Theoretical Computer Science 562 (2015) 606–620

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Using swaps and deletes to make strings match

Daniel Meister

Theoretical Computer Science, University of Trier, Germany

ARTICLE INFO

Article history:
Received 26 May 2014
Received in revised form 18 September 2014
Accepted 6 November 2014
Available online 13 November 2014
Communicated by G. Ausiello

Keywords:
String edit distance
String-to-string correction
Polynomial time

ABSTRACT

Given a source and a target string, their swap–delete–edit distance is the minimum number of interchange-consecutive-symbols and delete operations applied to the source string to make it equal the target string. We show that the swap–delete–edit distance of strings over alphabets of bounded size can be computed in polynomial time.

1. Introduction

Wagner and Fischer introduced the edit distance of strings, and considered computational aspects of determining this number [8]. In this early work, Wagner and Fischer considered three string modification – or *edit* – operations: changing a letter, and inserting and deleting a letter. The edit distance of two strings is the smallest number of operation applications to the one – *source* – string to obtain the other – *target* – string. They showed that the edit distance is polynomial-time computable, by applying a dynamic-programming approach [8]. In a subsequent paper, Lowrance and Wagner extended the list of applicable edit operations by interchanging consecutive symbols [5], which is often called a *swap*. Building up on these results, Wagner studied the complexity of computing the edit distance when restricting the allowed edit operations, showing hardness and tractability results [7]. It may be worth to mention that Wagner's considerations included weights on the edit operations, hereby modelling a favour for some edit operations over others.

In this paper, we consider the edit distance problem when restricting edit operation applications to the two operations swap and delete. Wagner showed that computing the edit distance of two arbitrary strings by applying only swap and delete is intractable [7]. A minor but important fact in Wagner's reduction is the use of an infinite set of symbols. When considering string problems in general, and when string problems appear in practical applications in particular, the involved sets of symbols are often fixed. It is therefore an interesting question of practical importance to ask whether the intractability of the edit distance problem remains when restricting the input strings to fixed alphabets. As the main result of this paper, we show that the edit distance problem when applying only swap and delete becomes tractable for fixed alphabets. This particularly shows the necessity of an infinite symbols set for achieving intractability.

The algorithm of this paper computes the unweighted swap–delete–edit distance of two strings. Since the number of delete operations is predetermined by the length difference of the source and target sting, the weighted and unweighted swap–delete–edit distance problems are computationally equivalent [7]. The algorithm itself follows a dynamic-programming approach, that has already been applied for the first algorithm by Wagner and Fischer [8]. The algorithm's

E-mail address: daniel.meister@uni-trier.de.

http://dx.doi.org/10.1016/j.tcs.2014.11.002
0304-3975/

work can be informally described as follows. The input is two strings, the *source* and *target* strings. The source string should not be of smaller length than the target string; otherwise, a solution does not exist. The source and target strings are broken into substrings, and the swap–delete–edit distance problem is solved on the substring pairs. For breaking into substrings, a fixed alphabet symbol is used, and substring pairs without that symbol are generated. The problem then is solved on input pairs for an alphabet of smaller size.

The final designed algorithm for computing the swap–delete–edit distance of two strings is an easy-to-implement algorithm, using only standard data structures and elementary techniques. The correctness proof of the algorithm, that is the main part of this paper, is technically involved but based on standard mathematical methods only. We use partial functions on the set of natural numbers to represent solutions for editing the source string into the target string. The partial functions approach strongly resembles the trace diagrams already used by Wagner and Fischer for computing the change–delete–insert–edit distance of two strings [8].

The swap–delete–edit distance problem has a special place in the group of edit distance problems allowing change, insert, delete, swap as edit operations. The swap–delete–edit distance problem, and the swap–insert–edit distance problem equivalently, is the only intractable problem among the possible problem instances [7]. It is a noteworthy side remark that the general edit distance problem, that is the change–insert–delete–swap–edit distance problem, is tractable with unit costs and under more selective cost assumptions [5]. Such a non-monotone behaviour is not often seen in computational complexity theory. A first approach to overcome the intractability of the swap–delete–edit distance problem was by Abu-Khzam et al., who devised first fixed-parameter tractable parametrised algorithms for the swap–delete–edit distance problem on input strings over arbitrary sets of input symbols [1]. They pointed out the problem of considering input strings over fixed-sized alphabets, binary alphabets even, and left the complexity status as an open problem. Spreen considered the binary alphabet case, and solved some special cases [6]. Fernau et al. resolved the complete binary alphabet case [3]. In this paper, we resolve the complexity of the swap–delete–edit distance problem for arbitrary alphabets fully.

A recently studied variant of the edit distance problem discussed here is the interchange–edit distance problem. Given two strings, the one being a permutation of the other, determine the minimum number of *interchange* operations to transform the source string into the target string. An *interchange* of two symbols interchanges the symbols. The interchange operation generalises swap. Amir et al. [2] and Kapah et al. [4] study the computational complexity of this and related edit distance problems under the unit cost and other cost models.

Organisation of the paper The main part develops the results for the dynamic-programming algorithm. Section 2 introduces terminology and presents the Splitting lemma as the main theoretical result of the paper, that is applied to breaking strings into substrings. Sections 3 and 4 present the proof of the Splitting lemma. In Section 5, our main algorithmic result is devised, that is a polynomial-time algorithm for computing the swap–delete–edit distance.

2. Definitions and the Splitting lemma

In this section, we define the terminology, notions, and main technical tools to develop our algorithm in Section 5. Appendix A contains the definitions that are not given here explicitly. Throughout the paper, k denotes a positive integer, that is considered arbitrary but fixed.

A *partial k-colouring* for $\mathbb{N}$ is a total mapping $\psi : \mathbb{N} \to \{0, 1, \ldots, k\}$. The elements of $\{1, \ldots, k\}$ are considered the k colours, and we use 0 to mean "uncoloured".

Definition 2.1. Let ψ and χ be partial k-colourings for $\mathbb{N}$.

1) The ordered pair (ψ, χ) is called a *k-colour embedding pair*.
2) A *colour-preserving embedding* for (ψ, χ) is a partial injective function φ on $\mathbb{N}$ such that for every $x \in \mathbb{N} \setminus \psi^{-1}(0)$: $\varphi(x)$ is defined and $\psi(x) = \chi(\varphi(x))$.
3) By $\mathscr{E}(\psi, \chi)$, we denote the set of the colour-preserving embeddings for (ψ, χ).

We can say that a colour-preserving embedding assigns each ψ-coloured integer to an integer of the same colour with χ. We are interested in k-colour embedding pairs (ψ, χ) for which $\mathscr{E}(\psi, \chi)$ is non-empty. Such pairs are called *satisfiable*. Consider the following iteratively defined partial function φ, for $x \in \mathbb{N} \setminus \psi^{-1}(0)$, taken in increasing order:

$$\varphi(x) =_{\text{def}} \min \chi^{-1}\big(\psi(x)\big) \setminus \big\{\varphi\big(x'\big) : x' < x\big\}.$$

We call φ the *canonical embedding* for (ψ, χ). If $\varphi(x)$ is defined for each $x \in \mathbb{N} \setminus \psi^{-1}(0)$ then φ is a colour-preserving embedding for (ψ, χ), i.e., $\varphi \in \mathscr{E}(\psi, \chi)$, and $\mathscr{E}(\psi, \chi)$ is non-empty, and (ψ, χ) is satisfiable.

A *k-colour vector* is a k-tuple over $\mathbb{N} \cup \{\aleph_0\}$. Let ψ be a partial k-colouring for $\mathbb{N}$, and let $B \subseteq \mathbb{N}$. The *colour vector* of (ψ, B) is the k-colour vector $\mathfrak{c}$ satisfying $c_j = |\{x \in B : \psi(x) = j\}|$ for $1 \le j \le k$, that counts the occurrences of the colours in ψ when restricting to the integers in B. The *colour vector* of ψ is the colour vector of $(\psi, \mathbb{N})$. Observe that uncoloured integers are not considered. Let $\mathfrak{a}$ be the colour vector of ψ. We say that a k-colour vector $\mathfrak{e}$ is a *colour vector for* ψ if $\mathfrak{e} \le \mathfrak{a}$, that means $e_j \le a_j$ for $1 \le j \le k$.