. The preliminary experimental results show that even for a dictionary with 869200 word entries the new method leads to overall correction times below one millisecond per word on average.

The complexity results for computing the minimal deterministic Levensthein-automaton of degree $1, 2$ for a given input word immediately lead to the following consequences.

Lemma 8.1 *For $k = 1, 2$, given two words W and V of length m and n respectively, it is decidable in time $O(\max(m, n))$ if the Levenshtein-distance between W and V*

word length	number of input words	av. time (ms) automaton construction	av. time traversal (ms)	av. total correction time (ms)	av. number of corrections
3	3152	0.581	0.228	0.809	10.312500
4	12121	0.579	0.258	0.837	8.657207
5	30243	0.585	0.264	0.848	6.972754
6	59835	0.584	0.291	0.875	6.660199
7	101763	0.589	0.286	0.875	6.440425
8	150046	0.588	0.301	0.889	6.343401
9	190318	0.590	0.322	0.911	5.836600
10	203520	0.597	0.320	0.917	5.247470
11	184138	0.595	0.333	0.929	4.653390
12	139982	0.596	0.343	0.939	4.089597
13	91252	0.599	0.344	0.944	3.612644
14	52603	0.600	0.346	0.946	3.238275
15	27997	0.603	0.339	0.942	2.983248
16	14763	0.604	0.335	0.940	2.733184
17	8179	0.608	0.330	0.938	2.615723
18	4601	0.611	0.326	0.937	2.510976
19	2790	0.613	0.326	0.939	2.347670
20	1585	0.606	0.341	0.946	2.312934

Table 3: Test results for Bulgarian lexicon with 869200 word entries

is $\leq k$.

Lemma 8.2 *For $k = 1, 2$, given a text of words of length n and a word W of length m we can compute in time $O(\max(m, n))$ all words V of the text where the Levenshtein-distance between V and W does not exceed k.*

The results obtained in this paper should be extended in several directions. First, the construction of the minimal deterministic Levenshtein-automaton can be generalized from degrees $1, 2$ to arbitrary degree k. The structure of the resulting automaton will be decribed in a forthcoming extension of the present paper. The description may be used to generalized the above two results to arbitrary tolerance bounds k. Note, however, that the bound k is always treated as a constant.

Second, in various applications it makes sense to allow for further primitive edit operation besides substitution, deletion and insertion. It is not difficult modify the structure of (weakly) non-deterministic Levenshtein-automata, taking other edit operations into account. In Figure 7 we illustrate the technique, using splits, merges and transpositions as additional operations. It seems interesting to analyze the structure of the minimal deterministic Levenshtein-automaton for an input word W if non-standard edit operations are used.

Third, the situation should be considered where edit operations come with specific costs that depend on the symbols of the operation. In addition different methods for ranking correction candidates should be discussed that take the occurrence frequency of the words into account.

Fourth, experimental results have to be extended in several directions. Tests are in preparation where we use a German dictionary [Gue96, MM95, Lan96, Sch98]

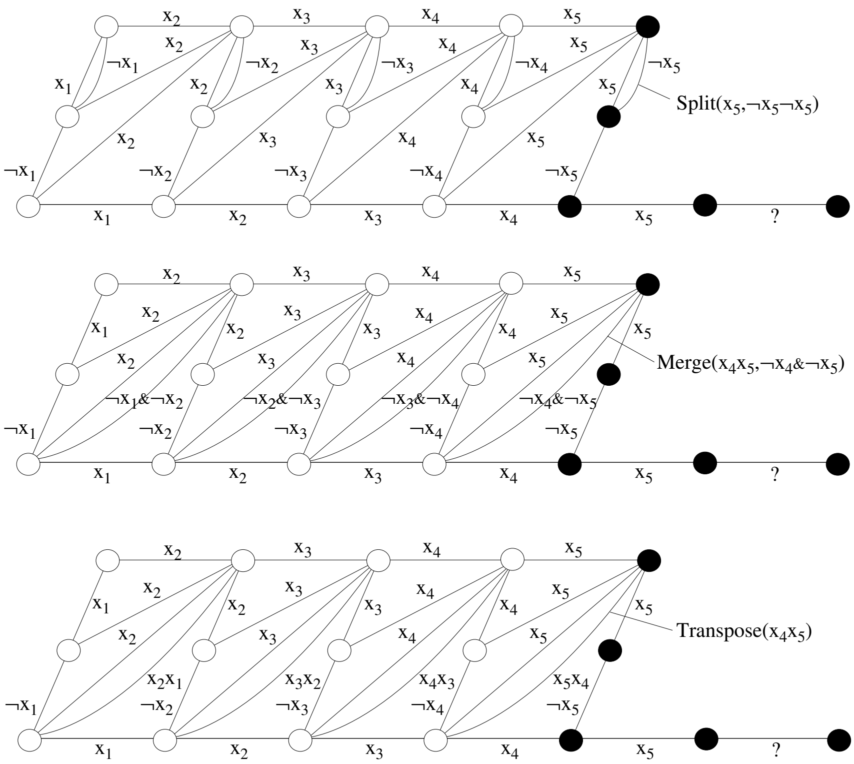


Figure 7: Weakly non-deterministic Levenshtein-automata of degree 1 with splits, merges and transpositions respectively.

with several millions of entries, the majority being composite nouns.

Fifth, the use of Levenshtein-automata for detecting approximate matches in texts should be tested. Here it seems interesting to generalize the construction algorithm for Levenshtein-automata in such a way that a finite number of input words is simultaneously taken into account.

References

[AFW83] R.C. Angell, G.E. Freund, and P. Willett. Automatic spelling correction using a trigram similarity measure. *Information Processing and Management*, 19:255–261, 1983.

[Bla60] C.R. Blair. A program for correcting spelling errors. *Information and Control*, 3:60–67, 1960.

[Bun93] Horst Bunke. A fast algorithm for finding the nearest neighbor of a word in a dictionary, 1993. Aus http://citeseer.nj.nec.com/cs ResearchIndex.

[dBdBT95] Francois de Bertrand de Beuvron and Philippe Trigano. Hierarchically coded lexicon with variants. *International Journal of Pattern Recognition and Artificial Intelligence*, 9(1):145–165, 1995.

[DHH+97] A. Dengel, R. Hoch, F. Hönes, T. Jäger, M. Malburg, and A. Weigel. Techniques for improving OCR results. In H. Bunke and P.S.P. Wang, editors, *Handbook of Character Recognition and Document Image Analysis*. World Scientific, 1997.

[DMWW00] J. Daciuk, S. Mihov, B. Watson, and R. Watson. Incremental construction of minimal acyclic finite state automata. *Computational Linguistics*, 26(1), 2000.

[Gue96] F. Guenthner. Electronic Lexica and Corpora Research at CIS. *International Journal of Corpus Linguistics*, 1(2), 1996.

[Hon95] Tao Hong. *Degraded Text Recognition Using Visual and Linguistic Context*. PhD thesis, CEDAR, State Unicersity of New York at Buffalo, 1995.

[HU79] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, MA, 1979.

[Hul92] J.J. Hull. A hidden markov model for language syntax in text recognition. In *Proc. of the 11th IAPR Int. Conf. on Pattern Recognition*, pages 416–423, The Hague, The Netherlands, 1992. IEEE Computer Society Press.

[KEW91] F.G. Keenan, L.J. Evett, and R.J. Withrow. A large vocabulary stochastic analyser for handwriting recognition. In *Proc. of the First International Conference on Document Analysis and Recognition (ICDAR 91)*, pages 794–802, 1991.

[Koz97] Dexter C. Kozen. *Automata and Computability*. Springer, New York, Berlin, 1997.

[KST92] J.Y. Kim and J. Shawe-Taylor. An approximate string-matching algorithm. *Theoretical Computer Science*, 92:107–117, 1992.

[KST94] J.Y. Kim and J. Shawe-Taylor. Fast string matching using an n-gram algorithm. *Software–Practice and Experience*, 94(1):79–88, 1994.

[Kuk92] Karen Kukich. Techniques for automatically correcting words in texts. *ACM Computing Surveys*, pages 377–439, 1992.

[Lan96] S. Langer. *Selektionsklassen und Hyponymie im Lexikon*. PhD thesis, CIS, Universität München, 1996.

[Lev66] V.I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Sov. Phys. Dokl.*, 1966.

[Mih98] Stoyan Mihov. Direct building of minimal automaton for given list. *Annuaire de l'Université de Sofia "St. Kl. Ohridski"*, 91, livre 1, 1998.

[MM95] P. Maier-Meyer. *Lexikon und automatische Lemmatisierung*. PhD thesis, CIS, Universität München, 1995.

[Mye99] E. Myers. A fast bit-vector algorithm for approximate string matching based on dynamic programming. *Journal of the ACM*, 46(3):395–415, 1999.

[Ofl96] Kemal Oflazer. Error-tolerant finite-state recognition with applications to morphological analysis and spelling correction. *Computational Linguistics*, 22(1):73–89, 1996.

[OL97] B.J. Oommen and R.K.S. Loke. Pattern recognition of strings with substitutions, insertions, deletions, and generalized transpositions. *Pattern Recognition*, 30(5):789–800, 1997.

[OM88] O. Owolabi and D.R. McGregor. Fast approximate string matching. *Software - Practice and Experience*, 18(4):387–393, 1988.

[RE71] E.M. Riseman and R.W. Ehrich. Contextual word recognition using binary digrams. *IEEE Transactions on Computers*, C-20, 1971.

[Sch98] D. Schnorbusch. *Einfache deutsche Verben. Eine syntaktische und semantische Beschreibung der verbalen Simplizia für das elektronische Lexikonsystem CISLEX*. PhD thesis, CIS, Universität München, 1998. Studien zur Informations- und Sprachverarbeitung, Band 1.

[SHC83] Sargur N. Srihari, Jonathan J. Hull, and Ramesh Choudhari. Integrating diverse knowledge sources in text recognition. *ACM Transactions on Office Information Systems*, 1983.

[Sin90] R.M.K. Sinha. On partitioning a dictionary for visual text recognition. *Pattern Recognition*, 23(5):497–500, 1990.

[SKS96] Giovanni Seni, V. Kripasundar, and Rohini K. Srihari. Generalizing edit distance to incorporate domain information: Handwritten text recognition as a case study. *Pattern Recognition*, 29(3), 1996.

[Sri85] S.N. Srihari. *Computer Text Recognition and Error Correction*. Tutorial, IEEE Computer Society Press, Silver Spring, MD, 1985.

[TIAY90] H. Takahashi, N. Itoh, T. Amano, and A. Yamashita. A spelling correction method and its application to an OCR system. *Pattern Recognition*, 23(3/4):363–377, 1990.

[Ukk85] E. Ukkonen. Algorithms for approximate string matching. *Information and Control*, 64:100–118, 1985.

[Ukk92] E. Ukkonen. Approximate string-matching with q-grams and maximal matches. *Theoretical Computer Science*, 92:191–211, 1992.

[Ull77] J.R. Ullmann. A binary n-gram technique for automatic correction of substitution, deletion, insertion and reversal errors. *The Computer Journal*, 20(2):141–147, 1977.

[WBR95] F. Weigel, S. Baumann, and J. Rohrschneider. Lexical postprocessing by heuristic search and automatic determination of the edit costs. In *Proc. of the Third International Conference on Document Analysis and Recognition (ICDAR 95)*, pages 857–860, 1995.

[WF74] R.A. Wagner and M. Fisher. The string-to-string correction problem. *Journal of the ACM*, 1974.

[ZD95] Justin Zobel and Philip Dart. Finding approximate matches in large lexicons. *Software–Practice and Experience*, 25(3):331–345, 1995.