In the Proc. of IEEE ISOC ITW2005 on Coding and Complexity; editor M.J. Dinneen; co-chairs U. Speidel and D. Taylor; pages 113-116

Computing the Levenshtein Distance of a Regular Language

Stavros Konstantinidis
Department of Mathematics and Computing Science
Saint Mary's University
Halifax, Nova Scotia B3H 3C3, Canada
Email: stavros@cs.smu.ca

***Abstract*— The edit distance (or Levenshtein distance) between two words is the smallest number of substitutions, insertions, and deletions of symbols that can be used to transform one of the words into the other. In this paper we consider the problem of computing the edit distance of a regular language (also known as constraint system), that is, the set of words accepted by a given finite automaton. This quantity is the smallest edit distance between any pair of distinct words of the language. We show that the problem is of polynomial time complexity. We distinguish two cases depending on whether the given automaton is deterministic or nondeterministic. In the latter case the time complexity is higher.**

I. Introduction

The problem of measuring the distance, or generally the difference, between words and languages (sets of words) is important in various applications of information processing such as error control in data communications, bio-informatics, speech recognition and spelling correction. The languages of interest are usually regular languages (also called constraint systems), that is, languages defined by finite automata, but they could be non regular as well. Well-known measures of the difference between two words are the edit (or Levenshtein) distance and the Hamming distance. The edit distance between two words is the smallest number of substitutions, insertions, and deletions of letters required to transform one of the words to the other. Typical problems pertaining to difference measures for words and languages are the following.

The word difference problem: Compute the edit distance between two given words. The problem can be solved using a dynamic programming algorithm – see [7], for instance.

The error-correction problem: Given a language of valid (or correct) words, correct a given word to some word of the language that is the least different to the given word. This problem presupposes an agreed measure of word difference such as the edit distance. For (arbitrary) regular languages and for the measure of edit distance the problem was first addressed in [8]. In [6] it is addressed for cases where the language in question is not even regular. In [5] and [1] the problem is considered for regular languages and for general word difference measures defined by weighted automata. In coding theory of course, the problem has been addressed extensively for various instances of the word difference measure – this measure is usually implicitly specified by the communications channel that is capable of changing words.

The error-detection capability problem: Compute the distance (also referred to as self-distance or inner distance) of a given language. This quantity is simply the minimum distance between any pair of distinct words in the language. When the language in question is viewed as a code, the value of the distance represents the maximum number of errors that the code can detect. For example, it is well known that a block code (set of equal length words) can detect up to m bit substitution errors if and only if the Hamming distance of the code is greater than m. A similar observation exists for the case of the edit distance [3]. In [2], the authors show how to compute the Hamming distance of a given regular language.

In this work, we address the third problem for the case of the edit distance of a regular language. In this case, the value of the edit distance represents the maximum number m, say, of substitution, insertion and deletion errors that can be *detected* in the words of the language. This means that no word of the language can be transformed to another different word of the language if up to m errors are used. The paper is organized as follows. In the next section we provide the basic notation about words, automata, and edit strings. An edit string is a special word whose symbols are called edit operations. It has been used to define formally the edit distance between ordinary words. Here we also consider languages and finite automata of edit strings as tools for reasoning about the distance of a language. In Section III, we obtain a few lemmata that will be used to prove the correctness of the main result of the paper. We believe that these lemmata might be of interest in their own right. Section IV contains the main result of the paper, that is, a polynomial time algorithm to compute the edit distance of a given regular language. The time complexity of the algorithm is higher when the language is given via a nondeterministic automaton. Section V contains a few concluding remarks and proposes possible directions for future research.

II. Basic background and notation

Cardinality, alphabet, word, language.
For any set S, we use the expression $|S|$ to denote the *cardinality* of the set S. A *word* or *string* (over some alphabet Δ) is a finite sequence $a_1 \cdots a_n$ such that each a_i is in Δ. A *language* is a set of words. The language of all words is denoted by Δ^*. A word w of length n can be viewed as a mapping from the index set $\{1, \ldots, n\}$ into Δ such that $w(i)$

is *the symbol of w at position i*. If $i > n$, we agree to write that $w(i) = \perp$, where $\perp$ is a symbol not in the alphabet Δ. With this convention, it follows that any two words u and v are different if and only if $u(i) \neq v(i)$ for some positive integer i.

Finite automaton, diameter, regular language.
A (nondeterministic) *finite automaton* is a quintuple $A = (\Delta, S, s_0, F, T)$ such that Δ is the alphabet, S is the finite set of states, s_0 is the start state, F is the set of final states, and T is the set of transitions, which we denote with expressions paq such that p, q are states and a is a symbol in Δ. The automaton A is said to be deterministic if, for any two transitions of the form paq_1, paq_2, it is the case that $q_1 = q_2$. The automaton can be viewed as an edge-labeled directed graph. In this sense, the *diameter of A*, denoted by $\mathrm{diam}(A)$, is the largest number of states in a path without cycles (repeated states) that starts from the start state of A. We have that $1 \leq \mathrm{diam}(A) \leq |S|$.

The *language accepted* by the automaton A, denoted by $L(A)$, is the set of words that are formed in paths of A from the start state to a final state. The *size* of A, denoted with $|A|$, is the quantity $|S| + |T|$, which is the number of states plus the number of transitions. A language L is called *regular*, or a *constraint system*, if there is a finite automaton accepting L [4], [9]. If A and A' are two finite automata then the expression $A \cap A'$ denotes the finite automaton that obtains using the standard cross product construction such that $L(A \cap A') = L(A) \cap L(A')$ – see [9]. Moreover, $|A \cap A'| = O(|A||A'|)$.

Ordinary word, edit string, weight, edit distance
In this paper we shall use a fixed, but arbitrary, alphabet Σ of *ordinary* symbols, and the alphabet E of the *(basic) edit operations* that depends on Σ. The *empty word* over Σ is denoted by λ, that is, $\lambda w = w\lambda = w$, for all words w. The alphabet E consists of all symbols (x/y) such that $x, y \in \Sigma \cup \{\lambda\}$ and at least one of x and y is in Σ. If (x/y) is in E and x is not equal to y then we call (x/y) an *error*. We write (λ/λ) for the empty word over the alphabet E. The elements of E^* are called *edit strings* [1], or alignments [5]. The *input* and *output* parts of an edit string $h = (x_1/y_1) \cdots (x_n/y_n)$ are the words (over Σ) $x_1 \cdots x_n$ and $y_1 \cdots y_n$, respectively. We write $\mathrm{inp}(h)$ for the input part and $\mathrm{out}(h)$ for the output part of h. The expression $\mathrm{weight}(h)$ denotes the number of errors in h. The *edit (or Levenshtein) distance* between two words u and v, denoted by $\mathrm{dist}(u, v)$, is the smallest number of errors (substitutions, insertions and deletions of symbols) that can transform u to v. More formally, $\mathrm{dist}(u, v) =$

$$\min\{\mathrm{weight}(h) \mid h \in E^*,\ \mathrm{inp}(h) = u,\ \mathrm{out}(h) = v\}.$$

For example, when the alphabet Σ is $\{a, b\}$, we have that $\mathrm{dist}(ababa, babbb) = 3$ and the edit string

$$h = (a/\lambda)(b/b)(a/a)(b/b)(a/b)(\lambda/b)$$

is a minimum weight edit string that transforms *ababa* to *babbb*. In words, h says that we can use the deletion (a/λ), the substitution (a/b), and the insertion (λ/b) to transform *ababa* to *babbb*.

If L is a language containing at least two words then the edit distance of L is

$$\mathrm{dist}(L) = \min\{\mathrm{dist}(u, v) \mid u, v \in L \text{ and } u \neq v\}.$$

III. Intermediate lemmata

In this section we obtain a few lemmata that will be used to prove the correctness of the main result of the paper. We believe that these lemmata might be of interest in their own right. Moreover the following paragraph provides a useful technical tool.

The automaton $A \cap_E A$.
Given two automata A, A' and a subset D of E, the finite automaton $A \cap_D A'$ [1] accepts all edit strings h that transform a word of $L(A)$ into a word of $L(A')$ using only the edit operations in D. Moreover, we note that $|A \cap_D A'| = O(|A||A'|)$. In our considerations in particular, we shall use the finite automaton $A \cap_E A$ such that

$$L(A \cap_E A) = \{h \in E^* \mid \mathrm{inp}(h), \mathrm{out}(h) \in L(A)\}.$$

The states of $A \cap_E A$ are pairs (p, q), where p and q are states of A. Hence, if A has s states then $A \cap_E A$ has at most s^2 states.

Lemma 1: Let A be a deterministic finite automaton accepting at least two words. There are distinct words u, v in $L(A)$ and an index $j \leq \mathrm{diam}(A)$ such that $u(j) \neq v(j)$ and $\mathrm{dist}(L(A)) = \mathrm{dist}(u, v)$.

Lemma 2: Let A be a (nondeterministic) finite automaton accepting at least two words. There are distinct words u, v in $L(A)$ and an index $j \leq s^2$ such that $u(j) \neq v(j)$ and $\mathrm{dist}(L(A)) = \mathrm{dist}(u, v)$, where s is the number of states in A.

Thus, when the automaton A is nondeterministic the upper bound on the index j is larger. In the next lemma we show that the edit distance of a regular language is upper bounded by the diameter of any automaton accepting the language.

Lemma 3: For any finite automaton A accepting at least two words, $\mathrm{dist}(L(A)) \leq \mathrm{diam}(A)$.

Next we show the existence of a polynomial size automaton accepting all edit strings h for which the input part of h and the output part of h differ at position k, for some given positive integer k.

Lemma 4: For any positive integer k, we can construct a finite automaton T_k of size $\Theta(k^2|\Sigma|^2 + k|\Sigma|^3)$ such that T_k accepts all edit strings h for which $(\mathrm{inp}(h))(k) \neq (\mathrm{out}(h))(k)$.

IV. Computing the distance in polynomial time

In the first place one could approach the problem of computing the edit distance of a regular language as follows. Let A be the given finite automaton, and let B be the automaton $A \cap_E A$ (see Section III) accepting all edit strings for which the input and output parts are in $L(A)$. If we are able to construct an automaton T accepting all edit strings for which the input and output parts are distinct, then we can compute the automaton $B \cap T$ that accepts all edit strings for which the input and output parts are distinct and in $L(A)$. When $B \cap T$ is treated as a weighted graph such that the weight of a transition is either 0 or 1, and it is 1 when the label of the transition is an error, then the edit distance of $L(A)$ is simply the weight of the shortest path in $B \cap T$. The question here is whether the required automaton T exists. We believe that T does not exist. For the sake of completeness, however, we note that one can construct a finite-state transducer T realizing all pairs of words (u, v) with $u \neq v$, but when viewed as an automaton, T does not accept *all* possible edit strings h whose parts are distinct as desired. Although the idea described above fails, we can build on that idea using the results of the previous section and arrive at the desired algorithm.

Theorem 1: The following problem is computable in polynomial time.

Input: A (deterministic or nondeterministic) finite automaton A accepting at least two words.

Output: The edit distance $\operatorname{dist}(L(A))$.

Proof: Let $q(A)$ be the quantity $\operatorname{diam}(A)$ or s^2, depending on whether A is deterministic or not, where s is the number of states in A. We have the following algorithm.

```
Input = some finite automaton $A$;
dist = $\operatorname{diam}(A)$;
$B = A \cap_E A$;
$m = q(A)$;
for each $j = 1, \ldots, m$
{
    $G = B \cap T_j$;
    weight = ShortestPathWeight($G$);
    if (weight $<$ dist) dist = weight;
}
Output = dist;
```

The function call ShortestPathWeight(G) treats the automaton G as a weighted graph and computes the shortest path to every state from the start state. The weight of a transition $p(x/y)q$ of G, where (x/y) is an edit operation and p, q are states, is 1 if $x \neq y$, or 0 if $x = y$. The algorithm initializes dist to the maximum possible value of $\operatorname{dist}(L(A))$ and then updates dist according to the equation $\operatorname{dist}(L(A)) =$

$$\min\{\operatorname{weight}(h) \mid h \in L(A \cap_E A) \cap L(T_j), \text{for some} j \leq q(A)\}.$$

Using the lemmata of Section III, this establishes that the algorithm operates correctly. For the time complexity of the algorithm, first note that the function call $\operatorname{diam}(A)$ operates in time $O(|A|)$. The automaton B can be computed in time $O(|A|^2)$ and the automaton G in time $O(|A|^2|T_j|)$. The function ShortestPathWeight(G) operates in linear time $O(|A|^2|T_j|)$ because the weights of G can only be 0 and 1 – see [2]. Hence, using also Lemma 4, the dominating term in the time complexity is

$$\sum_{j=1}^{m} |A|^2(j^2|\Sigma|^2 + j|\Sigma|^3) = O\left(|A|^2|\Sigma|^2 q(A)^2(q(A) + |\Sigma|)\right).$$

■

The question that arises here is whether the quantity $m = q(A)$, which is the upper bound of the loop of the algorithm, can be reduced asymptotically. In turn, the question is whether the upper bounds on the index j that appear in the lemmata of Section III can be improved asymptotically. Let $\operatorname{MinInd}(A)$ denote the smallest value of an index j for which there are words u, v in $L(A)$ such that $u(j) \neq v(j)$ and $\operatorname{dist}(u, v) = \operatorname{dist}(L(A))$. The upper bound $\operatorname{diam}(A)$ of Lemma 1 on the index $\operatorname{MinInd}(A)$ cannot be improved. More specifically there is a sequence (A_n) of deterministic finite automata such that $\operatorname{diam}(A_{n-1}) < \operatorname{diam}(A_n)$ and, for all $n \geq 2$, $\operatorname{MinInd}(A_n) = \operatorname{diam}(A_n) = n$. Similarly, the upper bound s^2 of Lemma 2 on the index $\operatorname{MinInd}(A)$ cannot be improved asymptotically. More specifically, there is a sequence (B_n) of nondeterministic finite automata such that $s_{n-1} < s_n$, where s_n is the number of states in B_n, and $\operatorname{MinInd}(B_n) = \Theta(s_n^2)$, as $n \to \infty$. The automaton B_n is shown in Figure 1. The words ab^{n^2-n} and $ab^{n^2-n}c$ differ at position $n^2 - n + 2$, which implies that $\operatorname{dist}(L(B_n)) = 1$ and $\operatorname{MinInd}(B_n) = n^2 - n + 2 = \Theta(s_n^2)$.

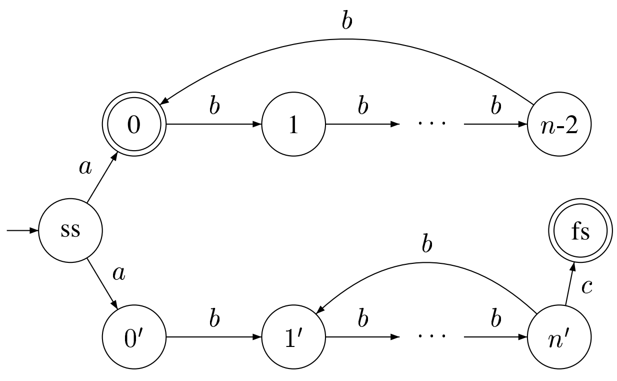

Fig. 1. The nondeterministic finite automaton B_n. The start state is ss. Final states are indicated with double lines.

V. Discussion

We have shown why the problem of computing the edit distance of a given regular language is of polynomial time complexity. We have restricted our attention to the case of the unweighted edit distance because this case is closely related to the concept of error detection as discussed in the introduction. However, the methods can be applied even in the case where the edit distance is weighted such that the weights on the errors are positive numbers with a slight increase in the time complexity of the algorithm.

The first question that arises is whether the time complexity of the algorithms can be improved asymptotically. It appears that this is possible for certain special cases at least. For example, when the given automaton A is a trellis, that is, an automaton with a single final state accepting only words of the same length n, then $n = \operatorname{diam}(A) - 1$ and in typical applications n is much smaller than $|A|$. The next question that arises is whether the results extend to the case of other difference measures for words, in particular, those defined by weighted automata [5], [1]. In this case, the edit strings that one can use to transform words are restricted to only those permitted by the weighted automaton.

Acknowledgment

This research was supported by grant R220259 of the Natural Sciences and Engineering Research Council of Canada. The author is grateful to Nicolae Sântean for providing fruitful comments on this paper.

References

[1] L. Kari, S. Konstantinidis, S. Perron, G. Wozniak, J. Xu, "Finite-state error/edit-systems and difference-measures for languages and words," *Tech. Report 2003-01, Dept. Math. and Computing Sci., Saint Mary's University, Canada,* pp 10, 2003. Available electronically at `http://www.stmarys.ca/academic/science/compsci/`

[2] L. Kari, S. Konstantinidis, S. Perron, G. Wozniak, J. Xu, "Computing the Hamming distance of a regular language in quadratic time," *WSEAS Transactions on Information Science & Applications* vol. 1, pp. 445–449, 2004.

[3] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet Physics Dokl.* vol. 10, pp. 707–710, 1966.

[4] B. H. Marcus, R. M. Roth, P. H. Siegel, "Constrained systems and coding for recording channels." In V. S. Pless, W. C. Huffman (eds): *Handbook of Coding Theory*. Elsevier, 1998, pp. 1635–1764.

[5] Mehryar Mohri, "Edit-distance of weighted automata: general definitions and algorithms," *International Journal of Foundations of Computer Science* vol. 14, pp. 957–982, 2003.

[6] G. Pighizzini, "How hard is computing the edit distance?" *Information and Computation* vol. 165, pp. 1–13, 2001.

[7] D. Sankoff, J. Kruskal (eds), *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*. CSLI Publications, 1999.

[8] R. A. Wagner, "Order-n correction for regular languages," *Communications of the ACM* vol. 17, pp. 265–268, 1974.

[9] S. Yu, "Regular languages." In G. Rozenberg, A. Salomaa (eds): *Handbook of Formal Languages, vol. I*. Springer, Berlin, 1997, pp 41–110.