IJDAR (2002) 5: 67–85
Digital Object Identifier (DOI) 10.1007/s10032-002-0082-8

International Journal on Document Analysis and Recognition

Fast string correction with Levenshtein automata

Klaus U. Schulz[1], **Stoyan Mihov**[2]

[1] CIS, University of Munich, Oettingenstr. 67, 80538 Munich, Germany (e-mail: schulz@cis.uni-muenchen.de)
[2] Linguistic Modelling Laboratory, LPDP, Bulgarian Academy of Sciences (e-mail: stoyan@lml.bas.bg)

Received: 13 February 2002 / Accepted: 13 March 2002

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. Levenshtein automata of degree n for a word W are defined as finite state automata that recognize the set of all words V where the Levenshtein distance between V and W does not exceed n. We show how to compute, for any fixed bound n and any input word W, a deterministic Levenshtein automaton of degree n for W in time linear to the length of W. Given an electronic dictionary that is implemented in the form of a trie or a finite state automaton, the Levenshtein automaton 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. This leads to a very fast method for correcting corrupted input words of unrestricted text using large electronic dictionaries. We then introduce a second method that avoids the explicit computation of Levenshtein automata and leads to even improved efficiency. Evaluation results are given that also address variants of both methods that are based on modified Levenshtein distances where further primitive edit operations (transpositions, merges and splits) are used.

Keywords: Spelling correction – Levenshtein distance – Optical character recognition – Electronic dictionaries

1 Introduction and motivation

The problem of how to find good correction candidates for a garbled input word is important for many fundamental applications, including spelling correction, speech recognition, optical character recognition, error-tolerant querying of search engines for the world wide web and other kinds of information systems. Due to its relevance, the problem has been considered by many authors (e.g., [Bla60, RE71, Ull77, AFW83, SHC83, Sri85, TIAY90, Kuk92, ZD95, DHH+97]). Most contributions suggest methods for correcting isolated words of a text.[1] Since purely statistical methods cannot offer sufficient correction accuracy, modern approaches are generally built on top of lexical techniques.

If an electronic dictionary is available that covers the possible input words, a simple procedure may be used for detecting and correcting errors. Given an input word W, it is first checked whether the word is in the dictionary. In the negative case, the words of the dictionary that are most similar to W are suggested as correction candidates. If necessary, appropriate statistical data can be used for refinement of ranking. Similarity between two words can be measured in several ways. Most useful are (dis)similarity measures based on variants of the Levenshtein distance [Lev66, WF74, WBR95, SKS96, OL97] or on n-gram distances [AFW83, Ukk92, KST92, KST94]. 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]), 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. The problem becomes even more serious when using dictionaries for highly inflectional or agglutinating languages (e.g., Russian, German, Turkish, Finnish, Hungarian), dictionaries for languages that allow for composition of nouns (German), multi-lingual dictionaries, or background dictionaries for correcting queries to search engines. In these cases, dictionaries may contain up to several millions of entries. The problem arises of how to compute the lexical Levenshtein neighbours of a garbled input word while

[1] Some more recent work tries to use the sentence or document context for correcting errors and resolving ambiguities, such as [Hul92, KEW91, Hon95].

respecting the efficiency constraints that arise from realistic industrial applications.

Several solutions have been proposed for fast selection of possible corrections. Often the dictionary is offline partitioned using some kind of similarity key (e.g., [Sin90, Kuk92, dBdBT95, ZD95]), or it is enriched with a special index structure ([OM88, KST92, ZD95]). Correction of a given input word is divided in two steps, In a first step, the similarity key or the index is used for coarse search, extracting a list of dictionary words that is guaranteed to contain all interesting corrections of the input string. In a second step (fine search), 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.

Oflazer suggested another method that can deal even with infinite dictionaries of agglutinating languages [Ofl96]. 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.[2] 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 n. Each dictionary word V within the given distance to W is added to the output list. Oflazer shows that for bounds $n = 1, 2, 3$ the control mechanism helps to avoid the inspection of most of the states of the dictionary automaton. The method leads to an efficient generation of an appropriate list of correction candidates, even for very large – or infinite – dictionaries.

The first 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 n, we first compute a deterministic finite state automaton A that accepts exactly all words V where the Levenshtein distance between V and W does not exceed n. 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. We obtain the intersection of the languages of the two automata as our list of correction candidates. Clearly, this intersection is the set of all dictionary words V where the Levenshtein distance between V and W does not exceed n.

Our main algorithmic result shows that for any fixed degree n and input W a deterministic Levenshtein automaton A_W for W can be computed in linear time and space in $|W|$. In order to maximize practical efficiency, the computation of A_W for fixed distance bound n is based on a precompiled table T_n that contains a parametric and generic description of states and transitions of A_W. At run-time, given input W, parametric states and transitions of T_n are instantiated, yielding the automaton A_W. The instantiation of the parametric transition rules of T_n is triggered by Boolean vectors that characterize the distribution of letters of W in subwords of length $2n + 1$. The table-based approach leads to an improved variant of the correction method where the traversal of the dictionary automaton is controlled using the table T_n itself. Moves in A_W are simulated and the actual computation of the Levenshtein automaton A_W is avoided, thus improving efficiency.

The above results always refer to the "standard" Levenshtein distance where the distance between two words W and V is defined as the minimal number of insertions, deletions and substitutions that are needed to transform W into V. For specific applications, variants of this metrics are preferable. In a typesetting context often two symbols are transposed. In the context of optical character recognition, two symbols are often merged into one symbol, or conversely one symbol is split into two symbols. Motivated by these cases, we also developed Levenshtein automata for the modified Levenshtein distance where insertions, deletions, substitutions and transpositions are used as primitive edit operations, and for the variant where insertions, deletions, substitutions, merges and splits are treated as primitive edit operations. In both cases, techniques and results obtained for the standard Levenshtein distance can be lifted.

Our evaluation results show that string correction with (simulated) Levenshtein automata is in fact very fast. For example, using a Bulgarian dictionary with 870 000 entries and (standard) distance bound $n = 1$, the average time to compute and output all lexical Levenshtein neighbours of a garbled input word are around 0.4 ms on a Pentium III. Using a German dictionary of composite nouns with 6 058 198 entries the average time was between 1.3 ms (short words) and 2.5 ms. Further results for modified Levenshtein distances and other distance bounds are given below.

The paper is structured as follows. Section 2 provides some general technical background. In Sect. 3, we formally define Levensthein automata, and we describe the first string correction method sketched above in more detail. Section 4 gives a generic description of a deterministic Levenshtein automaton of arbitrary degree n for arbitrary input word W. In Sect. 5, we show how to use

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

this description to derive tables $T_1, T_2, T_3, \ldots$ that respectively contain parametric descriptions of states and transitions of a deterministic Levenshtein automata of degree $n = 1, 2, 3 \ldots$ for arbitrary input word W. Using these tables it is trivial to generate a deterministic Levenshtein automaton for input W in time linear to the length of W. Section 6 discusses the second correction method, where the actual computation of the Levenshtein automaton for the input word W is avoided. In Sect. 7, we describe evaluation results for distinct notions of Levenshtein distance, distance bounds and dictionaries. We finish with a short conclusion where we mention some side results of our work and comment on related and future work. As to Levenshtein automata for modified Levenshtein distances where transpositions or merges and splits are treated as primitive edit operations, details are described in a technical report [SM01].

2 Formal preliminaries

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 $\Sigma^\varepsilon := \Sigma \cup \{\varepsilon\}$. We write $\mathcal{L}(A)$ for the language accepted by A.

A finite state automaton A is *deterministic* if the transition relation is a function $\delta : Q \times \Sigma \to Q$. Let $A = \langle \Sigma, Q, q_0, F, \delta \rangle$ be a deterministic FSA, let $\delta^* : Q \times \Sigma^* \to Q$ denote the generalized transition function, which is defined as usual. For $q \in Q$ we write $\mathcal{L}(q) := \{U \in \Sigma^* \mid \delta^*(q, U) \in F\}$ for the language of all words that lead from q to a final state.

The length of a word W is denoted $|W|$. Regular languages over Σ are defined as usual. With $\mathcal{L}_1 \circ \mathcal{L}_2$ we denote the concatenation of the languages $\mathcal{L}_1$ and $\mathcal{L}_2$.

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.

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 first consider the standard Levenshtein distance. Here the primitive operations are the *substitution* 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 1 *Let V, W be words over the alphabet Σ. The (standard)* Levenshtein distance *between V and W is the minimal number of edit operations (substitutions, deletions, or insertions) that are needed to transform V into W.*

Table 1 Computation of standard Levenshtein distance

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

for $V, W \in \Sigma^*$ and $a, b \in \Sigma$.

With $d_L(V, W)$ we denote the Levenshtein distance between V and W. It can be computed using the simple dynamic programming scheme given in Table 1 (cf. [WF74]).

The following simple observation follows immediately.

Lemma 1 *Let $W = UW'$ and $V = UV'$. Then $d_L(V, W) = d_L(V', W')$.*

Let $W = x_1 x_2 \cdots x_w$ and $V = y_1 y_2 \cdots y_v$ be two words with Levenshtein distance $n \geq 0$. Consider a sequence ν of edit operations of minimal length leading from W to V. If we substitute a letter x_i by another symbol z, the latter symbol will not be erased or subsituted by one of the following edit operations of ν since otherwise ν would not have minimal length. Hence there exists a unique letter y_j of V that represents the descendant of z in V and the substitution result of x_i. In the *trace representation* (cf. [WF74]) of ν we introduce a stroke from x_i to y_j. Similarly we introduce a stroke from x_i to y_j if x_i is not touched by any edit operation and if y_j represents the descendant of x_i in the new word V. Assume that all strokes of the above form are introduced. Clearly, two strokes never cross. Moreover, each letter x_i of W that does not represent the starting point of a stroke is deleted by some operation of ν, and each letter y_j of V that does not represent the end point of a stroke is an inserted symbol.

Remark 1 Let $W = x_1 x_2 \cdots x_w$ and $V = y_1 y_2 \cdots y_v$ be two words with Levenshtein distance $n \geq 1$. Assume that V is not a prefix of W or vice versa. Let $U = x_1 x_2 \cdots x_i$ (where $0 \leq i \leq v, w$) denote the maximal common prefix of V and W. Then, in any trace representation of a minimal sequence ν of edit operations leading from W to V exactly one of the following three cases holds:

1. *Insertion case.* A stroke is starting at x_{i+1} that points to some y_{i+j} where $j > 1$.
2. *Substitution case.* Letters x_{i+1} and y_{i+1} are connected by a stroke.

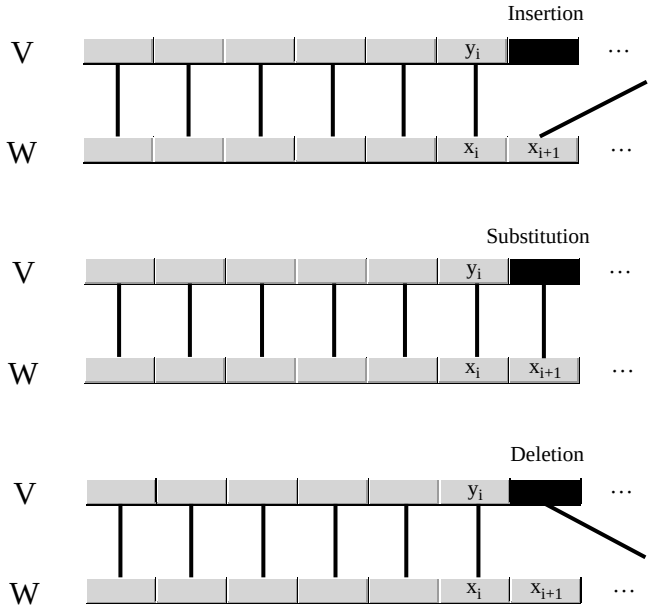

Fig. 1 Possible trace pictures for situation of Remark 1

3. *Deletion case.* A stroke ends at y_{i+1} that starts at some x_{i+j} where $j > i + 1$.

In fact, the only remaining case would be the situation where neither x_{i+1} nor y_{i+1} represent the end point of a stroke. This would mean that x_{i+1} is deleted and y_{i+1} is inserted in ν. Using one substitution instead, we would get a shorter sequence of edit operations from V to W, which gives a contradiction. The three possible cases are indicated in Fig. 1.

3 String correction with Levenshtein automata

As indicated in the introduction, we face a situation where an electronic dictionary is used 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.

In the sequel, Σ denotes the background alphabet. We assume that the dictionary is implemented in the form of a deterministic FSA or a trie. A 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.

Definition 2 *Let W be a word over the alphabet Σ. With $\mathcal{L}_{\mathrm{Lev}}(n, W)$ we denote the set of all words $V \in \Sigma^*$ such that $d_L(W, V) \leq n$.*

```
push (<ε, q_0^D, q_0^W>);
while not empty(stack) do begin
   pop (<V, q^D, q^W>);
   for x in Σ do begin
      q_1^D := δ^D(q^D, x);
      q_1^W := δ^W(q^W, x);
      if (q_1^D <> NIL) and (q_1^W <> NIL) then begin
         V_1 := concat(V, x);
         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;
```

Fig. 2 Backtracking procedure for parallel traversal of dictionary automaton and Levenshtein automaton

We now introduce the central concept of this paper.

Definition 3 *Let W be a word over the alphabet Σ, let $n \in \mathbb{N}$. A finite state automaton A is a* Levenshtein automaton of degree n for W *iff* $\mathcal{L}(A) = \mathcal{L}_{\mathrm{Lev}}(n, W)$.

The first correction method suggested in this paper follows a simple idea. In order to generate a list of correction candidates for a garbled input word W, we select a number n and compute a deterministic Levenshtein automaton $A_W = \langle \Sigma, Q^W, q_0^W, F^W, \delta^W \rangle$ of degree n for W. Using the simple and well-established backtracking procedure described in Fig. 2 we traverse the two automata A_W and A_D in parallel. 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 $x \in \Sigma$ to the actual word V and leads from a pair of states $\langle q^D, q^W \rangle \in Q^D \times Q^W$ to $\langle \delta^D(q^D, x), \delta^W(q^W, x) \rangle$. We proceed as long as both components are distinct from the empty failure state NIL.[3] Whenever in both automata a final state is reached, the actual word is added to the output.

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

The computational costs of the above algorithm is bound by the size of the dictionary automaton A_D and depends on the bound n that is used. If n reaches the length of the longest word in the dictionary, then in general (e.g., for the empty input word) the algorithm gives rise to a complete traversal of A_D. In practical cases, small bounds are used, and only a small portion of A_D will be visited. For bound 0, the algorithm only validates in time $O(|W|)$ if the input word W is in the dictionary.

We shall also introduce a second and related correction method. This method, which avoids the actual computation of Levenshtein automata, can only be de-

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

scribed once we have introduced a number of additional concepts.

4 A family of deterministic Levenshtein automata

In this section, we introduce a deterministic Levenshtein automaton $LEV_n(W)$ of degree n for an input word W. The description is generic in the sense that we neither make any specific assumption on the degree n nor on the length or the form of the input word W. The description will be the basis for efficient computation of Levenshtein automata for fixed degree n, to be described in the following section.

Profile sequences and characteristic vectors. We first introduce some notions that help to characterize the structural properties of the input word W that determine the structure of the automaton $LEV_n(W)$.

Definition 4 *Let $U = z_1 \cdots z_u \in \Sigma^u$ be a sequence of characters. The* profil *$Pr(U)$ of U is the sequence of naturals $(n_1 \cdots n_u)$ obtained in the following way. Define $n_1 := 1$. Assume that $n_1, \ldots, n_k$ are defined for some $1 \leq k < u$. If $z_{k+1} \in \{z_1, \ldots, z_k\}$, say $z_{k+1} = z_i$ (where $1 \leq i \leq k$), then $n_{k+1} := n_i$. In the other case, we define $n_{k+1} := \max\{n_i \mid 1 \leq i \leq k\} + 1$.*

Example 1 We have

$$Pr(aachen) = (112345),$$
$$Pr(odd) = (122),$$
$$Pr(even) = (1213).$$

Let W and W' denote two words of the same length. It should be clear that for any fixed degree n we can use isomorphic deterministic Levenshtein automata for input words W and W' whenever $Pr(W) = Pr(W')$. A stronger relationship can be established. We shall see that the structure of the deterministic Levenshtein automaton for an input word W to be described below depends – in a sense to be made precise – only on *local* subprofiles of the input word.

Definition 5 *Let $U = z_1 \cdots z_u$, let $k \geq 1$. The* k-profile sequence *of W is the sequence of profiles*

$$Pr(z_1 \cdots z_k), Pr(z_2 \cdots z_{k+1}), \ldots, Pr(z_{u-k+1} \cdots z_u)$$

for $k \leq u$. For $k > u$, the k-profile sequence of U is $Pr(z_1 \cdots z_u)$.

The k-profile sequences of two words can be identical even for non-isomorphic words.

Example 2 The 3-profile sequence of the word *butter* is

$$(1,2,3), (1,2,2), (1,1,2), (1,2,3).$$

The 3-profile sequence of *setter* is the same sequence.

The following notion plays a key role when defining the images of the states of Levenshtein automata under input symbols $x \in \Sigma$.

Definition 6 *Let $x \in \Sigma$ and let $V = y_1 \ldots y_v \in \Sigma^*$. The* characteristic vector *of x with respect to V is the bit-vector $\chi(x, V) := \langle b_1, \ldots, b_v \rangle$ where $b_j := 1$ iff $y_j = x$ and $b_j := 0$ otherwise.*

The following remark shows how the information contained in a profile can be modularized using characteristic vectors. The technique will be used when we define the transitions of $LEV_n(W)$.

Remark 2 Given the profile of a word V we can derive all characteristic vectors of the form $\chi(x, V)$, just using the characteristic vectors of numbers $1, 2, \ldots$ with respect to $Pr(V)$. For example, if $Pr(V) = (1,2,1,2,3,1,2)$, then the characteristic vectors $\chi(x, V)$ have the form

$$\langle 1,0,1,0,0,1,0 \rangle$$
$$\langle 0,1,0,1,0,0,1 \rangle$$
$$\langle 0,0,0,0,1,0,0 \rangle$$
$$\langle 0,0,0,0,0,0,0 \rangle$$

(assuming that Σ has at least four letters). Conversely, given the set of all characteristic vectors of the form $\chi(x, V)$, we may obviously derive $Pr(V)$.

Non-deterministic Levenshtein automata. The deterministic Levenshtein automata to be introduced below can be considered as the result of a specialized and optimized power set construction that is applied to non-deterministic Levenshtein automata. The construction of non-deterministic Levenshtein automata for a given input word W and degree n is straightforward and long-established in approximate string matching. As an example, Fig. 3 depicts such an automaton, A, for an arbitrary input word $W = w_1 w_2 w_3 w_4 w_5$ of length 5 and degree $n = 2$. The states of A are expressions of the form $i^{\sharp e}$. The exponent $\sharp e$ indicates the number e of edit operations that have been registered on the way from the initial state $0^{\sharp 0}$ to $i^{\sharp e}$. All transitions are left-to-right and/or bottom-up. Transitions from states $i^{\sharp e}$ to $i^{\sharp e+1}$ represent insertions. In order to indicate that these transitions can be applied consuming any input symbol we use bold lines. A transition from state $i^{\sharp e}$ to $(i+1)^{\sharp e+1}$ labeled with the empty word ϵ represents a deletion of the symbol w_{i+1}. A similar transition with label $\neg w_{i+1}$ represents a substitution of w_{i+1}, hence any symbol distinct from w_{i+1} can be consumed. Final states are $5^{\sharp 0}$, $5^{\sharp 1}$, and $5^{\sharp 2}$. Note that it would not make a difference to use $3^{\sharp 0}$, $4^{\sharp 0}$, and $4^{\sharp 1}$ as additional final states.

The automaton A is uniform in the sense that – modulo the given assignment of transition labels – the graph structure of A is the same for arbitrary input of length 5. In order to understand the role of the following construction, it is important to note that the result of a determinization of A using the usual power set construction

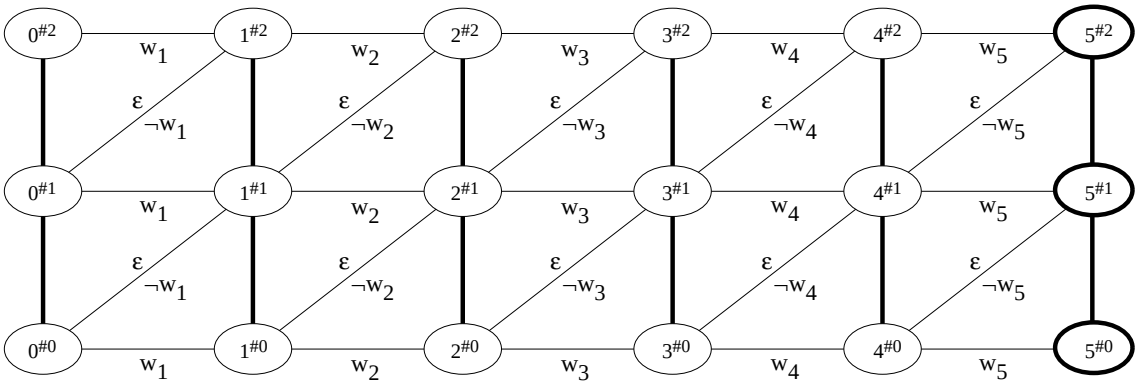

Fig. 3 Non-deterministic Levenshtein automaton of degree 2 for arbitrary input $W = w_1 w_2 w_3 w_4 w_5$ of length 5

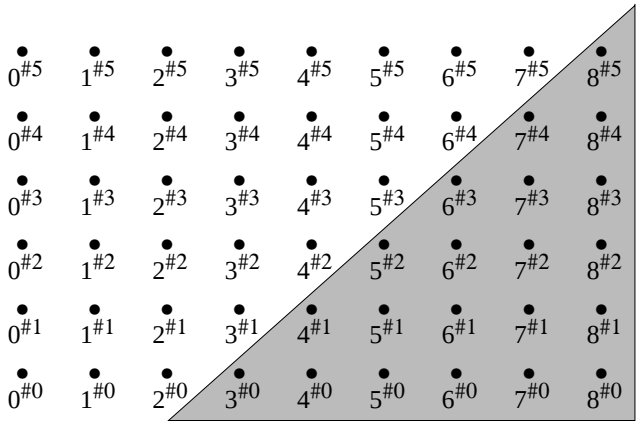

Fig. 4 Positions and accepting positions for $n = 5$ and $w = 8$

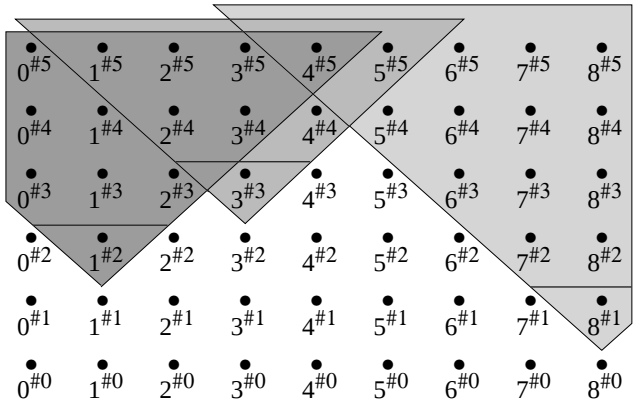

Fig. 5 Subsumption triangles (cf. Example 4)

heavily depends on identities between letters $w_1, \ldots, w_5$. For this reason, a uniform *deterministic* Levenshtein automaton for arbitrary input of length 5 cannot be given. Two examples of deterministic Levenshtein automata for input of length 5 (of degree 1) can be found in Figs. 8 and 9.

Positions and states. In order to characterize the states of our deterministic Levenshtein automata we now fix an arbitrary input word $W = x_1 \cdots x_w$ and a number $n \in \mathbb{N}$ that denotes the maximal Levenshtein distance that we want to capture. In the sequel, numbers $i \in \{0, \ldots, w\}$ will be called the *boundaries* of W. In order to distinguish between states of deterministic Levenshtein automata and states of the kind of non-deterministic Levenshtein automata described above, the latter will be called *positions* henceforth.

Definition 7 *A* position *is an expression of the form* $i^{\sharp e}$ *where* $0 \leq i \leq w$ *and* $0 \leq e \leq n$. *Position* $i^{\sharp e}$ *is* raised *iff* $e > 0$, *otherwise it is called a* base position.

Base positions can be considered as "error-free" positions.

Definition 8 *A position* $i^{\sharp e}$ *is* accepting *iff* $w - i \leq n - e$.

Accepting positions can be considered as final states of the non-deterministic Levenshtein automata described above.

Example 3 For $n = 5$ and $w = 8$, the set of all positions is depicted in Fig. 4. Accepting positions are marked.

Definition 9 *A position* $i^{\sharp e}$ subsumes *a position* $j^{\sharp f}$ *iff* $e < f$ *and* $|j - i| \leq f - e$. *The set of all positions that are subsumed by* $i^{\sharp e}$ *is called the* subsumption triangle *of* $i^{\sharp e}$.

Example 4 Let $n = 5$ and assume that $w = 8$. Figure 5 illustrates the subsumption triangles of $1^{\sharp 2}$, $3^{\sharp 3}$ and $8^{\sharp 1}$. Since subsumption is irreflexive, the positions $1^{\sharp 2}$, $3^{\sharp 3}$ and $8^{\sharp 1}$ do not belong to the respective triangles.

The following lemma indicates the background for the notion of subsumption.

Lemma 2 *Let* $W = x_1 \cdots x_w$ *and* n *as above. Let* Φ *denote the function that assigns to each position* $i^{\sharp e}$ *the language*

$$\Phi(i^{\sharp e}) := \mathcal{L}_{\mathrm{Lev}}(n - e, x_{i+1} \cdots x_w).$$

Let $\pi := i^{\sharp e}$ *and* $\pi' := j^{\sharp f}$ *be two distinct positions. If* π *subsumes* π', *then* $\Phi(\pi')$ *is a subset of* $\Phi(\pi)$.

Proof Assume that $\pi = i^{\sharp e}$ subsumes $\pi' = j^{\sharp f}$. Then $e < f$ and $|j - i| \leq f - e$. Since $x_{j+1} \cdots x_w$ can be obtained from $x_{i+1} \cdots x_w$ by a series of $|j - i|$ insertions (for $j \leq i$) or deletions (for $j > i$) it follows easily that $\Phi(\pi')$ is a subset of $\Phi(\pi)$. □

The states of $LEV_n(W)$ are sets of positions of a particular type.

Definition 10 *Let $0 \leq i \leq w$. A* state with base position $i^{\sharp 0}$ *is a set M of positions, not necessarily containing $i^{\sharp 0}$, that satisfies the following properties:*

1. *For each position $j^{\sharp e}$ in M, we have $|i - j| \leq e$. That is, each position of M, with the possible exception of $i^{\sharp 0}$, lies in the subsumption triangle of $i^{\sharp 0}$.*
2. *M does not contain any position that is subsumed by another element of M.*

Let us note that a state may have several possible base positions.

Example 5 First assume that $n = 1$ and $w = 2$. Then the states are

$$\emptyset, \{0^{\sharp 0}\}, \{1^{\sharp 0}\}, \{2^{\sharp 0}\}, \{0^{\sharp 1}\}, \{1^{\sharp 1}\},$$
$$\{2^{\sharp 1}\}, \{0^{\sharp 1}, 1^{\sharp 1}\}, \{0^{\sharp 1}, 2^{\sharp 1}\},$$
$$\{1^{\sharp 1}, 2^{\sharp 1}\}, \{0^{\sharp 1}, 1^{\sharp 1}, 2^{\sharp 1}\}.$$

Assume now that $w = 0$. Let n be any natural number. Then the set of non-empty states is $\{\{0^{\sharp e}\} \mid 0 \leq e \leq n\}$. Assume that $n = 0$. Let w be any natural number, denoting the length of the input word. Then the set of non-empty states is $\{\{i^{\sharp 0}\} \mid 0 \leq i \leq w\}$.

Definition 11 *Let M be a non-empty state. The minimal number i such that M contains a position of the form $i^{\sharp e}$ (for some e) is called the* minimal boundary *of M.*

It is trivial to verify the following lemma.

Lemma 3 *Let M be a state with minimal boundary i and let $j^{\sharp f} \in M$. Then $j - i \leq n + f$.*

At various places we shall consider the union of two states M and N with a common base position $i^{\sharp 0}$. 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. Since the subsumption relation is well-founded, this operation is well-defined. Note that $M \sqcup N$ is again a state with base position $i^{\sharp 0}$. $M \sqcup N$ will be called the *reduced union* of M and N.

Elementary transitions. The transitions of the Levenshtein automaton of degree n will be defined with the help of transitions that act on single positions. The latter transitions are called *elementary transitions of degree n*. Intuitively, elementary transitions correspond to sets of transitions of a non- deterministic Levenshtein automaton starting from the same source state. The image of a position under an elementary transition with an input symbol x depends on the distribution of x in a subword of W.

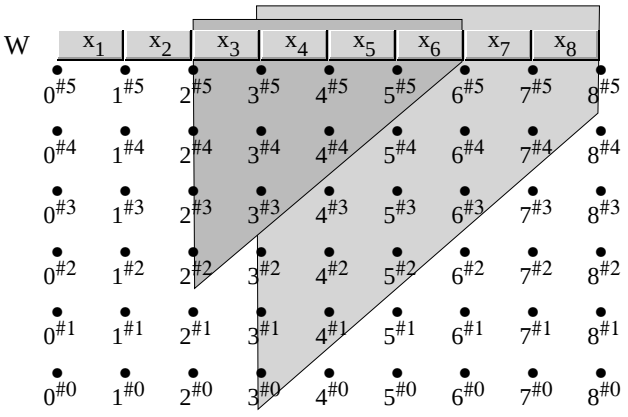

Fig. 6 Relevant subwords for elementary transitions (cf. Example 6)

Definition 12 *Let $W = x_1 \cdots x_w$ as above. Let $\pi := i^{\sharp e}$ be a position, and let $k := \min\{n - e + 1, w - i\}$. The* relevant subword of W for position π, *denoted $W_{[\pi]}$, is the subword $x_{i+1} \cdots x_{i+k}$ of W.*

Note that the length of $W_{[\pi]}$ cannot exceed $n + 1$.

Example 6 Let $w = 8$ and $n = 5$. Then the relevant subwords for positions $2^{\sharp 2}$ and $3^{\sharp 0}$ are $x_3x_4x_5x_6$ and $x_4x_5x_6x_7x_8$ respectively, as illustrated in Fig. 6.

Definition 13 *Let W and n be as above. An* elementary transition *assigns to each position $\pi = i^{\sharp e}$ and each symbol $x \in \Sigma$ a state $\delta(i^{\sharp e}, x)$. The complete set of elementary transitions is specified in Table 2. Notation $\langle 0, b_2, \ldots, b_k \rangle : j$ indicates that j is the minimal index in the set $\{2, \ldots, k\}$ where $b_j = 1$. This implies that such an index exists.*

Table 2 Table of elementary transitions for $\pi = i^{\sharp e}$

	(I) $0 \leq e \leq n - 1$
$i \leq w - 2$	$\delta(i^{\sharp e}, x) := \begin{cases} \{(i+1)^{\sharp e}\} \\ \quad \text{for } \chi(x, W_{[\pi]}) = \langle 1, b_2, \ldots, b_k \rangle, \\ \{i^{\sharp e+1}, (i+1)^{\sharp e+1}, (i+j)^{\sharp e+j-1}\} \\ \quad \text{for } \chi(x, W_{[\pi]}) = \langle 0, b_2, \ldots, b_k \rangle : j, \\ \{i^{\sharp e+1}, (i+1)^{\sharp e+1}\} \\ \quad \text{for } \chi(x, W_{[\pi]}) = \langle 0, \ldots, 0 \rangle. \end{cases}$
$i = w - 1$	$\delta(i^{\sharp e}, x) := \begin{cases} \{(i+1)^{\sharp e}\} \\ \quad \text{for } \chi(x, W_{[\pi]}) = \langle 1 \rangle, \\ \{i^{\sharp e+1}, (i+1)^{\sharp e+1}\} \\ \quad \text{for } \chi(x, W_{[\pi]}) = \langle 0 \rangle. \end{cases}$
$i = w$	$\delta(w^{\sharp e}, x) := \{w^{\sharp e+1}\}$
	(II) $e = n$
$i \leq w - 1$	$\delta(i^{\sharp n}, x) := \begin{cases} \{(i+1)^{\sharp n}\} & \text{for } \chi(x, W_{[\pi]}) = \langle 1 \rangle, \\ \emptyset & \text{for } \chi(x, W_{[\pi]}) = \langle 0 \rangle. \end{cases}$
$i = w$	$\delta(w^{\sharp n}, x) := \emptyset$.

The following – informal – comments explain the intuition behind these transitions. In Part I of the table for $i \leq w-1$ we distinguish three situations:

1. The first entry of $\chi(x, W_{[\pi]})$ is 1.
2. The first entry of $\chi(x, W_{[\pi]})$ is 0, but $\chi(x, W_{[\pi]})$ has an entry 1, the minimal one has index j.
3. All entries of $\chi(x, W_{[\pi]})$ are 0.

In Situation 3, x does not occur in $W_{[\pi]}$. The transition can be interpreted as a default transition. Image element $i^{\sharp e+1}$ captures the insertion of x at boundary i, image element $(i+1)^{\sharp e+1}$ captures the substitution of x_{i+1} with x. Other possible explanations for the occurrence of x are covered via subsumption. For example, assume that x_{i+1} is deleted and x_{i+2} is substituted by x. The position reached in this case is $(i+2)^{\sharp e+2}$. We do not add this position to the image set since it is subsumed by $(i+1)^{\sharp e+1}$. Note that default transitions for $e = n$ lead to the failure state $\emptyset$ (cf. Part II).

In Situation 2, the image element $i^{\sharp e+1}$ again covers the situation where symbol x is inserted before x_{i+1}. Element $(i+1)^{\sharp e+1}$ covers the situation where x_{i+1} is substituted by x. Element $(i+j)^{\sharp e+j-1}$ covers the situation where the elements $x_{i+1}, \ldots, x_{i+j-1}$ are deleted. The reader might wonder why only the entry 1 with minimal index j is essential. This entry corresponds to the first occurrence of x in $W_{[\pi]}$. Assume that $j = j_1 < \ldots < j_h$ is the list of all indices where $\chi(x, W_{[\pi]})$ has an entry 1. In this situation position $(i+j)^{\sharp e+j-1}$ subsumes all positions $(i+j_l)^{\sharp e+j_l-1}$ for $1 \neq l$. Hence, elimination of subsumed positions leads to the state $\{i^{\sharp e+1}, (i+1)^{\sharp e+1}, (i+j)^{\sharp e+j-1}\}$ which is used as image above. See Example 7 below for an illustration.

In Situation 1 we might expect that the image is rather the set $\{i^{\sharp e+1}, (i+1)^{\sharp e+1}, (i+1)^{\sharp e}\}$. But note that $(i+1)^{\sharp e}$ subsumes both other elements. Hence, via elimination of subsumed positions we arrive at $\{(i+1)^{\sharp e}\}$.

Example 7 Let $w = 8$ and $n = 5$. In Fig. 7 we consider the image of $\pi = 2^{\sharp 0}$ under $x \in \Sigma$. We have

$$W_{[\pi]} = x_3 \cdots x_8.$$

We assume that

$$\chi(x, W_{[\pi]}) = \langle 0, 0, 1, 0, 1, 1 \rangle.$$

This means that x_5, x_7 and x_8 are the symbols of $W_{[\pi]}$ that are identical to x. In this situation we have $\delta(2^{\sharp 0}, x) = \{2^{\sharp 1}, 3^{\sharp 1}, 5^{\sharp 2}\}$ as illustrated in Fig. 7.

The proof of the next lemma is straightforward.

Lemma 4 *Let M be a set of positions. Assume that all elements of M are subsumed by $i^{\sharp e}$ where $i < w$. Then the image of any element of M under any elementary transition contains only positions that are subsumed by $(i+1)^{\sharp e}$. If all elements of M are subsumed by $w^{\sharp e}$, then the image of any element of M under any elementary transition has only positions that are subsumed by $w^{\sharp e}$.*

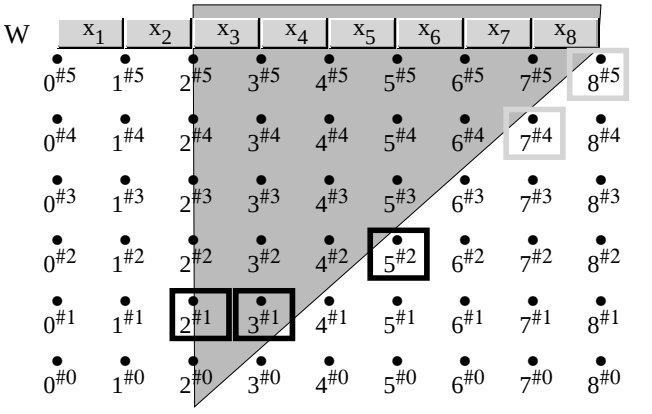

Fig. 7 Geometric interpretation of elementary transitions (cf. Example 7)

We now ask how the elementary transitions for Levenshtein automata of different degrees n for the same input word W are related. In order to avoid notational ambiguities we write $\delta^{(n)}$ for the function that describes the elementary transitions for the Levenshtein automaton of degree n. If e is a natural number, we write $[i^{\sharp f}]^{\sharp e}$ for the "lifted" position $i^{\sharp f+e}$ (it will be guaranteed that each such expression in fact denotes a position). If M is a state, we define the lifted version $[M]^{\sharp e} := \{[\pi]^{\sharp e} \mid \pi \in M\}$. The following lemma shows that once the elementary transitions from positions $i^{\sharp 0}$ for degrees $0, \ldots, n-1$ are fixed, the elementary transitions of degree n for positions $i^{\sharp e}$ for $1 \leq e \leq n$ are simply defined by "raising".

Lemma 5 (Raising Lemma for elementary transitions) *Let $n > 0$ and $1 \leq e \leq n$. Then for any position $i^{\sharp e}$ of degree n and any $x \in \Sigma$ we have*

$$\delta^{(n)}(i^{\sharp e}, x) = [\delta^{(n-e)}(i^{\sharp 0}, x)]^{\sharp e}.$$

Proof Since $\min\{n-e+1, w-i\} = \min\{(n-e)-0+1, w-i\}$ it follows that the relevant subword for $i^{\sharp e}$ for degree n is identical to the relevant subword of $i^{\sharp 0}$ for degree $n-e$. We may denote it in the form W_r. First assume that $i \leq w-2$ and the first entry of $\chi(x, W_r)$ is 1. Then we have

$$\begin{aligned} \delta^{(n)}(i^{\sharp e}, x) &= \{(i+1)^{\sharp e}\} \\ &= [\{(i+1)^{\sharp 0}\}]^{\sharp e} \\ &= [\delta^{(n-e)}(i^{\sharp 0}, x)]^{\sharp e}. \end{aligned}$$

The remaining cases are similar. □

The Levenshtein automaton. We now introduce the family of Levenshtein automata that we use for string correction.

Definition 14 *Let $W = x_1 \cdots x_w$ where $w \geq 0$, let $n \geq 0$. Then $LEV_n(W)$ is the deterministic finite state automaton $\langle \Sigma, Q, q_0, F, \Delta \rangle$ with the following properties:*

1. *The set of* states Q *contains all states in the sense of Definition 10.*
2. *The* initial state *is* $q_0 := \{0^{\sharp 0}\}$.
3. *The set* F *of* final states *contains all states* $M \in Q$ *that contain an accepting position.*
4. *The* transition function Δ *is defined in the following way: for any symbol* $y \in \Sigma$ *and any state* $M \in Q$, $\Delta(M, y) := \bigsqcup_{\pi \in M} \delta(\pi, y)$.

Theorem 1 *$LEV_n(W)$ is a deterministic and acyclic Levenshtein automaton of degree n for W. For fixed degree n, the size of $LEV_n(W)$ is linear in $|W|$.*

The proof of Theorem 1 is given in the Appendix. The rest of this section will be used to introduce some notions that help to obtain a concrete description of the transition function Δ of $LEV_n(W)$ in the situation where the degree n is fixed. Recall that the above definition of Δ is indirect in the sense that the image of a state is only defined in terms of the images of its members under elementary transitions. Clearly, if M is a state and $x \in \Sigma$, in order to directly define the image $\Delta(M, x)$ we have to distinguish appropriate subcases that take the distribution of the occurrences of x in W into account. As it turns out, it suffices to consider the occurrences of x in a particular subword of W.

Definition 15 *Let W and n as above. Let M be a non-empty state with minimal boundary i. Let $k := \min\{2n+1, w-i\}$. The* relevant subword of M, *denoted $W_{[M]}$, is the subword*

$$x_{i+1} \cdots x_{i+k}$$

of W.

Since the relevant subword does not depend on the state M itself, but only on the minimal boundary i of M, we also write $W_{[i]}$ for $W_{[M]}$. Note that the length of $W_{[i]}$ cannot exceed $2n+1$. It follows from Lemma 3 and from Definitions 12 and 15 that for each position $\pi \in M$ always $W_{[\pi]}$ is a subword of $W_{[M]}$. Table 2 shows that for any position π the image $\delta(\pi, x)$ only depends on vector $\chi(x, W_{[\pi]})$. Thus, given a state M, the image $\Delta(M, x)$ is completely determined by the characteristic vector $\chi(x, W_{[M]})$. In the following section we shall see that for fixed degree n this observation can be used to describe Δ in terms of a finite table.

Remark 3 The transition function Δ is completely determined by the characteristic vectors $\chi(x, W_{[i]})$ of symbols $x \in \Sigma$ with respect to the subwords $W_{[i]}$ of the form $W_{[i]} = x_{i+1} \cdots x_{i+k}$ where $k := \min\{2n+1, w-i\}$. If W and W' are two words of the same length, and if the $(2n+1)$-profile sequences of W and W' are identical, then $LEV_n(W)$ and $LEV_n(W')$ are isomorphic modulo transition labels.

5 Computation of deterministic Levenshtein automata of fixed degree

The general description of the Levensthein automaton $LEV_n(W)$ given in the previous section can be used to derive, for any *fixed* bound n, an algorithm that actually computes $LEV_n(W)$ in linear time, given any input word W. The principle will first be illustrated for degree $n = 1$.

5.1 Computing the Levensthein automaton of degree 1

Using the general description of the automaton $LEV_n(W)$, we derive a generic description of $LEV_1(W)$ for *arbitrary* input W in terms of the following:

- A parametric list of states, with a fixed initial state $\{0^{\sharp 0}\}$
- A parametric list of final states
- A table T_1 which gives a parametric description of the transition function Δ

Parametric list of states and final states. For input $W = x_1 \cdots x_w$ and $n = 1$ the list of positions is

$$0^{\sharp 0}, \ldots, w^{\sharp 0},$$
$$0^{\sharp 1}, \ldots, w^{\sharp 1}.$$

It follows easily from Definition 10 that we have the following states:

$$\begin{aligned}
&\emptyset \quad \text{failure state},\\
&A_i := \{i^{\sharp 0}\} \quad (0 \le i \le w),\\
&B_i := \{i^{\sharp 1}\} \quad (0 \le i \le w),\\
&C_i := \{i^{\sharp 1}, (i+1)^{\sharp 1}\} \quad (0 \le i \le w-1),\\
&D_i := \{i^{\sharp 1}, (i+2)^{\sharp 1}\} \quad (0 \le i \le w-2),\\
&E_i := \{i^{\sharp 1}, (i+1)^{\sharp 1}, (i+2)^{\sharp 1}\}\\
&\qquad\qquad (0 \le i \le w-2).
\end{aligned}$$

The initial state is A_0. Accepting positions are $w^{\sharp 1}$, $w^{\sharp 0}$, as well as $(w-1)^{\sharp 0}$ for $w \ge 1$. It follows immediately that the final states are

$$\begin{aligned}
&A_w, A_{w-1}, B_w, C_{w-1}, D_{w-2}, E_{w-2}\\
&\quad \text{for } w \ge 2,\\
&A_w, A_{w-1}, B_w, C_{w-1}\\
&\quad \text{for } w = 1,\\
&A_w, B_w\\
&\quad \text{for } w = 0.
\end{aligned}$$

Parametric description of the transitions function. In order to derive the parametric description of the transition function, we first refine the general description of elementary descriptions given in Table 2. For the case $n = 1$, we obtain the set of elementary transitions given in Table 3. Using this table it is simple to compute a

Table 3 Table of elementary transitions for degree 1

(I) $e = 0$	
$i \leq w-2$	$\delta(i^{\sharp 0}, x) := \begin{cases} \{(i+1)^{\sharp 0}\} \\ \quad \text{for } \chi(x, x_{i+1}x_{i+2}) = \langle 1, b_2\rangle, \\ \{i^{\sharp 1}, (i+1)^{\sharp 1}, (i+2)^{\sharp 1}\} \\ \quad \text{for } \chi(x, x_{i+1}x_{i+2}) = \langle 0, 1\rangle, \\ \{i^{\sharp 1}, (i+1)^{\sharp 1}\} \\ \quad \text{for } \chi(x, x_{i+1}x_{i+2}) = \langle 0, 0\rangle. \end{cases}$
$i = w-1$	$\delta(i^{\sharp 0}, x) := \begin{cases} \{(i+1)^{\sharp 0}\} \\ \quad \text{for } \chi(x, x_{i+1}) = \langle 1\rangle, \\ \{i^{\sharp 1}, (i+1)^{\sharp 1}\} \\ \quad \text{for } \chi(x, x_{i+1}) = \langle 0\rangle. \end{cases}$
$i = w$	$\delta(w^{\sharp 0}, x) := \{w^{\sharp 1}\}$
(II) $e = 1$	
$i \leq w-1$	$\delta(i^{\sharp 1}, x) := \begin{cases} \{(i+1)^{\sharp 1}\} & \text{for } \chi(x, x_{i+1}) = \langle 1\rangle, \\ \emptyset & \text{for } \chi(x, x_{i+1}) = \langle 0\rangle. \end{cases}$
$i = w$	$\delta(w^{\sharp 1}, x) := \emptyset$.

parametric description of the full transition function Δ, following the remarks at the end of the previous section. The description is given in Table 4. Images $\Delta(M, x)$ are specified using a subcase analysis where the possible characteristic vectors $\chi(x, W_{[M]})$ are distinguished. The following example shows how the entries of Table 4 are computed.

Example 8 Assume we ask for the image $\Delta(C_i, x)$ of state $C_i = \{i^{\sharp 1}, (i+1)^{\sharp 1}\}$ under $x \in \Sigma$, where $i \leq w-3$ and

$$\begin{aligned} \chi(x, W_{[C_i]}) &= \chi(x, x_{i+1}x_{i+2}x_{i+3}) \\ &= \langle 1, 1, 0\rangle. \end{aligned}$$

We have

$$\begin{aligned} W_{[i^{\sharp 1}]} &= x_{i+1} \\ W_{[(i+1)^{\sharp 1}]} &= x_{i+2}, \end{aligned}$$

hence

$$\chi(x, W_{[i^{\sharp 1}]}) = \langle 1\rangle = \chi(x, W_{[(i+1)^{\sharp 1}]}).$$

Using Table 3 we obtain

$$\begin{aligned} \Delta(C_i, x) &= \delta(i^{\sharp 1}, x) \sqcup \delta((i+1)^{\sharp 1}, x) \\ &= \{(i+1)^{\sharp 1}\} \sqcup \{(i+2)^{\sharp 1}\} \\ &= \{(i+1)^{\sharp 1}, (i+2)^{\sharp 1}\} \\ &= C_{i+1}. \end{aligned}$$

In the same way, all other entries of Table 4 can be computed.

Table 4 Parametric transitions for $LEV_1(W)$

$0 \leq i \leq w-3$

$\chi(x, x_{i+1}x_{i+2}x_{i+3})$	A_i	B_i	C_i	D_i	E_i
$\langle 0,0,0\rangle$	C_i	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$\langle 1,0,0\rangle$	A_{i+1}	B_{i+1}	B_{i+1}	B_{i+1}	B_{i+1}
$\langle 0,1,0\rangle$	E_i	$\emptyset$	B_{i+2}	$\emptyset$	B_{i+2}
$\langle 0,0,1\rangle$	C_i	$\emptyset$	$\emptyset$	B_{i+3}	B_{i+3}
$\langle 1,1,0\rangle$	A_{i+1}	B_{i+1}	C_{i+1}	B_{i+1}	C_{i+1}
$\langle 1,0,1\rangle$	A_{i+1}	B_{i+1}	B_{i+1}	D_{i+1}	D_{i+1}
$\langle 0,1,1\rangle$	E_i	$\emptyset$	B_{i+2}	B_{i+3}	C_{i+2}
$\langle 1,1,1\rangle$	A_{i+1}	B_{i+1}	C_{i+1}	D_{i+1}	E_{i+1}

$i = w-2$

$\chi(x, x_{i+1}x_{i+2})$	A_i	B_i	C_i	D_i	E_i
$\langle 0,0\rangle$	C_i	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$\langle 1,0\rangle$	A_{i+1}	B_{i+1}	B_{i+1}	B_{i+1}	B_{i+1}
$\langle 0,1\rangle$	E_i	$\emptyset$	B_{i+2}	$\emptyset$	B_{i+2}
$\langle 1,1\rangle$	A_{i+1}	B_{i+1}	C_{i+1}	B_{i+1}	C_{i+1}

$i = w-1$

$\chi(x, x_{i+1})$	A_i	B_i	C_i
$\langle 0\rangle$	C_i	$\emptyset$	$\emptyset$
$\langle 1\rangle$	A_{i+1}	B_{i+1}	B_{i+1}

$i = w$

$\chi(x, \varepsilon)$	A_i	B_i
$\langle\rangle$	B_i	$\emptyset$

Computation of the actual automaton. Obviously, given the above generic description of LEV_1 it is possible to generate for any concrete input W the automaton $LEV_1(W)$ in time $O(|W|)$.

Theorem 2 *There exists an algorithm that computes for any input word W the automaton $LEV_1(W)$ in time and space $O(|W|)$.*

Corollary 1 *For any input W, the minimal deterministic Levenshtein automaton of degree 1 for W can be computed in time and space $O(|W|)$.*

Proof A result by D. Revuz shows that acyclic deterministic finite state automata can be minimalized in linear time [Rev92]. Since $LEV_1(W)$ is deterministic and acyclic the result follows. □

Example 9 Figure 8 describes the automaton $LEV_1(W)$ for the input word *atlas*. For each word W of length 5 with 3-profile sequence

$$(1,2,3), (1,2,3), (1,2,3)$$

the automaton $LEV_1(W)$ has the same structure, modulo renaming of transition labels. Similarly Fig. 9 describes the structure of $LEV_1(W)$ for the word *otter*. Here for each word W of length 5 with 3-profile sequence

$$(1,2,2), (1,1,2), (1,2,3)$$

the automaton $LEV_1(W)$ has the same structure, modulo renaming of transition labels.

5.2 Computing Levensthein automata of higher degree

For any fixed degree $n \geq 2$, the computation of $LEV_n(W)$ essentially follows the same ideas as in the case $n = 1$. Given the degree n, an *offline computation* is used to compute the following:

1. A parametric description of the set of all states of $LEV_n(W)$ for arbitrary input word W, using the minimal boundary i of states as a parameter
2. A parametric description of the set of all final states
3. A parametric transition table T_n that defines the images of parametric states M under input $x \in \Sigma$, subject to the form of the characteristic vectors $\chi(x, W_{[M]})$

In each case, the initial state is $\{0^{\sharp 0}\}$. Once we have the parametric description of $LEV_n(W)$ for arbitrary W at our disposal, we may use it to compute for any concrete input word W the automaton $LEV_n(W)$ in time linear in $|W|$.

Theorem 3 *For any fixed degree n, there exists an algorithm that computes for input word W the automaton $LEV_n(W)$ in time and space $O(|W|)$.*

Corollary 2 *For any input W, the minimal deterministic Levenshtein automaton of fixed degree n for W can be computed in time and space $O(|W|)$.*

Proof As in the case $n = 1$. □

Remark 4 Whereas five parametric states (i.e., A_i, B_i, B_i, D_i, and E_i) are sufficient for degree $n = 1$, the number of parametric states that are needed for degrees $2, 3, 4, \ldots$ grows quickly. For $n = 2$ there are 30 parametric states (ignoring state $\emptyset$), which are listed in Table 5. Since relevant subwords $W_{[M]}$ may have length $2n + 1 = 5$, the boolean vectors that have to be considered when defining the transition function have maximal length 5. Hence the maximal subtable of Δ has dimension 30×32. For $n = 3$, the number of parametric states is 196, the maximal subtable for Δ has dimension 196×128. For $n = 4$, there are alread 1353 parametric states, the maximal subtable for Δ has dimension 1352×512.

Table 5 Non-empty parametric states of $LEV_2(W)$

$$
\begin{aligned}
&\{i^{\sharp 0}\} && (0 \leq i \leq w),\\
&\{i^{\sharp 1}\} && (0 \leq i \leq w),\\
&\{i^{\sharp 1}, (i+1)^{\sharp 1}\} && (0 \leq i \leq w-1),\\
&\{i^{\sharp 1}, (i+2)^{\sharp 1}\} && (0 \leq i \leq w-2),\\
&\{i^{\sharp 1}, (i+1)^{\sharp 1}, (i+2)^{\sharp 1}\} && (0 \leq i \leq w-2),\\
&\{i^{\sharp 2}, (i+2)^{\sharp 1}\} && (0 \leq i \leq w-2),\\
&\{i^{\sharp 2}, (i+3)^{\sharp 1}\} && (0 \leq i \leq w-3),\\
&\{i^{\sharp 2}, (i+4)^{\sharp 2}, (i+2)^{\sharp 1}\} && (0 \leq i \leq w-4),\\
&\{i^{\sharp 2}, (i+1)^{\sharp 2}, (i+3)^{\sharp 1}\} && (0 \leq i \leq w-3),\\
&\{i^{\sharp 2}, (i+2)^{\sharp 1}, (i+3)^{\sharp 1}\} && (0 \leq i \leq w-3),\\
&\{i^{\sharp 1}, (i+3)^{\sharp 2}\} && (0 \leq i \leq w-3),\\
&\{i^{\sharp 1}, (i+2)^{\sharp 2}\} && (0 \leq i \leq w-2),\\
&\{i^{\sharp 1}, (i+1)^{\sharp 1}, (i+3)^{\sharp 2}\} && (0 \leq i \leq w-3),\\
&\{i^{\sharp 1}, (i+2)^{\sharp 2}, (i+3)^{\sharp 2}\} && (0 \leq i \leq w-3),\\
&\{i^{\sharp 2}\} && (0 \leq i \leq w),\\
&\{i^{\sharp 2}, (i+1)^{\sharp 2}\} && (0 \leq i \leq w-1),\\
&\{i^{\sharp 2}, (i+2)^{\sharp 2}\} && (0 \leq i \leq w-2),\\
&\{i^{\sharp 2}, (i+3)^{\sharp 2}\} && (0 \leq i \leq w-3),\\
&\{i^{\sharp 2}, (i+4)^{\sharp 2}\} && (0 \leq i \leq w-4),\\
&\{i^{\sharp 2}, (i+1)^{\sharp 2}, (i+2)^{\sharp 2}\} && (0 \leq i \leq w-2),\\
&\{i^{\sharp 2}, (i+1)^{\sharp 2}, (i+3)^{\sharp 2}\} && (0 \leq i \leq w-3),\\
&\{i^{\sharp 2}, (i+1)^{\sharp 2}, (i+4)^{\sharp 2}\} && (0 \leq i \leq w-4),\\
&\{i^{\sharp 2}, (i+2)^{\sharp 2}, (i+3)^{\sharp 2}\} && (0 \leq i \leq w-3),\\
&\{i^{\sharp 2}, (i+2)^{\sharp 2}, (i+4)^{\sharp 2}\} && (0 \leq i \leq w-4),\\
&\{i^{\sharp 2}, (i+3)^{\sharp 2}, (i+4)^{\sharp 2}\} && (0 \leq i \leq w-4)\\
&\{i^{\sharp 2}, (i+1)^{\sharp 2}, (i+2)^{\sharp 2}, (i+3)^{\sharp 2}\} && (0 \leq i \leq w-3),\\
&\{i^{\sharp 2}, (i+1)^{\sharp 2}, (i+2)^{\sharp 2}, (i+4)^{\sharp 2}\} && (0 \leq i \leq w-4),\\
&\{i^{\sharp 2}, (i+1)^{\sharp 2}, (i+3)^{\sharp 2}, (i+4)^{\sharp 2}\} && (0 \leq i \leq w-4),\\
&\{i^{\sharp 2}, (i+2)^{\sharp 2}, (i+3)^{\sharp 2}, (i+4)^{\sharp 2}\} && (0 \leq i \leq w-4),\\
&\{i^{\sharp 2}, (i+1)^{\sharp 2}, (i+2)^{\sharp 2}, (i+3)^{\sharp 2}, (i+4)^{\sharp 2}\} && (0 \leq i \leq w-4).
\end{aligned}
$$

6 String correction using imitation of Levensthein automata

We now introduce a variant of the correction method described in Sect. 3. The main advantage of the new method is that it avoids the actual computation of Levenshtein automata. As before we assume that for some fixed degree n we have at our disposal a generic description of the automaton

$$LEV_n(W) = \langle \Sigma, Q^W, \{0^{\sharp 0}\}, F^W, \Delta^W \rangle$$

for arbitrary input W, as presented in the previous section for degree $n = 1$. With Δ_χ^W we denote the variant of the transition function where characteristic vectors are treated as input. Note that the tables T_n yield parametric descriptions of Δ_χ^W. For example, from Table 4 we see that A_0 is mapped to C_0 under vector $\langle 0, 0, 0 \rangle$.

As in Sect. 3 we assume that the dictionary is implemented in the form of a deterministic finite state automaton $A_D = \langle \Sigma, Q^D, q_0^D, F^D, \delta^D \rangle$.

Given any concrete input word W, we first compute the set of all characteristic vectors of the form $\chi(x, W_{[i]})$

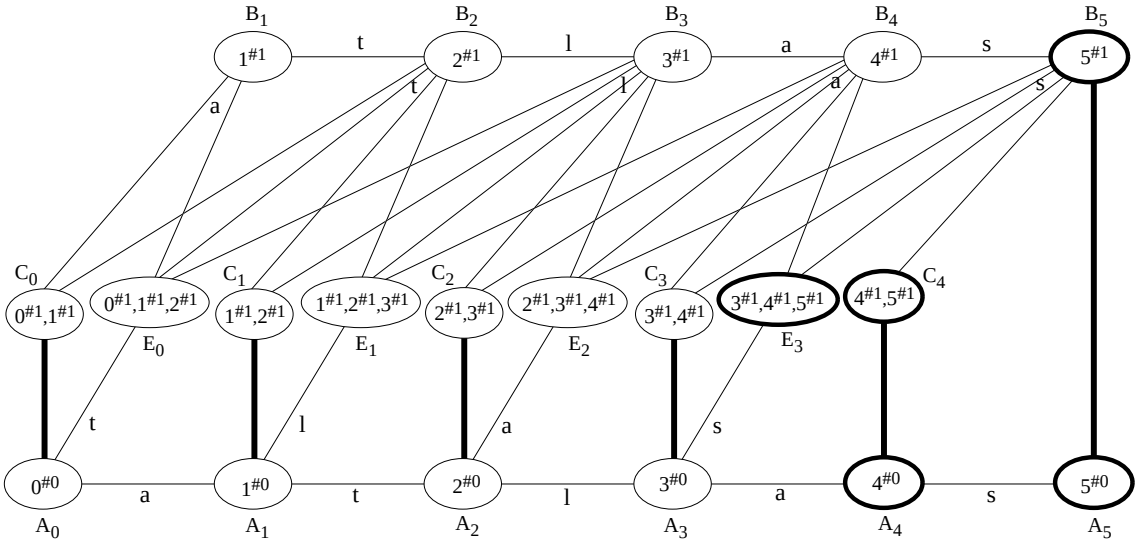

Fig. 8 Deterministic Levenshtein automaton $LEV_1(W)$ for input W = "atlas"

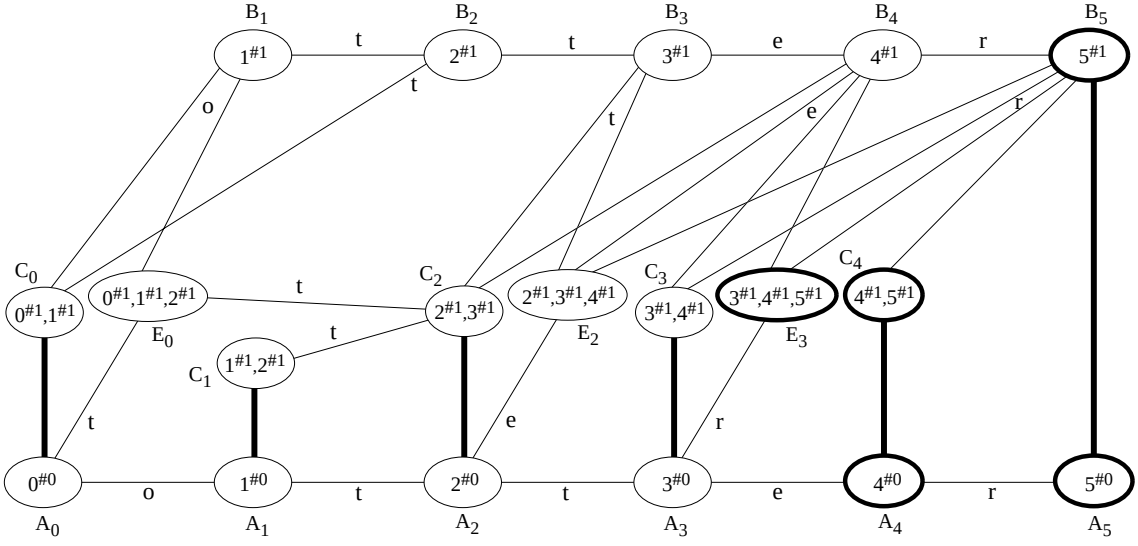

Fig. 9 Deterministic Levenshtein automaton $LEV_1(W)$ for input W = "otter"

where $x \in \Sigma$ and i denotes a boundary of W. Using this vectors, the backtracking procedure given in Sect. 3 can now be replaced by the variant depicted in Fig. 10: Note that in contrast to the situation described in Sect. 3, we do not assume that the Levenshtein automaton for the concrete input word W is available. Given the generic description of LEV_n, states of $LEV_n(W)$ are only introduced on demand in line 6. It is important to note that each image state $\Delta_\chi^W(M, \chi(x, W_{[M]}))$ can be found in constant time since both $\chi(x, W_{[M]})$ and the table T_n for Δ_χ have been precomputed. The following example illustrates the modified acceptance procedure.

Example 10 We consider the case $n = 1$. Assume that the (misspelled) input word W has the form *chold*. We consider the path of the dictionary automaton for the dictionary entry *child*, which is assumed to lead to a final state. The following transition sequence illustrates

```
push (<ε, q_0^D, {0^#0}>);
while not empty(stack) do begin
   pop (<V, q^D, M>);
   for x in Σ do begin
      q_1^D := δ^D(q^D, x);
      M' := Δ_χ^W(M, χ(x, W_[M]));
      if (q_1^D <> NIL) and (M' <> NIL) then begin
         V_1 := concat(V, x);
         push(<V_1, q_1^D, M'>);
         if (q_1^D ∈ F^D) and (M' ∈ F^W)
         then output(V_1);
      end;
   end;
end;
```

Fig. 10 Backtracking procedure for lexical traversal with imitation of Levenshtein automata

Table 6 Number and lengths of test words for evaluation with BL

Length	3	4	5	6	7	8
♯ prefixes	3152	12 121	30 243	59 835	101 763	150 046
Length	9	10	11	12	13	14
♯ prefixes	190 318	203 520	184 138	139 982	91 252	52 603
Length	15	16	17	18	19	20
♯ prefixes	27 997	14 763	8179	4601	2790	1585

Table 7 Results for BL, standard Levenshtein distance, bound $n = 1$

Length	LA	PT	TCT1	TCT2	NC
3	0.228	0.152	0.381	0.324	10.31
4	0.251	0.183	0.434	0.351	8.66
5	0.272	0.201	0.473	0.351	6.97
6	0.294	0.213	0.507	0.355	6.66
7	0.318	0.222	0.540	0.362	6.44
8	0.340	0.233	0.573	0.375	6.34
9	0.362	0.244	0.607	0.389	5.84
10	0.385	0.255	0.639	0.403	5.25
11	0.410	0.261	0.671	0.413	4.65
12	0.430	0.270	0.700	0.421	4.01
13	0.452	0.273	0.726	0.424	3.61
14	0.474	0.273	0.748	0.422	3.24
15	0.497	0.270	0.767	0.414	2.98
16	0.520	0.266	0.786	0.404	2.73
17	0.544	0.260	0.804	0.400	2.62
18	0.565	0.263	0.828	0.398	2.51
19	0.588	0.262	0.849	0.394	2.35

how states of $LEV_1(chold)$ are generated on demand, using precomputed characteristic vectors and Table 4.

$$\begin{aligned} A_0 \text{ input } \chi(c, cho) = \langle 1,0,0\rangle &\mapsto A_1 \\ A_1 \text{ input } \chi(h, hol) = \langle 1,0,0\rangle &\mapsto A_2 \\ A_2 \text{ input } \chi(i, old) = \langle 0,0,0\rangle &\mapsto C_2 \\ C_2 \text{ input } \chi(l, old) = \langle 0,1,0\rangle &\mapsto B_4 \\ B_4 \text{ input } \chi(d, d) = \langle 1\rangle &\mapsto B_5 \end{aligned}$$

Now B_5 is a final state. Hence, in the above procedure, the word *child* is suggested as one correction of the input *chold*. Assume now that the dictionary also contains the word *cold*. In this case we reach the following states of $LEV_1(chold)$:

$$\begin{aligned} A_0 \text{ input } \chi(c, cho) = \langle 1,0,0\rangle &\mapsto A_1 \\ A_1 \text{ input } \chi(o, hol) = \langle 0,1,0\rangle &\mapsto E_1 \\ E_1 \text{ input } \chi(l, hol) = \langle 0,0,1\rangle &\mapsto B_4 \\ B_4 \text{ input } \chi(d, d) = \langle 1\rangle &\mapsto B_5 \end{aligned}$$

Since B_5 is final, also *cold* is suggested as a correction candidate.

Table 8 Results for BL, standard Levenshtein distance, bound $n = 2$

Length	LA	PT	TCT1	TCT2	NC
3	1.40	2.18	3.57	3.19	227
4	1.62	2.56	4.18	3.68	175
5	1.78	2.76	4.55	3.74	111
6	1.94	2.85	4.80	3.75	76.2
7	2.10	2.91	5.02	3.65	57.3
8	2.26	3.00	5.26	3.64	48.2
9	2.42	3.08	5.50	3.66	39.1
10	2.58	3.16	5.74	3.72	32.2
11	2.74	3.22	5.95	3.77	26.2
12	2.89	3.25	6.15	3.80	20.7
13	3.06	3.27	6.33	3.81	16.3
14	3.21	3.25	6.46	3.77	12.8
15	3.37	3.19	6.56	3.70	10.7
16	3.53	3.12	6.65	3.63	9.19
17	3.68	3.08	6.77	3.58	8.29
18	3.84	3.05	6.89	3.54	7.84
19	4.00	3.03	7.03	3.52	7.35

Table 9 Results for BL, standard Levenshtein distance, bound $n = 3$

Length	LA	PT	TCT1	TCT2	NC
3	13.3	11.6	25.0	16.1	2411
4	14.8	13.8	28.6	18.9	2108
5	16.3	15.0	31.4	20.4	1397
6	18.0	15.7	33.7	21.2	852
7	19.2	16.0	35.2	20.9	538
8	20.3	16.4	36.7	20.9	381
9	21.5	16.7	38.2	20.6	269
10	22.8	16.9	39.7	20.4	192
11	23.9	17.1	41.1	20.3	138
12	27.6	22.8	50.4	25.6	96.0
13	30.9	21.7	52.6	25.2	63.7
14	32.2	21.9	54.2	25.1	41.8
15	33.6	20.9	54.5	24.0	28.6
16	34.8	21.2	56.0	24.0	21.8
17	36.8	20.1	56.9	23.8	18.3
18	38.1	19.8	57.8	23.0	16.0
19	39.0	19.9	58.9	22.9	14.6

7 Experimental results

Experimental results were made using a Bulgarian lexicon (BL) with 870 000 word entries and a dictionary of German composite nouns (GL) with 6 058 198 entries. The following algorithms were implemented in C and tested on 500 MHz (BL) and 600 MHz (GL) Pentium III machines under Linux:

– The algorithm for computing, given input W, the automaton $LEV_n(W)$ $(n = 1, 2, 3)$

Table 10 Results for BL, Levenshtein distance where transpositions are treated as primitive edit operations, bounds $n = 1, 2, 3$, times in milliseconds

Length	$n=1$ time	NC	$n=2$ time	NC	$n=3$ time	NC
3	0.330	10.4	4.03	231	17.0	2143
4	0.362	8.75	4.58	178	21.0	2139
5	0.357	7,04	4.67	114	23.5	1428
6	0.360	6.69	4.69	77.3	21.5	869
7	0.367	6.46	4.57	57.8	21.2	547
8	0.379	6.36	4.58	48.6	21.5	386
9	0.394	5.85	4.62	39.4	21.1	272
10	0.407	5.25	4.69	32.3	22.6	194
11	0.416	4.66	4.85	26.4	20.9	140
12	0.428	4.10	4.78	20.8	20.7	96.8
13	0.430	3.62	4.78	16.3	20.5	64.1
14	0.428	3.24	4.73	12.8	21.9	42.0
15	0.424	2.98	4.74	10.8	21.3	26.7
16	0.416	2.73	4.59	9.21	20.9	21.8
17	0.410	2.62	4.54	8.30	21.0	18.3
18	0.404	2.51	4.50	7.85	24.5	15.9
19	0.405	2.35	4.43	7.36	26.7	14.6

Table 11 Results for BL, Levenshtein distance where merges and splits are treated as primitive edit operations, bounds $n = 1, 2, 3$, times in milliseconds

Length	$n=1$ time	NC	$n=2$ time	NC	$n=3$ time	NC
3	1.86	48.1	30.3	3216	139	$> 10^4$
4	1.79	31.6	32.3	2125	161	$> 10^4$
5	1.79	21.3	33.7	1195	167	9998
6	1.78	16.7	34.0	667	175	8050
7	1.82	14.8	31.8	404	182	8106
8	2.32	13.9	31.4	278	186	5606
9	2.38	12.6	31.5	197	184	3654
10	2.41	11.1	31.8	144	174	2291
11	2.43	9.64	34.2	107	168	1433
12	2.47	8.19	34.3	77.5	169	872
13	2.47	6.88	33.8	53.8	175	493
14	2.45	5.85	35.4	36.8	171	257
15	2.39	5.20	30.6	26.1	170	116
16	2.34	4.65	30.1	20.2	165	54.7
17	2.29	4.26	29.7	17.0	166	35.8
18	2.25	4.17	29.5	14.8	166	27.9
19	2.22	3.85	29.3	13.7	163	24.3

- The correction algorithm based on Levenshtein automata described in Sect. 3 ($n = 1, 2, 3$)
- The correction algorithm based on imitation of Levensthein automata described in Sect. 6 ($n = 1, 2, 3$)
- The variants of the above algorithms for the modified Levenshtein distances where transpositions (merges and splits) are treated as additional edit operations (see [SM01] for a detailed description of Levenshtein automata for these modified distances)

Table 12 Results for Bulgarian word list with random errors introduced, standard Levenshtein distance, bounds $n = 1, 2, 3$

Length	$n=1$ time	NC	$n=2$ time	NC	$n=3$ time	NC
5	0.30	5.34	3.80	107.11	22.31	1490.85
6	0.31	3.97	3.81	61.73	23.25	812.17
7	0.32	3.11	3.77	37.06	23.02	432.94
8	0.32	2.68	3.78	25.05	22.93	246.79
9	0.33	2.22	3.79	16.53	22.60	134.84
10	0.33	1.75	3.76	10.72	22.27	71.42
11	0.34	1.44	3.81	7.32	21.96	37.67
12	0.34	1.17	3.79	5.12	21.86	20.44
13	0.34	1.03	3.77	4.01	21.85	13.19
14	0.35	0.94	3.88	3.47	21.92	9.93
15	0.35	0.85	3.77	3.04	21.60	7.93
16	0.37	0.83	3.87	2.89	22.02	7.34
17	0.39	0.76	3.92	2.50	21.95	6.47
18	0.40	0.74	3.92	2.25	22.06	5.58
19	0.25	0.58	3.84	1.94	21.70	4.93

Evaluation of correction with BL. For the Bulgarian lexicon, we used the prefixes of length $3, 4, \ldots, 19$ of all dictionary words as garbled input "words" and computed the correction candidates. The number of test words of each length is given in Table 6.

The tables given below describe the results for the following corrections:

1. Correction with BL and standard Levenshtein distance with bound $n = 1, 2, 3$ (Tables 7, 8 and 9)
2. Correction with BL and Levenshtein distance where transpositions are treated as primitive edit operations, with bounds $n = 1, 2, 3$ (Table 10)
3. Correction with BL and Levenshtein distance where merges and splits are treated as primitive edit operations, with bound $n = 1, 2, 3$ (Table 11)

In Tables 7, 8 and 9, column 1 gives the length of the input words. Column 2 (LA) describes the average time that is needed to compute the Levenshtein automaton for an input word. Column 3 describes the average time that is needed for parallel traversal (PT) of dictionary automaton and Levenshtein automaton. Column 4 (TCT1) gives the average total correction time for the correction method based on computation of Levenshtein automata. Column 6 (TCT2) gives the average total correction time for the correction method based on imitation of Levenshtein automata. Column 7 (NC) yields the average number of correction candidates (dictionary words within the given distance bound) per input word. Times are in milliseconds. It is important to note that the time that is needed to output the correction candidates is always included.

As a summary, the second correction method based on simulation of Levenshtein automata is more efficient. The smaller number of correction candidates for large prefixes leads to the effect that for prefixes of length > 13 correction times decrease for longer input words when using the second correction method. The use of transpositions as primitive edit operations does not influence correction times and the number of correction candidates in a significant way. In contrast, a much longer search is needed when treating merges and splits as additional primitive edit operations. Both correction times and number of correction candidates grow.

In another series of experiments, all lexical words of a large Bulgarian corpus were garbled in order to simulate recognition errors. For each word, we randomly chose a number $m \in \{1, 2, 3\}$ of edit operations (substitutions, deletions, insertions) to be applied. The type and position of the edit operations, as well as the image symbols for substitutions and insertions were also randomly selected. The resulting list of garbled words was sorted according to the length of the words. The number of words per equivalence class was between 159 (length 19) and 19 220 (length 8). We then used the second correction method based on imitation of Levenshtein automata with bounds $n = 1, 2, 3$ (standard distance) and computed correction candidates. Table 12 gives the average correction times and the average number of correction candidates per word, for each equivalence class (lengths $5, \ldots, 19$).

Evaluation of correction with GL. Table 13 describes the results for correction with the german dictionary of composite nouns GL with 6 058 198 entries. For each length $l = 5, \ldots, 19$, we randomly selected 1000 prefixes of length l of entries and computed for each prefix all entries of GL where the standard Levenshtein distance does not exceed bound $n = 1, 2, 3$. We give the correction time (including output of correction candidates) and the average number of corrections.

Remark 5 Oflazer gives the following average correction times for a German dictionary with 174 573 words. For distance bound $n = 1$, 27.09 milliseconds, for $n = 2$, 169.88 ms, for $n = 3$, 582.45 ms. Oflazer's experiments were made on a SPARCstation 10/41. Since we used for our test series a faster machine, yet also much larger dictionaries, an exact comparison of both approaches is impossible. We think, however, that our results show that our method is clearly superior in terms of efficiency.

Table 13 Results for GL, standard Levenshtein distance, bounds $n = 1, 2, 3$, times in milliseconds

Length	$n = 1$ time	NC	$n = 2$ time	NC	$n = 3$ time	NC
5	1.33	4.41	41.5	51.3	313	434
6	1.44	3.40	42.5	34.7	321	337
7	1.59	2.95	39.9	21.9	307	216
8	1.63	2.71	38.7	13.1	307	120
9	1.66	2.48	40.4	9.86	306	80.2
10	1.73	2.32	39.0	7.46	288	50.3
11	1.77	2.14	39.5	6.15	290	36.7
12	1.82	2.01	39.1	4.67	288	23.8
13	1.85	1.89	39.8	4.34	293	19.5
14	1.89	1.80	40.2	3.72	296	14.2
15	1.92	1.71	40.2	3.17	295	10.7
16	1.95	1.65	40.0	2.82	291	7.77
17	1.99	1.60	38.9	2.52	285	6.22
18	2.02	1.56	38.3	2.37	281	5.36
19	2.04	1.34	37.8	1.89	274	3.77

8 Conclusion

We introduced two related methods for correcting garbled words using an electronic dictionary that is implemented as a deterministic finite state automaton. The correction procedures are similar to Oflazer's approach [Ofl96], but completely avoid the computation of Levenshtein distances. Instead, Levenshtein automata for the input words are used to control lexical search. We have shown that appropriate deterministic Levenshtein automata can be computed in time linear to the length of the input. Our second method shows that even the actual computation of a deterministic Levenshtein automaton for the input word can be avoided since precompiled tables may be used to simulate transitions in the automaton. The experimental results show that our techniques lead to a very fast selection of correction candidates for garbled words.

The complexity results for computing the (minimal) deterministic Levensthein automaton for a given input word immediately lead to the following (known) side results.

Lemma 6 *For any fixed number n, given two words W and V of length w and v respectively, it is decidable in time $O(\max(w, v))$ if the Levenshtein distance between W and V is $\leq n$.*

Lemma 7 *For any fixed number n, given a text (sequence of words) of length h and a word W of length w we can compute in time $O(\max(h, w))$ all words V of the text where the Levenshtein distance between V and W does not exceed n.*

The results obtained in this paper could be extended in several directions. The situation could be considered where edit operations come with specific costs that depend on the symbols of the operation. Eventually, in application scenarios different methods for ranking correction candidates could be tested that take the frequency of occurrences of a given correction candidate into account.

The problem considered in this paper is sometimes called the "n-differences" problem. See, e.g., [GP90]

and [WM92] for other approaches. As to more closely related work, two other contributions, both using methods from automata theory for string correction should be mentioned. Bunke 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 [Bun93]. For any word V the automaton may be used to compute the Levenshtein distance between V and W in time linear to the length $|V|$ of V. Given a dictionary of words $W_1, \ldots, W_d$, 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 approach can only be used for very small dictionaries.

Another interesting approach is described in [CSY99]. Recall that we assign to each input word a Levenshtein automaton and leave the dictionary automaton unmodified. In [CSY99] a construction is given for computing, given a finite state automaton A, a lifted version $A^{[n]}$ that accepts all words V that have Levenshtein distance $\leq n$ to some word accepted by A.[4] In principle, this construction can be used to lift a dictionary automaton A in order to compute a *correction transducer* $A^{[n]}$ that yields, given input V, all dictionary words with Levenshtein distance $\leq n$ to V. Assuming that $A^{[n]}$ is deterministic, a run – hence correction of an input word – does not involve any search, or backtracking. However, determinization of a non-deterministic correction transducer is likely to be too space-comsuming for large dictionaries. Nevertheless, it seems promising to consider variants of the techniques described in [CSY99] for lexical correction.

9 Appendix

We here prove Theorem 1. We proceed in several steps.

Lemma 8 *$LEV_n(W)$ is a deterministic finite state automaton.*

Proof It follows from Lemma 4 that Δ assigns to each state $M \in Q$ and each $y \in \Sigma$ again a state $M \in Q$. In fact, a state with base position $i^{\sharp 0}$ $(i < w)$ is always mapped to a state with base position $(i+1)^{\sharp 0}$, and states with base position $w^{\sharp 0}$ are mapped to states with base position $w^{\sharp 0}$. This shows that $LEV_n(W)$ is a deterministic finite state automaton. □

We now show how the transition functions for Levenshtein automata of different degrees for the same input word W are related. We write $\Delta^{(n)}$ for the transition function of the Levenshtein automaton of degree n.

[4] This description is simplified. In [CSY99] distinct metrics for defining neighbourhoods are considered, and a generalization of finite state automata, called *lexical analyzers*, is used.

Lemma 9 (Raising Lemma for transitions) *Let $n > 0$ and $1 \leq e \leq n$. Then for any state of degree n of the form $[M]^{\sharp e}$ and any $x \in \Sigma$ we have*

$$\Delta^{(n)}([M]^{\sharp e}, x) = [\Delta^{(n-e)}(M, x)]^{\sharp e}.$$

Proof Using our earlier notation and Lemma 5 we obtain

$$\begin{aligned}\Delta^{(n)}([M]^{\sharp e}, x) &= \bigsqcup_{\pi \in M} \delta^{(n)}([\pi]^{\sharp e}, x) \\ &= \bigsqcup_{\pi \in M} [\delta^{(n-e)}(\pi, x)]^{\sharp e} \\ &= [\bigsqcup_{\pi \in M} \delta^{(n-e)}(\pi, x)]^{\sharp e} \\ &= [\Delta^{(n-e)}(M, x)]^{\sharp e}\end{aligned}$$

The result follows. □

In the sequel, let $LEV_n(W) = \langle \Sigma, Q, q_0, F, \Delta \rangle$ as in Definition 14.

Proposition 1 *The following properties hold:*

1. $\mathcal{L}(\emptyset) = \emptyset$.
2. *For all states M, N with a common base position and all $y \in \Sigma$:*

$$\Delta(M \sqcup N, y) = \Delta(M, y) \sqcup \Delta(N, y).$$

3. *For all states M, N with a common base position and all $V \in \Sigma^*$:*

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

4. *For all states $M \subseteq Q \setminus \{\{0^{\sharp 0}\}, \ldots, \{w^{\sharp 0}\}\}$: $\mathcal{L}(M) = \bigcup_{\pi \in M} \mathcal{L}(\{\pi\})$.*

Proof Property 1 is trivial.

Proof of Property 2. Since M and N have a common base position it follows that $M \sqcup N$ is again a state. 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 Property 3. Follows from Property 2 by a trivial induction on the length of V.

Proof of Property 4. Let RA denote the set of all raised accepting positions. Using Property 3 we obtain

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

To see the marked equivalence notice that all positions in $\bigcup_{\pi \in M} \Delta^*(\{\pi\}, V)$ are raised. Each position of this set that subsumes a raised accepting position is itself a raised accepting position. □

Proposition 2 *For all $0 \le i \le w$ and all $0 \le e \le n$ we have*

$$\mathcal{L}(\{i^{\sharp e}\}) = \mathcal{L}_{\mathrm{Lev}}(n - e, x_{i+1} \cdots x_w).$$

Proof We proceed by induction on n. The case $n = 0$ is simple: the set of positions is $\{i^{\sharp 0} \mid 0 \le i \le w\}$. Only $w^{\sharp 0}$ is an accepting position. States have the form $\emptyset$ or $\{i^{\sharp 0}\}$ $(0 \le i \le w)$. Transitions from states $\{i^{\sharp 0}\}$ can be considered as elementary transitions from positions $i^{\sharp 0}$. By Property 1 of Proposition 1, $\mathcal{L}(\emptyset) = \emptyset$. Part II of Table 2 shows that only transitions leading from states $\{i^{\sharp 0}\}$ for $i < w$ with input x_{i+1} to $\{(i+1)^{\sharp 0}\}$ are relevant – all other transitions lead to the failure state $\emptyset$. Hence $\mathcal{L}(\{i^{\sharp 0}\}) = \{x_{i+1} \cdots x_w\} = \mathcal{L}_{\mathrm{Lev}}(0, x_{i+1} \cdots x_w)$ for all $0 \le i \le w$.

Now let $n \ge 1$ and assume that the proposition is correct for all $0 \le n' < n$. The case $e = n$ is simple since all relevant transitions are of the form $\{i^{\sharp n}\} \mapsto \{(i+1)^{\sharp n}\}$ under x_{i+1} $(i < w)$. Hence assume that $e < n$. Since for $1 \le e < n$ transitions from states $\{i^{\sharp e}\}$ are defined by raising of transitions of degree $n' = n - e$ (cf. Lemma 9), the induction hypothesis shows that

$$\begin{aligned}
\mathcal{L}(\{i^{\sharp e}\}) &= \mathcal{L}_{\mathrm{Lev}}(n - e, x_{i+1} \cdots x_w) \\
&(0 \le i \le w, 1 \le e \le n). \qquad (\dagger)
\end{aligned}$$

Hence it only remains to prove that for all $0 \le i \le w$ we have

$$\mathcal{L}(\{i^{\sharp 0}\}) = \mathcal{L}_{\mathrm{Lev}}(n, x_{i+1} \cdots x_w).$$

In the sequel, let W_i denote the suffix $x_{i+1} \cdots x_w$ of W $(0 \le i \le w)$. From (†) and Lemma 2 we obtain the following: for all positions $i^{\sharp e}$ and $j^{\sharp f}$ such that $e \ne 0 \ne f$, if $i^{\sharp e}$ is subsumed by $j^{\sharp f}$, then $\mathcal{L}(\{i^{\sharp e}\})$ is a proper subset of $\mathcal{L}(\{j^{\sharp f}\})$ (††).

I. We first show that $\mathcal{L}_{\mathrm{Lev}}(n, W_i) \subseteq \mathcal{L}(\{i^{\sharp 0}\})$. Let $V \in \mathcal{L}_{\mathrm{Lev}}(n, W_i)$.

Case 1.1: V is obtained from W_i by deleting a suffix of length k $(0 \le k \le n)$ of W_i. Starting from state $\{i^{\sharp 0}\}$ and consuming V we reach state $\{(w-k)^{\sharp 0}\}$. Since $(w-k)^{\sharp 0}$ is an accepting position, state $\{(w-k)^{\sharp 0}\}$ is final, hence $V \in \mathcal{L}(\{i^{\sharp 0}\})$.

Case 1.2: W_i is obtained from V by deleting a suffix of length k $(1 \le k \le n)$ of V. Starting from state $\{i^{\sharp 0}\}$ and first consuming W_i we reach $\{w^{\sharp 0}\}$. The k additional transitions lead to $\{w^{\sharp k}\}$. Since $w^{\sharp k}$ is an accepting position, the latter state is final, hence $V \in \mathcal{L}(\{i^{\sharp 0}\})$.

Case 2: In the remaining cases there exists an index $j < w$ such that $V = x_{i+1} \cdots x_j y V'$ where $y \ne x_{j+1}$.

We have $yV' \in \mathcal{L}_{\mathrm{Lev}}(n, x_{j+1} \cdots x_w)$ by Lemma 1. We fix a sequence ν of edit operations leading from $x_{j+1} \cdots x_w$ to yV' of minimal length and consider the three cases described in Remark 1.

2.1. If the occurrence of y is an insertion before x_{j+1} (where $i \le j \le w$), then $V' \in \mathcal{L}_{\mathrm{Lev}}(n-1, x_{j+1} \cdots x_w)$. Since $y \ne x_{j+1}$, starting from state $\{i^{\sharp 0}\}$ and consuming the letters $x_{i+1}, \ldots x_j, y$ we reach states $\{(i+1)^{\sharp 0}\}, \ldots, \{j^{\sharp 0}\}, M$ where M contains $j^{\sharp 1}$ (cf. elementary transitions). It follows from (†) and Property 4 of Proposition 1 that $V' \in \mathcal{L}(M)$. Hence $V \in \mathcal{L}(\{i^{\sharp 0}\})$.

2.2. If the occurrence of y substitutes x_{j+1} (where $i \le j < w$), then V' belongs to $\mathcal{L}_{\mathrm{Lev}}(n-1, x_{j+2} \cdots x_w)$. Starting from state $\{i^{\sharp 0}\}$ and consuming the letters $x_{i+1}, \ldots x_j, y$ we reach states $\{(i+1)^{\sharp 0}\}, \ldots, \{j^{\sharp 0}\}, M$ where M contains $(j+1)^{\sharp 1}$. It follows from (†) and Property 4 of Proposition 1 that $V' \in \mathcal{L}(M)$. Hence $V \in \mathcal{L}(\{i^{\sharp 0}\})$.

2.3. In the remaining cases, by Remark 1 there exists some $1 \le k \le n$ such that we have a stroke from x_{j+k+1} to y in the trace representation of ν, as we see in Fig. 11. This means that the k letters $x_{j+1}, x_{j+2}, \ldots, x_{j+k}$ are erased. The distance between $x_{j+k+1} \cdots x_w$ and yV' is bounded by $n - k$. In the sequel, let k_0 be the smallest index in $\{1, \ldots, k+1\}$ such that $x_{j+k+1} = x_{j+k_0}$. It follows from the definition of elementary transitions (cf. Table 2) that we reach state $M := \{j^{\sharp 1}, (j+1)^{\sharp 1}, (j + k_0)^{\sharp k_0 - 1}\}$ from $\{j^{\sharp 0}\}$ by consuming $x_{j+k_0} = x_{j+k+1}$.

2.3.1. Assume first that $x_{j+k+1} = y$. Then the distance between $x_{j+k+2} \cdots x_w$ and V' is bounded by $n-k$. By (†), $V' \in \mathcal{L}(\{(j+k+1)^{\sharp k}\})$. Since $(j+k+1)^{\sharp k}$ is subsumed by $(j+k_0)^{\sharp k_0 - 1}$, also $V' \in \mathcal{L}(\{(j+k_0)^{\sharp k_0 - 1}\})$ by (††), and $V' \in \mathcal{L}(M)$ by Property 4 of Proposition 1. It follows that $V \in \mathcal{L}(\{i^{\sharp 0}\})$.

2.3.2. Assume that $x_{j+k+1} \ne y$. Then the distance between $x_{j+k+2} \cdots x_w$ and V' is bounded by $n - k - 1$. Starting from state $\{i^{\sharp 0}\}$ and consuming the letters $x_{i+1}, \ldots x_j, y$ we eventually reach a state M containing $(j+1)^{\sharp 1}$. This position subsumes $(j+k+1)^{\sharp k+1}$. It follows from (†), (††) and Property 4 of Proposition 1 that $V' \in \mathcal{L}(M)$. Hence $V \in \mathcal{L}(\{i^{\sharp 0}\})$.

II. It remains to prove that $\mathcal{L}(\{i^{\sharp 0}\}) \subseteq \mathcal{L}_{\mathrm{Lev}}(n, W_i)$ for $0 \le i \le w$. Let $V \in \mathcal{L}(\{i^{\sharp 0}\})$. If V is accepted – start-

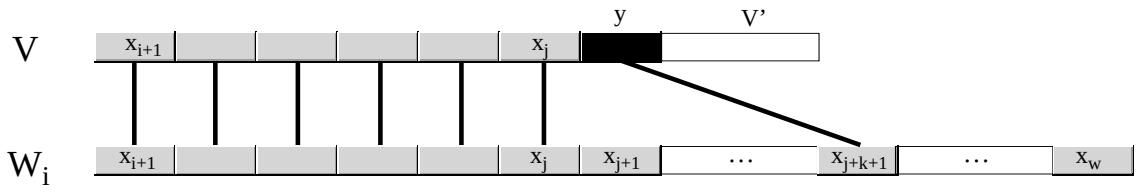

Fig. 11 Illustration for Subcase 2.3

ing from $\{i^{\sharp 0}\}$ – on a path of singleton sets with basic positions $\{(i+1)^{\sharp 0}\}, \{(i+2)^{\sharp 0}\}, \ldots, \{k^{\sharp 0}\}$, then $k^{\sharp 0}$ is accepting, which implies that V has the form $x_{i+1} \cdots x_k$ where $w - k \leq n$. This shows that $V \in \mathcal{L}_{\mathrm{Lev}}(n, W_i)$. In the other case, starting from state $\{i^{\sharp 0}\}$ and consuming the prefix $V'y$ of $V = V'yV''$, we reach states $\{(i+1)^{\sharp 0}\}, \ldots \{j^{\sharp 0}\}, M$ where $M \neq \{(j+1)^{\sharp 0}\}$.

Case (a): $j < w$ and M has the form $\{j^{\sharp 1}, (j+1)^{\sharp 1}\}$. In this case, by (†) and Property 4 of Proposition 1, either V'' has distance $\leq n-1$ to $x_{j+1} \cdots x_w$ or V'' has distance $\leq n-1$ to $x_{j+2} \cdots x_w$. In the former case, with an additional insertion of y we see that V has distance $\leq n$ to W_i. In the latter case, using a substitution $x_{j+1} \mapsto y$ we see V has distance $\leq n$ to W_i.

Case (b): $j < w$ and M has the form $\{j^{\sharp 1}, (j+1)^{\sharp 1}, (j+k)^{\sharp k-1}\}$. Here we have to consider the additional case where V'' has distance $\leq n - (k-1)$ to $x_{j+k+1} \cdots x_w$. However, we know that $y = x_{j+k}$. Deleting $x_{j+1}, \ldots, x_{j+k-1}$, we see that yV' has distance $\leq n$ to $x_{j+1} \ldots x_w$, hence the same holds for V and W_i.

Case (c): $j = w$. In this case $M = \{w^{\sharp 1}\}$ and $V' = W_i$. It follows from (†) that V'' has distance $\leq n-1$ to the empty word ε. Hence V has distance $\leq n$ to W_i. □

Theorem 4 *$LEV_n(W)$ is a deterministic and acyclic Levenshtein automaton of degree n for W. For fixed degree n, the size of $LEV_n(W)$ is linear in $|W|$.*

Proof Proposition 2 shows that

$$\mathcal{L}(LEV_n(W)) = \mathcal{L}(\{0\}) = \mathcal{L}_{\mathrm{Lev}}(n, W),$$

hence $LEV_n(W)$ is a deterministic Levenshtein automaton of degree n for W. If $j^{\sharp f}$ is in the image set of position $i^{\sharp e}$, then $i + e < j + f$. Hence it is easy to see that $LEV_n(W)$ is acyclic. Obviously the number of possible base positions for states is linear in $|W|$, and for fixed degree n there exists a uniform bound on the number of distinct states with a fixed base position $i^{\sharp 0}$. It follows that the number of states of $LEV_n(W)$ is linear in $|W|$. Since the alphabet Σ is fixed, the size of $LEV_n(W)$ is linear in $|W|$. □

References

[AFW83] Angell RC, Freund GE, Willett P (1983) Automatic spelling correction using a trigram similarity measure. Inf Process Manage 19:255–261

[Bla60] Blair CR (1960) A program for correcting spelling errors. Inf Control 3:60–67

[Bun93] Bunke H (1993) A fast algorithm for finding the nearest neighbor of a word in a dictionary. In: Proceedings of 2nd International Conference on Document Analysis and Recognition ICDAR '93, pp 632–637. IEEE Computer Society Press

[CSY99] Calude CS, Salomaa K, Yu S (1999) Metric lexical analysis. In: 4th International Workshop on Implementing Automata, WIA'99. Lecture Notes in Computer Science, vol 2214. Springer, Berlin Heidelberg New York

[dBdBT95] de Bertrand de Beuvron F, Trigano P (1995) Hierarchically coded lexicon with variants. Int J Pattern Recogn Artif Intell 9(1):145–165

[DHH+97] Dengel A, Hoch R, Hönes F, Jäger T, Malburg M, Weigel A (1997) Techniques for improving OCR results. In: Bunke H, Wang PSP (eds) Handbook of Character Recognition and Document Image Analysis. World Scientific, Singapore

[DMWW00] Daciuk J, Mihov S, Watson B, Watson R (2000) Incremental construction of minimal acyclic finite state automata. Comput Linguist 26(1):3–16

[GP90] Galil Z, Park K (1990) An improved algorithm for approximate string matching. SIAM J Comput 19(6):989–999

[Hon95] Hong T (1995) Degraded text recognition using visual and linguistic context. PhD thesis. CEDAR, State University of New York at Buffalo

[HU79] Hopcroft JE, Ullman JD (1979) Introduction to automata theory, languages, and computation. Addison-Wesley, Reading, Mass.

[Hul92] Hull JJ (1992) A hidden Markov model for language syntax in text recognition. In: Proceedings of the 11th IAPR International Conference on Pattern Recognition, The Hague, The Netherlands. IEEE Computer Society Press, pp 416–423

[KEW91] Keenan FG, Evett LJ, Withrow RJ (1991) A large vocabulary stochastic analyser for handwriting recognition. In: Proceedings of the First International Conference on Document Analysis and Recognition (ICDAR '91), pp 794–802. IEEE Computer Society Press

[Koz97] Kozen DC (1997) Automata and computability. Springer, New York Berlin Heidelberg

[KST92] Kim JY, Shawe-Taylor J (1992) An approximate string-matching algorithm. Theor Comput Sci 92:107–117

[KST94] Kim JY, Shawe-Taylor J (1994) Fast string matching using an n-gram algorithm. Softw Pract Exper 94(1):79–88

[Kuk92] Kukich K (1992) Techniques for automatically correcting words in texts. ACM Comput Surv 24(4):377–439

[Lev66] Levenshtein V (1996) Binary codes capable of correcting deletions, insertions, and reversals. Sov Phys Dokl 10(8):707–710

[Mih98] Mihov S (1998) Direct building of minimal automaton for given list. In: Annuaire de l'Université de Sofia "St. Kl. Ohridski", vol 91(1)

[Ofl96] Oflazer K (1996) Error-tolerant finite-state recognition with applications to morphological analysis and spelling correction. Comput Linguist 22(1):73–89

[OL97] Oommen BJ, Loke RKS (1997) Pattern recognition of strings with substitutions, insertions, deletions, and generalized transpositions. Pattern Recogn 30(5):789–800

[OM88] Owolabi O, McGregor DR (1988) Fast approximate string matching. Softw Pract Exper 18(4):387–393

[RE71] Riseman EM, Ehrich RW (1971) Contextual word recognition using binary digrams. IEEE Trans Comput C-20(4):397–403

[Rev92] Revuz D (1992) Minimisation of acyclic deterministic automata in linear time. Theor Comput Sci 92(1):181–189

[SHC83] Srihari SN, Hull JJ, Choudhari R (1983) Integrating diverse knowledge sources in text recognition. ACM Trans Office Inf Syst 1(1):68–87

[Sin90] Sinha RMK (1990) On partitioning a dictionary for visual text recognition. Pattern Recogn 23(5):497–500

[SKS96] Seni G, Kripasundar V, Srihari RK (1996) Generalizing edit distance to incorporate domain information: handwritten text recognition as a case study. Pattern Recogn 29(3):405–414

[SM01] Schulz KU, Mihov S (2001) Fast string correction with Levenshtein-automata. CIS-Report 01-127, CIS, Universität München

[Sri85] Srihari SN (1985) Computer text recognition and error correction. Tutorial. IEEE Computer Society Press, Silver Spring, Md.

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

[Ukk85] Ukkonen E (1985) Algorithms for approximate string matching. Inf Control 64:100–118

[Ukk92] Ukkonen E (1992) Approximate string-matching with q-grams and maximal matches. Theor Comput Sci 92:191–211

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

[WBR95] Weigel F, Baumann S, Rohrschneider J (1995) Lexical postprocessing by heuristic search and automatic determination of the edit costs. In: Proceedings of the Third International Conference on Document Analysis and Recognition (ICDAR '95), pp 857–860. IEEE Computer Society Press

[WF74] Wagner RA, Fisher M (1974) The string-to-string correction problem. J ACM 21(1):168–173

[WM92] Wu S, Manber U (1992) Fast text searching allowing errors. Commun ACM 35(10):83–91

[ZD95] Zobel J, Dart P (1995) Finding approximate matches in large lexicons. Softw Pract Exper 25(3):331–345

K.U. Schulz obtained his diploma in mathematics in 1984. In the period until 1986 he finished his PhD in mathematics and started working in the area of natural language processing. From autumn 1987 until spring 1988 Professor Schulz was a guest professor at the University of Niterói, Rio de Janeiro. In his habilitation, which was completed in 1991, K.U. Schulz presented various simplifications and extensions of Makanin's famous decidability result for the solvability of word equations. In 1991 he was appointed professor of computational linguistics and information processing at the University of Munich. Since 1992 he has been a director of the Center for Information and Language Processing (CIS) of the University of Munich. His research interests are mainly centered on the areas of information retrieval, text analysis and document processing. A special emphasis is on retrieval systems for structured documents and semistructured data, postprocessing of OCR results, and document contents analysis. A second line of research, more prominent in the former work of Professor Schulz, addresses theoretical questions in the area of automated deduction.

Stoyan Mihov obtained his diploma in mathematics in 1993. He completed his PhD in mathematics and computer science in 2000. In his PhD, Mihov presented various constructions and algorithms for direct building of minimal acyclic finite state devices. Since 1994 he has worked at the Linguistic Modelling Department of Bulgarian Academy of Science (CLPP-BAS). Since 1996 he has had a research position at the Bulgarian Academy of Sciences. His scientific interests concern finite state automata theory and natural language processing. Recently, his main emphasis has been on application of finite state methods for information extraction and document analysis.