Approximate word matching with synchronized rational relations

Petar Mitankin
Institute for Parallel Processing, Bulgarian Academy of Sciences
petar@lml.bas.bg

Abstract

Algorithms that use approximate word matching are widely used in different areas. We present the work in progress on a method that has the potential to select extremely fast from a large dictionary of correct words a small set of words that are proximate to a given input erroneous word. The method can be applied for a large class of distances based on weighted edit operations. By given distance, we build a two-tape transducer whose language is a rational relation with bounded length difference. In this paper we present an algorithm for synchronizing a rational relation with bounded length difference. We show how our method could be applied for automatic correction of OCR-ed text.

Keywords

String correction, OCR-correction, approximate string matching, Levenshtein distance, regular relations, finite state transducers, finite state automata.

1 Introduction

The problem of choosing suitable distance between finite words is crucial for many applications. One basic approach to get a measure for proximity of words is to determine a set of *primitive edit operations*. Then the distance between the strings w and v is the minimal number of primitive edit operations that transform w into v. For example the edit operations *substitution* (replacement of one letter with another), *deletion* of a letter and *insertion* of a letter give the so-called *Levenshtein distance* [8], known also as *edit distance*. To refine the distance we can attach to every edit operation a weight (cost) [10, 18, 19, 7]. Then the distance between w and v can be defined as the minimal cost that is necessary to get v from w applying edit operations.

For example, for automatic correction of a text recognized by an OCR system, it is appropriate the primitive edit operations to reflect the possible errors done by the OCR system and the weights of the operations to reflect the frequencies of the errors. For instance, if the OCR system often recognizes *in* as *m*, but never recognizes *in* as *t*, in order to get an appropriate measure for correction we have to assign a high weight to the *split* $m \mapsto in$ and not to consider the *split* $t \mapsto in$ as a primitive edit operation.

The other problem that often must be solved is how to extract efficiently from a large dictionary of correct words a small set of correction candidates that almost always contains the right correction word. Usually this is done in two steps. On the first step a big set of correct candidates is extracted from the dictionary. On the second step the selected candidates are ranked in order to find the most appropriate ones among them [6, 11, 20, 16]. In our previuos work [15] we show a new approach for directly selecting the best candidates from the dictionary. Having an appropriate distance d, the erroneous word w over a finite alphabet Σ and the dictionary D, the desired small set can be generated as the intersection of D and the set of the words v such that $d(w, v)$ is not greater than n and n is an edit bound that is fixed in advance.

One possible solution is to build by the erroneous word w a deterministic finite automaton $A_n(w)$ with language $L(A_n(w)) = \{v \mid d(w, v) \leq n\}$ and if the dictionary is represented also as a deterministic finite automaton A_{dict} to generate the intersection via a parallel traversal of both of $A_n(w)$ and A_{dict} in depth [14]. Here we suppose that at least one of the two automata has a finite language, so the intersection can be generated with such a traversal. The disadvantage of this method is that $A_n(w)$ depends on w and for every erroneous word w we have to build $A_n(w)$. Instead of building $A_n(w)$ for every w, we can use the so-called *universal Levenshtein automaton*, that has a special input alphabet of bitvectors, but does not depend on a concrete word w [9].

In our previous work [15] we show that a variant of the *universal Levenshtein automata* can be used very successfully for a solution of the problem mentioned above. The input alphabet of these universal automata consists of tuples of bitvectors. By given two words w and v we first generate a sequence of tuples of bitvectors $i(w, v)$ which is the input word for a universal automaton $A_n^\forall$. The automaton $A_n^\forall$ accepts $i(w, v)$ if and only if $d(w, v) \leq n$. We use a parallel traversal of $A_n^\forall$ and A_{dict} in depth to generate the intersection. By the erroneous word w and the label (the letter) of the current transition of A_{dict} we compute the tuple of bitvectors with which we have to continue the traversal of $A_n^\forall$. These universal automata have another advantage: they do not depend on the concrete primitive edit operations but only on their *type*. Here by *type* of a primitive edit operation we mean a couple of natural numbers $\langle k, m\rangle$, which indicates that the edit operation replaces a string of k letters with a string of m letters. For example the substitution $a \mapsto b$ has the type $\langle 1, 1\rangle$ (we replace one letter with another), the deletion of a has the type $\langle 1, 0\rangle$ (we re-

place one letter with zero letters), the insertion of a has the type $\langle 0,1\rangle$ (we replace zero letters with one), the split $a \mapsto bc$ has the type $\langle 1,2\rangle$ (we replace one letter with two) etc. Thus for every type $t = \langle k,m\rangle$ we have a set $A_t \subseteq \Sigma^k \times \Sigma^m$ of all edit operations of type $\langle k,m\rangle$ that are allowed to be used for the transformation of w into v. The universal automaton depends only on the edit bound n and the types t, but neither on the concerete sets A_t, nor on a concrete word w. The sets A_t are used for the computation of the tuples of bitvectors.

The so-called *forward-backward* method [9] can be applied to reduce vastly the time needed to find the intersection. In this method we use A_{dict} and A^{rev}_{dict} - a deterministic finite automaton with language consisting of the reversed words from the dictionary D. We split the erroneous word w into w_1 and w_2, $w = w_1w_2$, such that w_1 and w_2 have approximately equal lengths. For simplicity let us consider that d is the usual Levenshtein distance and the edit bound $n = 1$. We traverse A_{dict} with w_1 and after it we traverse parallelly both A_{dict} and $A^{\forall}_1$ considering w_2 as an erroneous word, starting from the inital state of $A^{\forall}_1$ and the state of A_{dict} that we have reached with w_1. In this way we generate $\{w_1v \mid d(w_1w_2, w_1v) \leq 1\}$. We traverse A^{rev}_{dict} with w^{rev}_2 (the reversed w_2) and after it we traverse parallelly both A^{rev}_{dict} and $A^{\forall}_1$ considering w^{rev}_1 as an erroneous word, starting from the inital state of $A^{\forall}_1$ and the state of A^{rev}_{dict} we have reached with w^{rev}_2. In this way we generate $\{vw_2 \mid d(w_1w_2, vw_2) \leq 1\}$. The *forward-backward* method is much faster than the usual parallel traversal, because the branching dergees in the initial parts of the automata A_{dict} and A^{rev}_{dict} are very high especially when the automata represent natural languages.

The disadvantage of the *universal Levenshtein automata* is that we have to compute tuples of bitvectors. In this paper we show how this can be avoided. We use two-tape automata and synchronized rational relations instead of universal automata. The deterministic automata, that we finally get, are also "universal" in the sense that they do not depend on concrete word w. Their great plus is that they do not require computation of tuples of bitvectors. Their minus is that their size is too great and they are not as much universal as the universal automata because they depend strongly on the sets A_t and on the size of the alphabet Σ. So they are applicable in the cases when the set of primitive edit operations, the edit bound n and the alphabet Σ are fixed.

2 Formal background

We assume that the reader is familiar with automata theory [4, 13]. We use ϵ to denote the empty word. If Σ is a finite alphabet, as usual we define $\Sigma^0 := \{\epsilon\}$, $\Sigma^{i+1} := \{wa \mid w \in \Sigma^i \,\&\, a \in \Sigma\}$ for $i \in N$ and $\Sigma^* := \bigcup_{i\in N} \Sigma^i$. $\Sigma^\epsilon := \Sigma \cup \{\epsilon\}$. The length of a word $w \in \Sigma^*$ is denoted by $|w|$.

Definition 2.1 Let $w \in \Sigma^*$, $t_1, t_2 \in N$. Then

$$w(t_1,t_2] := \begin{cases} w_{t_1+1}w_{t_1+2}\ldots w_{t_2} & \text{if } t_1 < t_2 \leq |w| \\ \epsilon & \text{otherwise} \end{cases}$$

Definition 2.2 Let $A = \langle \Sigma, Q, q_0, \Delta, F\rangle$ be a (nondeterministic) finite state automaton with alphabet Σ, set of states Q, initial state q_0, transition relation $\Delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q$ and set of final states F. For $q \in Q$ the set $L(q) := \{v \in \Sigma^* \mid \exists f \in F : \langle q, v, f\rangle \in \Delta^*\}$ is called the *language of state* q. Here Δ^* denotes the extended transition relation, which is defined as usual: Δ^* is the smallest subset of $Q \times \Sigma^* \times Q$ satisfying the following conditions:

1. $\langle q, \epsilon, q\rangle \in \Delta^*$ for all $q \in Q$
2. $\langle q_1, w, q_2\rangle \in \Delta^* \,\&\, \langle q_2, x, q_3\rangle \in \Delta \Rightarrow \langle q_1, wx, q_3\rangle \in \Delta^*$ for all $w \in \Sigma^*$ and all $x \in \Sigma \cup \{\epsilon\}$

The *language of* A is $L(A) := L(q_0)$. A set of words $M \subseteq \Sigma^*$ is called *regular* (*rational*) iff there exists a finite state automaton A such that $L(A) = M$.

Definition 2.3 Let $T = \langle \Sigma, q_0, I, \Delta, F\rangle$ be a two-tape finite state transducer with alphabet Σ for the two tapes, set of states Q, initial state q_0, transition relation $\Delta \subseteq Q \times (\Sigma^\epsilon \times \Sigma^\epsilon) \times Q$ and set of final states F. For $q \in Q$ the set $L(q) := \{\langle w, v\rangle \in \Sigma^* \times \Sigma^* \mid \exists f \in F : \langle q, \langle w, v\rangle, f\rangle \in \Delta^*\}$ is called the *language of state* q. Here Δ^* denotes the extended transition relation, which is defined as usual: Δ^* is the smallest subset of $Q \times (\Sigma^* \times \Sigma^*) \times Q$ satisfying the following conditions:

1. $\langle q, \langle \epsilon, \epsilon\rangle, q\rangle \in \Delta^*$ for all $q \in Q$
2. $\langle q_1, \langle w, v\rangle, q_2\rangle \in \Delta^* \,\&\, \langle q_2, \langle x, y\rangle, q_3\rangle \in \Delta \Rightarrow \langle q_1, \langle wx, vy\rangle, q_3\rangle \in \Delta^*$ for all $w, v \in \Sigma^*$ and all $x, y \in \Sigma \cup \{\epsilon\}$

The *left language of* q is $\overleftarrow{L}(q) := \{\langle w, v\rangle \mid \langle q_0, \langle w, v\rangle, q\rangle \in \Delta^*\}$. The *language of* T is $L(T) := L(q_0)$. A binary relation $R \subseteq \Sigma^* \times \Sigma^*$ is called *regular* (*rational*) iff there exists a finite state transducer T such that $L(T) = R$.

3 Generalized word distances

We define a family of functions that measure the proximity between two words. Each function is based on a set of weighted edit operations.

Definition 3.1 A *weighted edit operation* is a quadruple $op = \langle op^x, op^y, op^r, op^w\rangle$ where op^x, $op^y \in N$, $op^x + op^y > 0$, $op^r \subseteq \Sigma^{op^x} \times \Sigma^{op^y}$ and $op^w \in N$. op^w is called the *weight* of the operation op and op^r is called the *replacement relation* of op.

Definition 3.2 Let $w, v \in \Sigma^*$ and Op be a set of weighted edit operations. We say that v *can be obtained from* w *with cost* c *by applying* k *operations from* Op iff w can be represented in the form $w = w_{(1)}w_{(2)}\ldots w_{(k)}$, v can be represented in the form $v = v_{(1)}v_{(2)}\ldots v_{(k)}$, such that each pair $\langle w_{(i)}, v_{(i)}\rangle$ belongs to some relation op^r_i for some $op_i \in Op$ $(1 \leq i \leq k)$ and the cost c is given by $op^w_1 + \ldots + op^w_k$.

Definition 3.3 Given a set of operations Op we define the function $d[Op] : \Sigma^* \times \Sigma^* \to N \cup \{\infty\}$. $d[Op](w, v)$ is the minimal cost to obtain v from w by applying operations from Op. If v cannot be obtained

from w by applying operations from Op $d[Op](w,v)$ equals the special value ∞ ($i < \infty$, $\infty + i := i + \infty := \infty$ for all $i \in N$, $\infty + \infty := \infty$).

Using a variant of the dynamic programming scheme described in [17] we can define the function $d[Op]$ in another way:

Definition 3.4 Let $w, v \in \Sigma^*$ be given. In order to define $d[Op](w,v) \in N \cup \{\infty\}$ we inductively define a $(|v|+1) \times (|w|+1)$ matrix M with entries in $N \cup \{\infty\}$:

1. $M_{0,0} := 0$
2. Let $i \neq 0$ or $j \neq 0$. Assume that $M_{i',j'}$ is defined for all $i' \leq i$ and all $j' \leq j$ such that $i'+j' < i+j$.
   (a) If, for some $op \in Op$, we have $\langle w(j - op^x, j], v(i - op^y, i] \rangle \in op^r$, let $M_{i,j} := min_{op \in Op} M^{op}[i,j]$ where $M^{op}[i,j] := op^w + M_{i-op^y, j-op^x}$
   (b) If there does not exist $op \in Op$ such that $\langle w(j - op^x, j], v(i - op^y, i] \rangle \in op^r$, let $M_{i,j} := \infty$.

Eventually we define $d[Op](w,v) := M_{|v|,|w|}$.

Proposition 3.5 *Defintions 3.3 and 3.4 are equivalent.*

Definition 3.6 Let Op be a set of weighted edit operations. Op is called *normal* iff Op is finite and the identity $op_{id} := \langle 1, 1, \{\langle a, a \rangle \mid a \in \Sigma\}, 0 \rangle$ belongs to Op.

In what follows, each set of weighted edit operations is assumed to be normal. This requirement is natural because it guarantees that $d[Op](w,w) = 0$ for each word $w \in \Sigma^*$.

Example 3.7 Let $Op := \{op_{id}, op_{del}, op_{ins}, op_{subs}\}$ where

$$\begin{aligned} op_{del} &= \langle 1, 0, \{\langle a, \epsilon \rangle \mid a \in \Sigma\}, 1 \rangle, \\ op_{ins} &= \langle 0, 1, \{\langle \epsilon, a \rangle \mid a \in \Sigma\}, 1 \rangle \text{ and} \\ op_{subs} &= \langle 1, 1, \{\langle a, b \rangle \mid a, b \in \Sigma\}, 1 \rangle. \end{aligned}$$

Then $d[Op]$ is the usual *Levenshtein distance* [8].

Example 3.8 Let $Op := \{op_{id}, op_{del}, op_{ins}, op_{subs}, op_{tr}\}$ where op_{del}, op_{ins}, op_{subs} are as above and $op_{tr} := \langle 2, 2, \{\langle ab, ba \rangle \mid a, b \in \Sigma\}, 1 \rangle$. Then $d[Op]$ is the *generalized Levenshtein distance with transposition.*

Example 3.9 Let $Op := \{op_{id}, op_{del}, op_{ins}, op_{subs}, op_{merge}, op_{split}\}$ where op_{del}, op_{ins}, op_{subs} are as above and

$$\begin{aligned} op_{merge} &= \langle 2, 1, \{\langle ab, c \rangle \mid a, b, c \in \Sigma\}, 1 \rangle \text{ and} \\ op_{split} &= \langle 1, 2, \{\langle c, ab \rangle \mid a, b, c \in \Sigma\}, 1 \rangle. \end{aligned}$$

Then $d[Op]$ is the *generalized Levenshtein distance with merge and split.*

Example 3.10 Let $S \subseteq \Sigma \times \Sigma$ and $Op = \{op_{id}, op_{del}, op_{ins}, op_{S-subs}\}$ where

$$op_{S-subs} = \langle 1, 1, S, 1 \rangle.$$

Then $d[Op]$ is the *generalized Levenshtein distance with S-restricted substitutions.* If $S = \Sigma \times \Sigma$, then $d[Op]$ is the usual Levenshtein distance defined in Example 3.7.

Example 3.11 Let $S \subseteq \Sigma \times \Sigma$, $M \subseteq \Sigma^2 \times \Sigma$ and $Sp \subseteq \Sigma \times \Sigma^2$. Let $Op := \{op_{id}, op_{del}, op_{ins}, op_{S-subs}, op_{M-merge}, op_{Sp-split}\}$, where

$$\begin{aligned} op_{M-merge} &= \langle 2, 1, M, 1 \rangle \text{ and} \\ op_{Sp-split} &= \langle 1, 2, Sp, 1 \rangle. \end{aligned}$$

Then $d[Op]$ is the *generalized Levenshtein distance with S-restricted substitutions, M-restricted merges and Sp-restricted splits.*

Let us note that the distance $d[Op]$ is not always a measure. For example the generalized Levenshtein distance with S-restricted substitutions (Example 3.10) is symmetric only when the relation S is symmetric. The Levenshtein distance extended with transpositions (Example 3.8), d^t_L, is symmetric, but not a measure, because the triangle inequality does not hold always: Let $\Sigma = \{a, b, c, d\}$. Then $d^t_L(abcd, abdc) = 1$, $d^t_L(abdc, bdac) = 2$, but $d^t_L(abcd, bdac) = 4$.

4 Two tape transducers for bounded word distances and synchronized rational relations

For a normal set of operations Op and an *edit bound* $n \in N$ we define a two-tape transducer $T_n[Op]$ such that $L(T_n[Op]) = \{\langle w, v \rangle | d[Op](w,v) \leq n\}$. In what follows, let $Op' := \{op \in Op \mid max(op^x, op^y) \geq 2\}$.

Definition 4.1 $T_n[Op] := \langle \Sigma, Q, 0, \Delta, F \rangle$ where

1. the alphabet of the two tapes is Σ,
2. the set of states is $Q := Q' \cup Q''$ where
   - $Q' := \{k \in N | k \leq n\}$ (main states),
   - $Q'' := \{k_{op,\alpha,\beta,j} \mid k + op^w \in Q' \;\&\; op \in Op' \;\&\; 1 \leq j < max(op^x, op^y) \;\&\; \langle \alpha, \beta \rangle \in op^r\}$ (intermediate states),
3. the initial state is 0,
4. the transition relation Δ is the following subset of $Q \times (\Sigma^\epsilon \times \Sigma^\epsilon) \times Q$: Let $\pi_1, \pi_2 \in Q$ and $\langle a, b \rangle \in \Sigma^\epsilon \times \Sigma^\epsilon$. Then $\langle \pi_1, \langle a, b \rangle, \pi_2 \rangle \in \Delta$ iff one of the following conditions is satisfied:
   (a) π_1 has the form k and π_2 has the form $k + op^w$ where $\langle a, b \rangle \in op^r$ for some operation $op \in Op$ such that $op \notin Q'$,

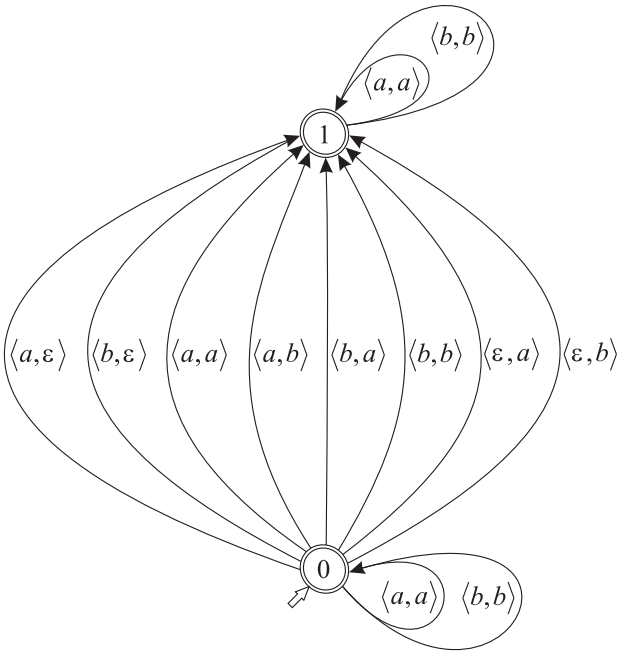

Fig. 1: *Two-tape transducer $T_1[\{op_{id}, op_{del}, op_{ins}, op_{subs}\}]$ for the usual Levenshtein distance, $\Sigma = \{a, b\}$*

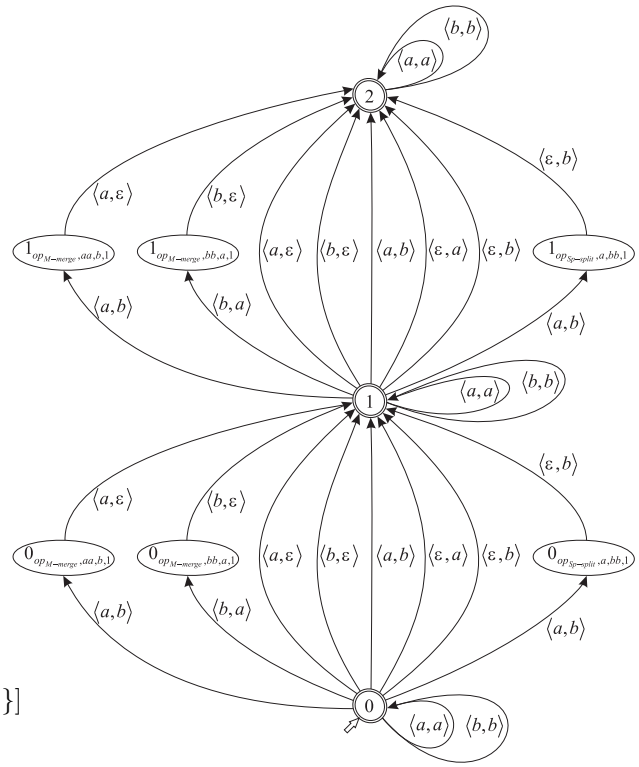

Fig. 2: *Two-tape transducer for generalized Levenshtein distance with restricted substitutions, merges and splits. The edit bound is $n = 2$.*

(b) π_1 has the form k, π_2 has the form $k_{op,\alpha,\beta,1}$, $a = \alpha_1$ if $op^x > 0$, $a = \epsilon$ if $op^x = 0$, $b = \beta_1$ if $op^y > 0$, $b = \epsilon$ if $op^y = 0$, for some operation $op \in Op'$

(c) π_1 has the form $k_{op,\alpha,\beta,j}$, π_2 has the form $k_{op,\alpha,\beta,j+1}$, $a = \alpha_{j+1}$ if $op^x \geq j+1$, $a = \epsilon$ if $op^x < j+1$, $b = \beta_{j+1}$ if $op^y \geq j+1$, $b = \epsilon$ if $op^y < j+1$, for some operation $op \in Op'$ and some $j < max(op^x, op^y) - 1$

(d) π_1 has the form $k_{op,\alpha,\beta,m}$ for $m = max(op^x, op^y) - 1$, π_2 has the form $k + op^w$, $a = \alpha_{m+1}$ if $op^x \geq m+1$, $a = \epsilon$ if $op^x < m+1$, $b = \beta_{m+1}$ if $op^y \geq m+1$, $b = \epsilon$ if $op^y < m+1$, for some operation $op \in Op'$

5. the set of final states is $F := Q'$.

Example 4.2 The transducer $T_1[\{op_{id}, op_{del}, op_{ins}, op_{subs}\}]$ for the usual Levenshtein distance (Example 3.7), alphabet $\Sigma = \{a, b\}$ and edit bound $n = 1$ is shown in Figure 1.

Example 4.3 Let $\Sigma = \{a, b\}$, $S = \{\langle a, b\rangle\}$, $M = \{\langle aa, b\rangle, \langle bb, a\rangle\}$, $Sp = \{\langle a, bb\rangle\}$ and $Op = \{op_{id}, op_{del}, op_{ins}, op_{S-subs}, op_{M-merge}, op_{Sp-split}\}$. $d[Op]$ is the generalized Levenshtein distance with S-restricted substitutions, M-restricted merges and Sp-restricted splits (Example 3.11). $T_2[Op]$ is shown in Figure 2.

Proposition 4.4
$L(T_n[Op]) = \{\langle w, v\rangle \mid d[Op](w, v) \leq n\}$

Proof. Let $k \in Q'$. Then it can be shown that $L(k) = \{\langle w, v\rangle \mid d[Op](x, y) \leq n - k\}$. Hence $L(T_n[Op]) = L(0) = \{\langle w, v\rangle \mid d[Op](w, v) \leq n\}$.

The disadvantage of the two-tape transducer $T_n[Op]$ is that generally there are many ways to decompose a couple of words $\langle w, v\rangle \in \Sigma^* \times \Sigma^*$ into elements of $\Sigma^\epsilon \times \Sigma^\epsilon$. Hence there may be many paths in the transducer that we have to check to verify that $\langle w, v\rangle \in L(T_n[Op])$ or $\langle w, v\rangle \notin L(T_n[Op])$. For example if we use $T_1[\{op_{id}, op_{del}, op_{ins}, op_{subs}\}]$ (Figure 1) to verify that the usual Levenshtein distance between $w = abbab$ and $v = bbba$ is greater than one, we traverse $T_1[\{op_{id}, op_{del}, op_{ins}, op_{subs}\}]$ with the three sequences

1. $\langle a, b\rangle, \langle b, b\rangle, \langle b, b\rangle, \langle a, a\rangle, ...$
2. $\langle a, \epsilon\rangle, \langle b, b\rangle, \langle b, b\rangle, ...$
3. $\langle \epsilon, b\rangle, ...$

to see that $\langle w, v\rangle \notin L(T_1[\{op_{id}, op_{del}, op_{ins}, op_{subs}\}])$. Note that even if we determinize $T_1[\{op_{id}, op_{del}, op_{ins}, op_{subs}\}]$ as a usual nondeterministic automaton over $\Sigma^\epsilon \times \Sigma^\epsilon$, we shall have to traverse again with the above three sequences.

To overcome this problem we use the so-called *synchronized rational relations*. The idea is to convert $T_n[Op]$ into a finite state automaton A that has not transitions of the type $\langle x, \epsilon\rangle$ and $\langle \epsilon, x\rangle$. This means that the two input words for A must have equal lenghts. We pad on the right side the shorter of the two input words with a new symbol $\$ \notin \Sigma$ to match in length the longer input word. So we would like to build a finite state automaton $A = \langle \Sigma^\$ \times \Sigma^\$, Q_A, q_0^A, \Delta_A, F_A\rangle$, $(\Sigma^\$:= \Sigma \cup \{\$\})$, such that $L(A) = \{pad(w, v) \mid \langle w, v\rangle \in L(T_n[Op])\}$, where the function $pad : \Sigma^* \times \Sigma^* \to (\Sigma^\$ \times \Sigma^\$)^*$ is defined as follows:

Definition 4.5 $pad(w,v) :=$

$$\begin{cases} \epsilon & \text{if } w = v = \epsilon \\ \langle w_1, v_1\rangle\langle w_2, v_2\rangle \dots \langle w_{|w|}, v_{|w|}\rangle & \text{if } |w| = |v| \\ \langle w_1, v_1\rangle\langle w_2, v_2\rangle \dots \langle w_{|w|}, v_{|w|}\rangle & \\ \langle \$, v_{|w|+1}\rangle\langle \$, v_{|w|+2}\rangle \dots \langle \$, v_{|v|}\rangle & \text{if } |w| < |v| \\ \langle w_1, v_1\rangle\langle w_2, v_2\rangle \dots \langle w_{|v|}, v_{|v|}\rangle & \\ \langle w_{|v|+1}, \$\rangle\langle w_{|v|+2}, \$\rangle \dots \langle w_{|w|}, \$\rangle & \text{if } |w| > |v| \end{cases}$$

Definition 4.6 Let $R \subseteq \Sigma^* \times \Sigma^*$. R is called *synchronized rational relation* if there exists a finite state automaton A over the alphabet $\Sigma^\$ \times \Sigma^\$$, such that $L(A) = \{pad(w,v) \mid \langle w, v\rangle \in R\}$.

Synchronized rational relations are well investigated: [2, 1, 3]. A comprehensive study of the family of the synchronized rational relations is [3].

It is easy to see that there exists a normal set of operations Op and an edit bound n, such that $L(T_n[Op])$ is not a synchronized rational relation. For example let $\Sigma = \{a\}$, $Op = \{op_{id}, \langle 1, 2, \{\langle a, aa\rangle\}, 0\rangle\}$ and $n = 0$. Let us suppose that $L(T_n[Op])$ is a synchronized rational relation. Then $\{\langle a, a\rangle^k \langle \$, a\rangle^m \mid m \leq k\}$ is a regular language. Let $x = \langle a, a\rangle$ and $y = \langle \$, a\rangle$. So $\{x^k y^m \mid m \leq k\}$ is a regular language over the alphabet $\Sigma = \{x, y\}$. Using the pumping lemma for the regular languages we can check that $\{x^k y^m \mid m \leq k\}$ is not a regular language.

We define a class of set of operations for which $L(T_n[Op])$ is a synchronized rational relation.

Definition 4.7 Let Op be a set of operations. *The zero weighted edit operations of Op lie on the main diagonal* iff $\forall op \in Op : (op^w = 0 \;\&\; op^r \neq \phi) \Rightarrow op^x = op^y$.

Proposition 4.8 *Let Op be a normal set of operations and the zero weighted edit operations of Op lie on the main diagonal. Then $L(T_n[Op])$ is a synchronized rational relation.*

To prove Proposition 4.8 we use the following

Theorem 4.9 (Frougny and Sakarovitch [3]) *A rational relation with bounded length difference is a synchronized rational relation.*

Definition 4.10 Let $T = \langle \Sigma, Q, q_0, \Delta, F\rangle$ is a two-tape transducer and $q \in Q$. The *imbalance set of* q is $Imb(q) := \{||w| - |v|| \mid \langle w, v\rangle \in \overleftarrow{L}(q)\}$. A rational relation R *has a bounded length difference* if there exists a transducer T such that $L(T) = R$ and every state of T has a finite imbalance set.

Proposition 4.11 *Let Op be a normal set of operations. Then the zero weighted edit operations of Op lie on the main diagonal iff every state of $T_n[Op]$ has a finite imbalance set.*

Definition 4.12 Let $P = \langle\langle \pi_1, \langle a_1, b_1\rangle, \pi_2\rangle, \langle \pi_2, \langle a_2, b_2\rangle, \pi_3\rangle, \dots, \langle \pi_k, \langle a_k, b_k\rangle, \pi_{k+1}\rangle\rangle \in \Delta^k$ be a path of $k > 0$ transitions in the two-tape transducer T, whose transition relation is Δ. The *imbalance of* P is $imb(P) := ||a_1 a_2 \dots a_k| - |b_1 b_2 \dots b_k||$.

Proof of Proposition 4.11

Let Op be a normal set of operations and the zero weighted edit operations of Op lie on the main diagonal. Then every simple cycle of transitions $P = \langle\langle \pi_1, \langle a_1, b_1\rangle, \pi_2\rangle, \langle \pi_2, \langle a_2, b_2\rangle, \pi_3\rangle, \dots, \langle \pi_k, \langle a_k, b_k\rangle, \pi_1\rangle\rangle \in \Delta^k$ (Δ is the transition relation of $T_n[Op]$ and $\pi_i \neq \pi_j$ if $i \neq j$ and $max(i,j) < k$) has an imbalance which is equal to zero, $imb(P) = 0$. Hence every state of $T_n[Op]$ has a finite imbalance set. Now let every state of $T_n[Op]$ have a finite imbalance set and let us suppose that $op \in Op$ such that $op^r \neq \phi$, $op^w = 0$ and $op^x \neq op^y$. Then $\{k | op^x - op^y| \mid k \in N\} \subseteq Imb(0)$. Hence $Imb(0)$ is infinite. Contradiction.

Proof of Proposition 4.8

Proposition 4.8 follows from Proposition 4.11, Definiton 4.10 and Theorem 4.9.

5 Synchronizing algorithm

In [3] Frougny and Sakarovitch present an algorithm for synchronization of a rational relation with bounded length difference. Their algorithm converts the input transducer into a letter-to-letter 2-automaton with terminal function (see [3]). Here we present an algorithm that builds from the input transducer $T = \langle \Sigma, Q, q_0, \Delta, F\rangle$ a nondeterministic finite state automaton $A = \langle \Sigma^\$ \times \Sigma^\$, Q_A, q_0^A, \Delta_A, F_A\rangle$, such that $L(A) = \{pad(w,v) \mid \langle w, v\rangle \in L(A)\}$. Our algorithm terminates if every state of T has a finite imbalance set.

Without loss of generality we consider that $\Delta \subseteq Q \times ((\Sigma \times \Sigma) \cup (\{\epsilon\} \times \Sigma) \cup (\Sigma \times \{\epsilon\})) \times Q$ (T has no $\langle \epsilon, \epsilon\rangle$ transitions), $L(q) \neq \phi$ and $\overleftarrow{L}(q) \neq \phi$ for every $q \in Q$.

Traversing a two-tape transducer may be considered as a process of asynchronous moving of two reading heads from state to state (one head for the first tape and one for the second tape). The beginning of the process is the following:

1. We start with the two heads on the initial state q_0

2. If there is a transition $\langle q_0, \langle a, b\rangle, q\rangle \in \Delta$ for some $a, b \in \Sigma$, then we continue the traversal with moving the two heads on the state q

3. If there is a transition $\langle q_0, \langle a, \epsilon\rangle, q\rangle \in \Delta$ for some $a \in \Sigma$, then we continue the traversal with moving only the first head on the state q. The second head stays on state q_0 and waits for the symbol of the next transition that starts from q.

   (a) If it is $\langle q, \langle c, \epsilon\rangle, p\rangle$ for some $c \in \Sigma$, we move the first head on the state p and the second head stays and waits again.

   (b) If it is $\langle q, \langle c_1, c_2\rangle, p\rangle$, then the first head goes in p and the second head goes in q.

   ...

4. If there is a transition $\langle q_0, \langle \epsilon, a\rangle, q\rangle$ for some $a \in \Sigma$, then we proceed symmetrically to case 3.

The process continues with the movements of the two heads or their stays according to the transitions of the transducer. Since $L(T)$ is a rational relation with bounded length difference, there is a constant $c \in N$ such that the slower head, that has more stays and lags, can never be more than c transitions behind the faster one. In our construction every state of the synchronized automaton A encodes the path that the slower head must traverse to reach the other head.

Let us suppose that every state of T has a finite imbalance set. We decompose the transition relation Δ into two relations $\Delta_1 \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q$ and $\Delta_2 \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q$. Δ_1 and Δ_2 give the transitions on the the first and the second tape of the transducer T resp.: $\langle q', x, q'' \rangle \in \Delta_1$ iff there exists $y \in \Sigma \cup \{\epsilon\}$ such that $\langle q', \langle x, y \rangle, q'' \rangle \in \Delta$, $\langle q', y, q'' \rangle \in \Delta_2$ iff there exists $x \in \Sigma \cup \{\epsilon\}$ such that $\langle q', \langle x, y \rangle, q'' \rangle \in \Delta$. We extend Δ_1 and Δ_2 to $\Delta_1^* \subseteq Q \times \Sigma^* \times Q$ and $\Delta_2^* \subseteq Q \times \Sigma^* \times Q$ as usual: Δ_1^* (Δ_2^*) is the smallest subset of $Q \times \Sigma^* \times Q$ such that

1. $\langle q, \epsilon, q \rangle \in \Delta_1^*$ ($\langle q, \epsilon, q \rangle \in \Delta_2^*$) for each $q \in Q$,
2. $\langle q_1, x, q_2 \rangle \in \Delta_1^*$ & $\langle q_2, c, q_3 \rangle \in \Delta_1 \Rightarrow \langle q_1, xc, q_3 \rangle \in \Delta_1^*$
   ($\langle q_1, x, q_2 \rangle \in \Delta_2^*$ & $\langle q_2, c, q_3 \rangle \in \Delta_2 \Rightarrow \langle q_1, xc, q_3 \rangle \in \Delta_2^*$)
   for each $q_1, q_2, q_3 \in Q$, each $x \in \Delta_1^*$ ($x \in \Delta_2^*$) and each $c \in \Sigma \cup \{\epsilon\}$.

To define the states of A we use the relations $\xi_i \subseteq Q \times \Sigma \times Q$ for $i = 1, 2$, which are some kind of ϵ-closed transition relations, defined as follows: $\langle q_1, x, q_2 \rangle \in \xi_i$ iff there exists $q \in Q$ such that $\langle q_1, x, q \rangle \in \Delta_i$ and $\langle q, \epsilon, q_2 \rangle \in \Delta_i^*$. P_i for $i = 1, 2$ are the following sets of sequences of ξ_i-transitions:

$P_i := \{q_1 \xrightarrow{x_1} q_2 \xrightarrow{x_2} q_3 \dots q_k \xrightarrow{x_k} q_{k+1} \in \xi_i^k \mid 0 < k \leq maxImb\}$, where $maxImb = max_{\pi \in Q \ \& \ i \in Imb(\pi)} i$. Here and in what follows we use

$q_1 \xrightarrow{x_1} q_2 \xrightarrow{x_2} q_3 \dots q_k \xrightarrow{x_k} q_{k+1}$

to denote the path

$\langle \langle q_1, x_1, q_2 \rangle, \langle q_2, x_2, q_3 \rangle, \dots, \langle q_k, x_k, q_{k+1} \rangle \rangle$.

P_i is a finite set for $i = 1, 2$, since every state of T has a finite imbalance set and $maxImb$ is well-defined.

We define the nondeterministic automaton $A = \langle \Sigma^\$ \times \Sigma^\$, Q_A, q_0^A, \Delta_A, F_A \rangle$:

1. The set of states is $Q_A := Q_0 \cup Q_1 \cup Q_2 \cup Q_1' \cup Q_2' \cup \{\bot\}$, where
   (a) $Q_0 := \{\langle q, q \rangle \mid q \in Q\}$ (states encoding that the two heads are on the same state)
   (b) $Q_1 := \{\langle \epsilon, P \rangle \mid P \in P_1\}$ (states encoding the path that the first head must traverse to reach the second one)
   (c) $Q_2 := \{\langle P, \epsilon \rangle \mid P \in P_2\}$ (states encoding the path that the second head must traverse to reach the first one)
   (d) $Q_1' := \{\langle \$, P \rangle \mid P \in P_1\}$ (states encoding that the first input word has been consumed and \$-s are to be consumed)
   (e) $Q_2' := \{\langle P, \$ \rangle \mid P \in P_2\}$ (states encoding that the second input word has been consumed and \$-s are to be consumed)
   (f) $\bot$ is a *sink state*, $\langle \bot, c, p \rangle \notin \Delta_A$ for all $c \in (\Sigma^\$ \times \Sigma^\$) \cup \{\epsilon\}$ and all $p \in Q_A$
2. The initial state is $q_0^A := \langle q_0, q_0 \rangle$
3. The set of final states is $F_A := \{\langle q, q \rangle \mid q \in F\} \cup \{\bot\}$
4. The transition relation Δ_A is the subset of $Q_A \times ((\Sigma^\$ \times \Sigma^\$) \cup \{\epsilon\}) \times Q_A$ defined as follows: Let $\pi_1, \pi_2 \in Q_A$ and $c \in (\Sigma^\$ \times \Sigma^\$) \cup \{\epsilon\}$. Then $\langle \pi_1, c, \pi_2 \rangle \in \Delta_A$ iff one of the following conditions is satisfied:
   (a) π_1 has the type $\langle q', q' \rangle$ for some $q' \in Q$, $c \in \Sigma^2$, π_2 has the type $\langle q'', q'' \rangle$ for some $q'' \in Q$ and $\langle q', c, q'' \rangle \in \Delta$
   (b) π_1 has the type $\langle q', q' \rangle$ for some $q' \in Q$, $c = \epsilon$, π_2 has the type $\langle \epsilon, q' \xrightarrow{y} q'' \rangle$ for some $q'' \in Q$ and some $y \in \Sigma$ such that $\langle q', \langle \epsilon, y \rangle, q'' \rangle \in \Delta$
   (c) π_1 has the type $\langle q', q' \rangle$ for some $q' \in Q$, $c = \epsilon$, π_2 has the type $\langle q' \xrightarrow{x} q'', \epsilon \rangle$ for some $q'' \in Q$ and some $x \in \Sigma$ such that $\langle q', \langle x, \epsilon \rangle, q'' \rangle \in \Delta$
   (d) π_1 has the type $\langle \epsilon, q_1 \xrightarrow{y_1} q_2 \xrightarrow{y_2} q_3 \xrightarrow{y_3} q_4 \dots q_k \xrightarrow{y_k} q_{k+1} \rangle$, $c = \langle a, y_1 \rangle$ for some $a \in \Sigma$, π_2 has the type $\langle \epsilon, q_2 \xrightarrow{y_2} q_3 \xrightarrow{y_3} q_4 \dots q_k \xrightarrow{y_k} q_{k+1} \xrightarrow{b} q_{k+2} \rangle$ for some $b \in \Sigma$ and some $q_{k+2} \in Q$ such that $\langle q_{k+1}, \langle a, b \rangle, q_{k+2} \rangle \in \Delta$
   (e) π_1 has the type $\langle \epsilon, q_1 \xrightarrow{y_1} q_2 \xrightarrow{y_2} q_3 \dots q_k \xrightarrow{y_k} q_{k+1} \rangle$, $c = \epsilon$, π_2 has the type $\langle \epsilon, q_1 \xrightarrow{y_1} q_2 \xrightarrow{y_2} q_3 \dots q_k \xrightarrow{y_k} q_{k+1} \xrightarrow{b} q_{k+2} \rangle$ for some $b \in \Sigma$ and some $q_{k+2} \in Q$ such that $\langle q_{k+1}, \langle \epsilon, b \rangle, q_{k+2} \rangle \in \Delta$
   (f) π_1 has the type $\langle \epsilon, q_1 \xrightarrow{y_1} q_2 \rangle$, $c = \langle a, y_1 \rangle$ for some $a \in \Sigma$, π_2 has the type $\langle q, q \rangle$ for some $q \in Q$ such that $\langle q_2, \langle a, \epsilon \rangle, q \rangle \in \Delta$
   (g) π_1 has the type $\langle \epsilon, q_1 \xrightarrow{y_1} q_2 \xrightarrow{y_2} q_3 \xrightarrow{y_3} q_4 \dots q_k \xrightarrow{y_k} q_{k+1} \rangle$ for some $k \geq 2$, $c = \langle a, y_1 \rangle$ for some $a \in \Sigma$, π_2 has the type $\langle \epsilon, q_2 \xrightarrow{y_2} q_3 \xrightarrow{y_3} q_4 \dots q_k \xrightarrow{y_k} q \rangle$ for some $q \in Q$ such that $\langle q_{k+1}, \langle a, \epsilon \rangle, q \rangle \in \Delta$
   (h) π_1 has the type $\langle q_1 \xrightarrow{x_1} q_2 \xrightarrow{x_2} q_3 \xrightarrow{x_3} q_4 \dots q_k \xrightarrow{x_k} q_{k+1}, \epsilon \rangle$, $c = \langle x_1, b \rangle$ for some $b \in \Sigma$, π_2 has the type $\langle q_2 \xrightarrow{x_2} q_3 \xrightarrow{x_3} q_4 \dots q_k \xrightarrow{x_k} q_{k+1} \xrightarrow{a} q_{k+2}, \epsilon \rangle$ for some $a \in \Sigma$ and some $q_{k+2} \in Q$ such that $\langle q_{k+1}, \langle a, b \rangle, q_{k+2} \rangle \in \Delta$
   (i) π_1 has the type $\langle q_1 \xrightarrow{x_1} q_2 \xrightarrow{x_2} q_3 \dots q_k \xrightarrow{x_k} q_{k+1}, \epsilon \rangle$, $c = \epsilon$, π_2 has the type $\langle q_1 \xrightarrow{x_1} q_2 \xrightarrow{x_2} q_3 \dots q_k \xrightarrow{x_k} q_{k+1} \xrightarrow{a} q_{k+2}, \epsilon \rangle$ for some $a \in \Sigma$ and some $q_{k+2} \in Q$ such that $\langle q_{k+1}, \langle a, \epsilon \rangle, q_{k+2} \rangle \in \Delta$
   (j) π_1 has the type $\langle q_1 \xrightarrow{x_1} q_2, \epsilon \rangle$, $c = \langle x_1, b \rangle$ for some $b \in \Sigma$, π_2 has the type $\langle q, q \rangle$ for some $q \in Q$ such that $\langle q_2, \langle \epsilon, b \rangle, q \rangle \in \Delta$
   (k) π_1 has the type $\langle q_1 \xrightarrow{x_1} q_2 \xrightarrow{x_2} q_3 \xrightarrow{x_3} q_4 \dots q_k \xrightarrow{x_k} q_{k+1}, \epsilon \rangle$ for some $k \geq 2$, $c = \langle x_1, b \rangle$ for

some $b \in \Sigma$, π_2 has the type $\langle q_2 \xrightarrow{x_2} q_3 \xrightarrow{x_3} q_4 \dots q_k \xrightarrow{x_k} q, \epsilon\rangle$ for some $q \in Q$ such that $\langle q_{k+1}, \langle \epsilon, b\rangle, q\rangle \in \Delta$

(l) π_1 has the type $\langle \epsilon, q_1 \xrightarrow{y_1} q_2\rangle$ for some $q_2 \in F$, $c = \langle \$, y_1\rangle$, π_2 is the sink state $\perp$

(m) π_1 has the type $\langle \epsilon, q_1 \xrightarrow{y_1} q_2 \xrightarrow{y_2} q_3 \xrightarrow{y_3} q_4 \dots q_k \xrightarrow{y_k} q_{k+1}\rangle$ for some $k \geq 2$ and some $q_{k+1} \in F$, $c = \langle \$, y_1\rangle$, π_2 has the type $\langle \$, q_2 \xrightarrow{y_2} q_3 \xrightarrow{y_3} q_4 \dots q_k \xrightarrow{y_k} q_{k+1}\rangle$

(n) π_1 has the type $\langle q_1 \xrightarrow{x_1} q_2, \epsilon\rangle$ for some $q_2 \in F$, $c = \langle x_1, \$\rangle$, π_2 is the sink state $\perp$

(o) π_1 has the type $\langle q_1 \xrightarrow{x_1} q_2 \xrightarrow{x_2} q_3 \xrightarrow{x_3} q_4 \dots q_k \xrightarrow{x_k} q_{k+1}, \epsilon\rangle$ for some $k \geq 2$ and some $q_{k+1} \in F$, $c = \langle x_1, \$\rangle$, π_2 has the type $\langle q_2 \xrightarrow{x_2} q_3 \xrightarrow{x_3} q_4 \dots q_k \xrightarrow{x_k} q_{k+1}, \$\rangle$

(p) π_1 has the type $\langle \$, q_1 \xrightarrow{y_1} q_2\rangle$, $c = \langle \$, y_1\rangle$, π_2 is the sink state $\perp$

(q) π_1 has the type $\langle q_1 \xrightarrow{x_1} q_2, \$\rangle$, $c = \langle x_1, \$\rangle$, π_2 is the sink state $\perp$

(r) π_1 has the type $\langle \$, q_1 \xrightarrow{y_1} q_2 \xrightarrow{y_2} q_3 \xrightarrow{y_3} q_4 \dots q_k \xrightarrow{y_k} q_{k+1}\rangle$ for some $k \geq 2$, $c = \langle \$, y_1\rangle$, π_2 has the type $\langle \$, q_2 \xrightarrow{y_2} q_3 \xrightarrow{y_3} q_4 \dots q_k \xrightarrow{y_k} q_{k+1}\rangle$

(s) π_1 has the type $\langle q_1 \xrightarrow{x_1} q_2 \xrightarrow{x_2} q_3 \xrightarrow{x_3} q_4 \dots q_k \xrightarrow{x_k} q_{k+1}, \$\rangle$ for some $k \geq 2$, $c = \langle x_1, \$\rangle$, π_2 has the type $\langle q_2 \xrightarrow{x_2} q_3 \xrightarrow{x_3} q_4 \dots q_k \xrightarrow{x_k} q_{k+1}, \$\rangle$

The proof of the following proposition is omitted.

Proposition 5.1
$L(A) = \{pad(\langle w, v\rangle) \mid \langle w, v\rangle \in L(T)\}$

The procedure, that builds the nondeterministic automaton $A = \langle \Sigma^\$ \times \Sigma^\$, Q_A, q_0^A, \Delta_A, F_A\rangle$ by a given input transducer T, starts from the initial state $q_0^A = \langle q_0, q_0\rangle$ and using the definiton of the transition relation Δ_A finds in breadth all states of A that are reachable from the initial one. The procedure terminates if every state of T has a finite imbalance set. After ϵ-removal, determinization and minimization we get from A the final *synchronized automaton* that can be used in practice.

So we build a synchronized automaton S from $T_n[Op]$. S recognizes an input word $pad(w, v)$ if and only if $d[Op](w, v) \leq n$. Let us suppose that we finish the traversal of S with $pad(w, v)$ in a final state. It may be useful to know by the final state the distance $d[Op](w, v)$. For this aim a little modification of the construction of A is needed. In cases (l), (n), (p) and (q) π_2 has to be $\langle q_2, q_2\rangle$ instead of $\perp$. Then the distance that corresponds to a final state $\langle k, k\rangle$ is k. This information can be kept during the determinization and the minimization. After the above modification $L(A) \supset \{pad(w, v) \mid d[Op](w, v) \leq n\}$, but $d[Op](w, v) \leq n \Rightarrow pad(w, v) \in L(A)$ and $d[Op](w, v) > n \Rightarrow pad(w, v) \notin L(A)$.

Table 1 shows the dependence of the synchronized automaton for the usual Levenshtein distance on the size of the alphabet and the edit bound.

alphabet size	*edit bound*	*states*	*transitions*
2	1	14	58
2	2	187	1,082
2	3	2,438	16,384
2	4	28,557	205,500
10	1	222	5,250
50	1	5,102	63,0250

Table 1: *Synchronized automata for the usual Levenshtein distance*

	subs	*merge*	*split*	*cand*	*recall*	*univ*	*synchr*
(a)	0	1	1	7.78	70.565%	0.032 ms	0.012 ms
(b)	0	0	0	45.73	94.52%	0.107 ms	0.076 ms
(c)	0.0006	0.0325	0.0005	5.48	94.519%	0.049 ms	0.013 ms

Table 2: *Test for distance bound $n = 1$.*

The synchronized automaton for the usual Levenshtein distance, $\Sigma = \{a, b\}$ and edit bound 1 is shown in Figure 3. If we finish a traversal of the automaton with $pad(w, v)$ in state 0, then the Levenshtein distance between w and v is 0. If we finish in state $1, 2, 3, 4, 6, 7$ or 9, then the distance is 1.

6 Evaluation results

As a test set for the new method we use the TREC-5 corpus with character error rate 5% [5] and an English dictionary with 264,061 words. Via dynamic programming we extract from the corpus couples of the type $\langle$*pattern*, *original*$\rangle$, where *pattern* is an OCRed word and *original* is its corresponding original. The problem of case errors was ignored for these tests.

As a measure for the proximity between *pattern* and *original* we use the generalized Levenshtein distance with S-restricted substitutions, M-restricted merges and Sp-restricted splits (Example 3.11). We use 1/2 of these couples as a training set to calculate the relative frequency of each symbol dependent substitution, merge and split. We use thresholds *subs*, *merge*, and *split* to define resp. the sets S, M and Sp. S contains only the substitutions with relative frequencies greater than *subs*. Analogously we determine the sets M and Sp. Estimating the relative frequencies can be done also without using ground truth training data [12]. Using the remaining 1/2 we prepared a test set that consist of 100,000 couples $\langle$*pattern*, *original*$\rangle$ such that *original* is in the dictionary. For each *pattern* from the test set we select a set of correction candidates from the dictionary, applying the forward-backward method. Table 2 represents the results for edit bound $n = 1$. The number *cand* gives the average number of correction candidates per pattern found in the dictionary. The length of every test pattern does not exceed 6. For longer patterns greater edit bounds are needed. *univ* is the average time per pattern needed to find the correction candidates using universal automaton. *synchr* is the average time per pattern needed to find the correction candidates using synchronized automaton. *recall* is the percentage of patterns where the correct *original* is found in the selected set of correction candidates. The CPU of the machine used for the experiments is Pentium 4, 2.4 GHz with 1GB RAM.

For the usual Levenshtein distance (a) the algo-

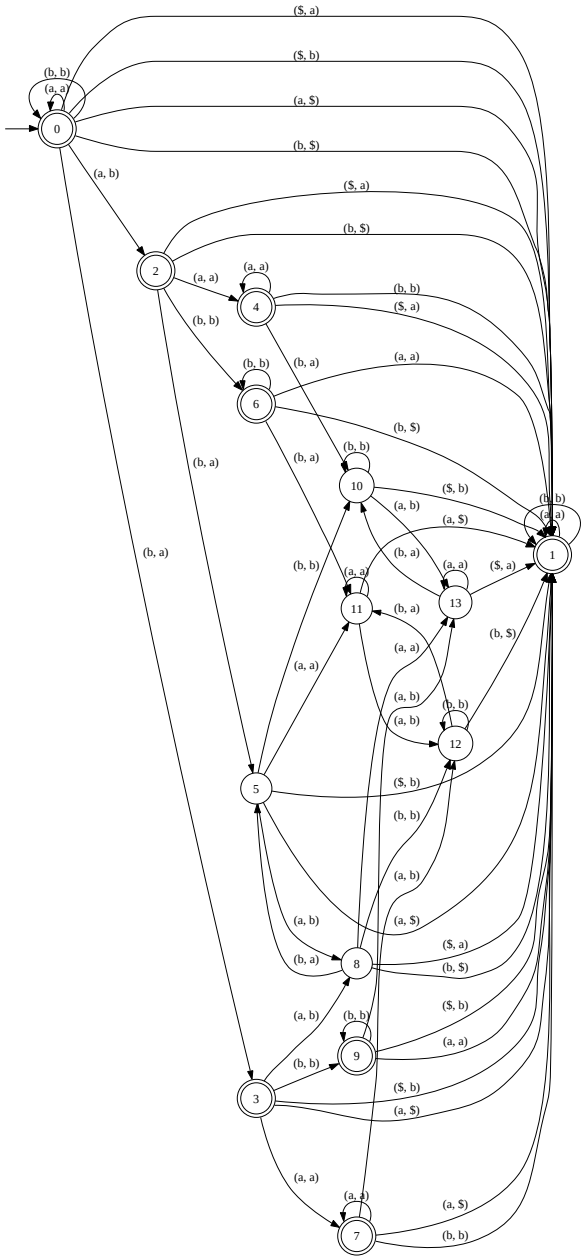

Fig. 3: *Synchronized automaton for the usual Levenshtein distance,* $\Sigma = \{a, b\}$*, the edit bound is* 1

rithm that uses synchronized automata is approximately three time faster than the algorithm that uses universal automata. For the refined Levenshtein distance (c) - approximately four times. This is due to the fact that the universal automaton for the usual Levenshtein distance uses only one characteristic vector and the universal automaton for the refined distance (c) uses four characteristic vectors. Synchronized automata do not require computation of characteristic vectors. Although in case (b) the universal automaton also uses one characteristic vector, synchronized automata are not as faster than in cases (a) and (c). The reason for this is probably the high number of candidates extracted from the dictionary. The high number of candidates means that there is too much traversal of the dictionary. Probably the traversal (that is changing the current states, some call stack operations etc.) requires more time than computing the characteristic vectors.

Unfortunately, when we increase the edit bound, the size of the synchronized automaton grows drasticly fast. Even when the edit bound is 2 and the alphabet size is 26, for the refined Levenshtein distance (c) with proper thersholds we did not succeed to determinize A.

Other negative feature of the synchronized automata in comparison to the universal automata is that for the forward-backward method we have to use two synchronized automata: A_1, such that $L(A_1) = \{pad(w, v) \mid d[Op](w, v) \leq n\}$, and A_2, such that $L(A_2) = \{pad(w, v) \mid d[Op^{rev}](w, v) \leq n\}$. Here Op^{rev} is the set of the reversed operations of Op: $Op^{rev} = \{\langle op^x, op^y, \{\langle \alpha^{rev}, \beta^{rev}\rangle \mid \langle \alpha, \beta\rangle \in op^r\}, op^w\rangle \mid op \in Op\}$.

7 Conclusion and future work

Synchronized automata are extremely effective instrument that could be used for approximate matching problems. In our experiments we used only uniform wegths (0 and 1) of the edit operations. But with synchronized automata we could manage with more than two degrees of discretization. Future work will be devoted to the problem of the great size of the synchronized automata. Algorithm for direct incremental building of the minimal synchronized automata could be useful.

References

[1] S. Eilenberg. *Automata, Languages and Machines, Vol. A.* Academic Press, New York, 1974.

[2] C. Elgot and J. Mezei. On relations defined by generalized finite automata. *IBM J. Res Develop.*, 9:47–68, 1965.

[3] C. Frougny and J. Sakarovitch. Synchronized rational relations of finite and infinite words. *Theoretical Computer Science*, 108:45–82, 1993.

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

[5] P. B. Kantor and E. M. Voorhees. The TREC-5 confusion track: Comparing retrieval methods for scanned text. *Information Retrieval*, 2(2/3):165–176, 2000.

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

[7] T. A. Lasko and S. E. Hauser. Approximate string matching algorithms for limited-vocabulary ocr output correction. In *Proceedings of SPIE, Vol. 4307, Document Recognition and Retrieval DRR(VIII), January 24-25, 2001.*, pages 241–249, 2001.

[8] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Sov. Phys. Dokl.*, 10:707–710, 1966.

[9] S. Mihov and K. U. Schulz. Fast approximate search in large dictionaries. *Computational Linguistics*, 30(4):451–477, December 2004.

[10] B. J. Oommen and R. K. Loke. Pattern recognition of strings with substitutions, insertions, deletions and generalized transposition. *Pattern Recognition*, 30(7):789–800, 1997.

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

[12] C. Ringlstetter, U. Reffle, A. Gotscharek, and K. U. Schulz. Deriving symbol dependent edit weights for text correction - the use of error dictionaries. In *Proceedings of the ninth International Conference on Document Analysis and Recognition (ICDAR)*, page to appear, 2007.

[13] E. Roche and Y. Schabes. Deterministic part-of-speech tagging with finite state transducers. *Computational Linguistics*, 21(2):227–253, 1995.

[14] K. U. Schulz and S. Mihov. Fast String Correction with Levenshtein-Automata. *International Journal of Document Analysis and Recognition*, 5(1):67–85, 2002.

[15] K. U. Schulz, S. Mihov, and P. Mitankin. Fast selection of small and precise candidate sets from dictionaries for text correction tasks. In *Proceedings of the ninth International Conference on Document Analysis and Recognition (ICDAR)*, page to appear, 2007.

[16] C. Strohmaier, C. Ringlstetter, K. U. Schulz, and S. Mihov. A visual and interactive tool for optimizing lexical postcorrection of OCR results. In *Proceedings of the IEEE Workshop on Document Image Analysis and Recognition, DIAR'03*, 2003.

[17] R. A. Wagner and M. J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173, 1974.

[18] A. 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.

[19] A. Weigel, T. Jager, and A. Pies. Estimation of probabilities for edit operations. In *Proceedings of the 15th International Conference on Pattern Recognition*, volume 2, pages 777–780, 2000.

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