Pattern Recognition Letters 33 (2012) 842–852

Contents lists available at ScienceDirect

Pattern Recognition Letters

journal homepage: www.elsevier.com/locate/patrec

Generalized median string computation by means of string embedding in vector spaces

Xiaoyi Jiang [a,*], Jöran Wentker [a], Miquel Ferrer [b]

[a] Department of Mathematics and Computer Science, University of Münster Einsteinstrasse 62, 48149 Münster, Germany
[b] Centre de Visió per Computador, Departament de Ciències de la Computació Universitat Autònoma de Barcelona, 08193 Bellaterra, Spain

ARTICLE INFO

Article history:
Available online 23 July 2011

Keywords:
String
Generalized median
Embedding
Vector space
Lower bound

ABSTRACT

In structural pattern recognition the median string has been established as a useful tool to represent a set of strings. However, its exact computation is complex and of high computational burden. In this paper we propose a new approach for the computation of median string based on string embedding. Strings are embedded into a vector space and the median is computed in the vector domain. We apply three different inverse transformations to go from the vector domain back to the string domain in order to obtain a final approximation of the median string. All of them are based on the weighted mean of a pair of strings. Experiments show that we succeed to compute good approximations of the median string.

1. Introduction

Strings are one of the fundamental representation formalisms in structural pattern recognition (Bunke and Sanfeliu, 1990). Using a sequence of symbols rather than a vector of features often has some advantages. For example, we have the freedom of using a variable number of symbols in a string to model the varying complexity of patterns. Numerous applications have been found in a broad range of fields including computer vision, speech recognition, and molecular biology (Sankoff and Kruskal, 1983).

A large number of operations and algorithms have been proposed to deal with strings (Gusfield, 1997; Sankoff and Kruskal, 1983; Stephen, 1994). Some of them are inherent to the special nature of strings such as the shortest common superstring and the longest common substring, while others are adapted from other domains.

In data mining, clustering, and machine learning, a typical task is to represent a set of (similar) objects by means of a single prototype. The median concept is useful to obtain such a prototype in various contexts. It represents a fundamental quantity in statistics. In sensor fusion, multisensory measurements of some quantity are averaged to produce the best estimate. Averaging the results of several classifiers is used in multiple classifier systems in order to achieve more reliable classifications. Other applications of the median concept have been demonstrated in dealing with 2D shapes, binary feature maps, 3D rotation, geometric features (points, lines, or 3D frames), brain models, anatomical structures, facial images, and more; see (Jiang and Bunke, 2010) for an overview.

This paper deals with the generalized median in the domain of strings (Jiang et al., 2004). Given a set of strings, the median string is defined as a string that has the minimum sum of distances to all strings in the set. Despite of this simple definition, the computation of generalized median strings is an $\mathcal{NP}$-hard problem. This leads to the need of approximate approaches in practice. In this paper we address the problem of median string computation from a new point of view based on three main pillars: the string embedding in vector spaces (Spillmann et al., 2006), the string edit distance (Wagner and Fischer, 1974), and the weighted mean of a pair of strings (Bunke et al., 2002).

The rest of the paper is organized as follows. In the next section we define the basic notations, introduce in detail the concept of median string, and discuss related works. In Section 3 the proposed method for median computation is described. Section 4 reports a number of experiments. Finally, in Section 5 we draw conclusions and point out potential future work.

2. Generalized median string

Assuming an alphabet Σ of symbols, a string x is simply a sequence of symbols from Σ, i.e. $x = x_1 x_2 \cdots x_n$, where $x_i \in \Sigma$ for $i = 1,\ldots,n$. Given the space U of all strings over Σ, we need a distance function $d(p,q)$ to measure the dissimilarity between two strings p, $q \in U$. Let S be a set of N strings from U. The essential information of S is captured by a string $\bar{p} \in U$ that minimizes the sum of distances (SOD) of $\bar{p}$ to all strings from S, also called the consensus error $E_S(p)$:

$$\bar{p} = \arg\min_{p \in U} E_S(p), \quad \text{where} \quad E_S(p) = \sum_{q \in S} d(p,q).$$

* Corresponding author.
E-mail address: xjiang@math.uni-muenster.de (X. Jiang).

0167-8655/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.patrec.2011.07.006

The string $\bar{p}$ is called a *generalized median* of S. If the search is constrained to the given set S, the resulting string

$$\hat{p} = \arg\min_{p \in S} E_S(p)$$

is called a *set median* of S. Note that for a given set S, neither the generalized median nor the set median is necessarily unique. This definition was introduced by Kohonen (1985). Since then, different terminology has been used in the literature. In (Gusfield, 1997) the set median string and the generalized median string are termed *center string* and *Steiner string*, respectively. In (Lopresti and Zhou, 1997) the generalized median string is called *consensus sequence*.

2.1. String distance functions

Several string distance functions have been proposed in the literature. The most popular one is doubtlessly the Levenshtein edit distance (Wagner and Fischer, 1974). Let $x = x_1x_2\cdots x_n$ and $y = y_1y_2\cdots y_m$ be two strings over Σ. The Levenshtein edit distance $d(x,y)$ is defined in terms of elementary edit operations, which are required to transform x into y. Usually, three different types of edit operations are considered, namely (1) substitution of a symbol $a \in x$ by a symbol $b \in y$, $a \neq b$, (2) insertion of a symbol $a \in \Sigma$ in y, and (3) deletion of a symbol $a \in \Sigma$ in x. Symbolically, we write $a \to b$ for a substitution, $\epsilon \to a$ for an insertion (ϵ is the empty string), and $a \to \epsilon$ for a deletion. To model the fact that some distortions may be more likely than others, costs of edit operations, $c(a \to b)$, $c(\epsilon \to a)$, and $c(a \to \epsilon)$, are introduced. Let $s = l_1l_2\cdots l_k$ be a sequence of edit operations transforming x into y. We define the cost of this sequence by $c(s) = \sum_{i=1}^{k} c(l_i)$. Given two strings x and y, the Levenshtein edit distance is given by

$$d(x,y) = \min\{c(s) | s \text{ is a sequence of edit operations transforming } x \text{ into } y\}.$$

In (Wagner and Fischer, 1974) an algorithm is proposed to compute the Levenshtein edit distance by means of dynamic programming. Other algorithms are discussed in (Gusfield, 1997; Sankoff and Kruskal, 1983; Stephen, 1994).

Further string distance functions are known from the literature, for instance, normalized edit distance (Marzal and Vidal, 1993; Vidal et al., 1995), maximum posterior probability distance (Kohonen, 1985), feature distance (Kohonen, 1985), stochastic edit distance (Olivares-Rodriguez and Oncina, 2008), and others (Fred and Leitao, 1998). However, the Levenshtein edit distance is by far the most popular one and will thus be used in our work.

2.2. Computational complexity

Independent of the distance function, we can always find the set median of N strings by means of $\frac{1}{2}N(N-1)$ pairwise distance computations. This computational burden can be further reduced by making use of special properties of the distance function (e.g. metric) or resorting to approximate procedures (Juan and Vidal, 1998; Mico and Oncina, 2001).

Compared to set median strings, the computation of generalized median strings represents a much more demanding task. Indeed, it was proved (de la Higuera and Casacuberta, 2000) that the generalized median string problem is $\mathcal{NP}$-hard for unbounded alphabets. The work (Sim and Park, 2003) proved that the problem is $\mathcal{NP}$-hard for finite alphabets with a metric distance matrix. A more recent result (Nicolas and Rivals, 2005) demonstrated the $\mathcal{NP}$-hardness under Levenshtein distance for bounded and even binary alphabets. Another result comes from computational biology. The optimal evolutionary tree problem there turns out to be equivalent to the problem of computing generalized median strings if the tree structure is a star (a tree with $n+1$ nodes, n of them being leaves). In (Wang and Jiang, 1994) it was proved that in this particular case the optimal evolutionary tree problem is $\mathcal{NP}$-hard. The distance function used is problem dependent and does not even satisfy the triangle inequality. All these theoretical results indicate the inherent complexity in finding generalized median strings.

The generalized median is a more general concept and usually a better representation of the given strings than the set median. From a practical point of view, the set median can be regarded as an approximate solution of the generalized median. Indeed, it can be shown that the set median is a suboptimal solution with approximation factor 2 (Gusfield, 1997). As such, it may thus serve as the start for an iterative refinement process to find more accurate approximations (Martinez-Hinarejos et al., 2000).

2.3. Related works

An algorithm for the exact computation of generalized median strings under the Levenshtein distance is given in (Kruskal, 1983), which is based on dynamic programming in an N-dimensional array, similarly to string edit distance computation. If the strings of the input set S are of length $O(n)$, both the time and space complexity amounts to $O(n^N)$ in this case.

Despite of its mathematical elegance the exact algorithm above is impractical because of the exponential complexity. There have been efforts to shorten the computation time using heuristics or domain-specific knowledge. Such an approach from Lopresti and Zhou (1997) assumes that the strings of S be quite similar. In this case the optimal dynamic programming path must be close to the main diagonal in the distance table. Therefore, only part of the N-dimensional table needs to be explored. Its asymptotic time complexity is $O(nk^N)$. While this remains exponential, k is typically much smaller than n, resulting in a substantial speedup compared to the full search of the original algorithm (Kruskal, 1983).

We may also use any domain-specific knowledge to limit the search space. An example is the approach in the context of classifier combination for handwritten sentence recognition (Marti and Bunke, 2001). An ensemble of classifiers provide multiple classification results of a scanned text. Then, the consensus string is expected to yield the best overall recognition performance. The input strings from the individual classifiers are associated with additional information of position, i.e. the location of each individual word in a sequence of handwritten words. Obviously, it is very unlikely that a word at the beginning of a sequence corresponds to a word at the end of another sequence. More generally, only words at a similar position in the text image are meaningful candidates for being matched to each other. The work (Marti and Bunke, 2001) makes use of this observation to exclude a large portion of the full N-dimensional search space from consideration.

Because of the $\mathcal{NP}$-hardness of generalized median string computation, efforts have been undertaken to develop approximate approaches, which provide suboptimal solutions in reasonable time. In (Casacuberta and de Antoni, 1997) a greedy algorithm constructs an approximate generalized median string symbol by symbol, starting from an empty string. A slightly improved version can be found in (Kruzslicz, 1999).

The set median represents an approximation of the generalized median string. The greedy algorithms also give approximate solutions. An approximate solution can be further improved by an iterative process of systematic perturbations. This idea was first suggested in (Kohonen, 1985), but no algorithmic details are specified there. A concrete algorithm for realizing systematic perturbations is given in (Martinez-Hinarejos et al., 2000).

There are also median string algorithms developed for other distance functions, for instance, the stochastic edit distance (Olivares-Rodriguez and Oncina, 2008). In a dynamic context we are faced with the situation of a steady arrival of new data items,

represented by strings. In (Jiang et al., 2003) a dynamic approach is proposed, in which the updated median string is computed in an incremental manner.

3. Approximative algorithm based on string embedding in vector spaces

In the last section we have shown that the computation of generalized median string is a rather complex task. The exact method (Kruskal, 1983) and even the one with constrained search space (Lopresti and Zhou, 1997) have an exponential complexity and thus cannot be the choice in practice. Approximative approaches like greedy methods (Casacuberta and de Antoni, 1997; Kruzslicz, 1999) have a high risk of making wrong local decision at each symbol. In addition, their computational time grows with the alphabet size.

In this section we present a novel global approach to approximately computing the median string based on string embedding in a vector space. It is inspired by Ferrer et al. (2010, 2011) and consists of three main steps. Given a set $S = \{s_1, s_2, \ldots, s_n\}$ of n strings,

- *Step 1:* embed every string from S into the m-dimensional space ($m \leqslant n$) of real numbers, i.e. each string becomes a point in $\Re^m$.
- *Step 2:* compute the median of the vectors obtained in the previous step based on the Euclidean distance.
- *Step 3:* go from the vector space back to the string domain, converting the median vector into a string. The resulting string is taken as the median string of S.

These three steps are depicted in Fig. 1. String embedding has been investigated in (Spillmann et al., 2006) in the context of structural pattern classification. The focus of our work is rather different. In particular, we need to carefully consider the final step of inverse transformation back to the string space. In the following subsections, these three main steps will be explained in detail.

3.1. String embedding in vector spaces

Given a set of strings $S = \{s_1, s_2, \ldots, s_n\}$, we select a subset of m prototypes from S: $P = \{p_1, p_2, \ldots, p_m\} \subseteq S$ ($m \leqslant n$). Then, the dissimilarity (distance) between a given string $s \in S$ and every prototype $p \in P$ is computed. This leads to m distance values, $d_1, d_2, \ldots, d_m$, where $d_i = d(s, p_i)$. These distances can be arranged in a vector $(d_1, d_2, \ldots, d_m)$. In this way, we can transform any string of S into an m-dimensional vector using the prototype set P. More formally, this embedding procedure can be defined as follows:

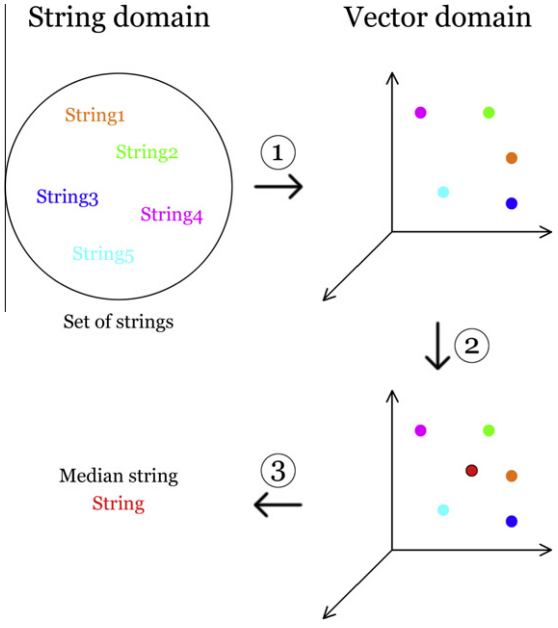

Fig. 1. Overview of the approximate procedure for median string computation.

Definition 1. Given a set of strings $S = \{s_1, s_2, \ldots, s_n\}$ and a set of prototypes $P = \{p_1, p_2, \ldots, p_m\} \subseteq S$ with $m \leqslant n$, the string embedding:

$$\Psi : S \to \Re^m$$

is defined as the function:

$$\Psi(s) = (d(s, p_1), d(s, p_2), \ldots, d(s, p_m)),$$

where $d(s, p_i)$ is a string dissimilarity measure between the string s and the ith prototype p_i (in this paper the string edit distance).

The size m of the prototype set P is a parameter of the embedding process. In our experimental results reported in Section 4, it will be shown that m can be chosen to be a rather small fraction of the original size n without distorting the overall quality of the computed median string.

Given a size m of the prototype set, we need a strategy for selecting m samples from the input set S. According to Spillmann et al. (2006), a good selection algorithm should consider three criteria: avoid redundancies; the prototypes should be uniformly distributed over the whole set of patterns (in order to include as much structural information as possible in the prototypes); and disregard outliers. Based on these criteria, four prototype selection algorithms were suggested:

- Center prototype selector. The first prototype is the set median string of S. Every further prototype is the set median of all remaining samples.
- Border prototype selector. It acts just contrary to the center prototype selector and selects prototypes from the border based on marginal strings.
- Spanning prototype selector. The first prototype is the set median string of S. Every further prototype is the string with the largest distance to the set of previously selected prototypes.
- K-medians prototype selector. It finds m clusters in the given set of data using the k-mean clustering algorithm and declares each cluster center (set median of each cluster) to be a prototype.

Not all the four selection methods satisfy the three requirements. The center prototype selector tends to produce redundant prototypes, while the border and spanning prototype selector are not particularly robust against outliers. In addition, the center and border prototype selector tend to violate the requirement of uniform distribution. The k-medians prototype selector is a good choice. Similar strings are represented by the same prototype and redundant prototypes are thus mostly avoided. The selected prototypes are expected to be evenly spread over the whole set of data. Furthermore, outliers usually are not positioned in the center of a k-medians cluster and thus the chance for them to be selected as a prototype is small. The readers are referred to Spillmann et al. (2006) for further discussion and visual comparison of these selection methods. As will be shown later in Section 4, the k-medians prototype selector also turns out to be the choice for our task. Note that although a random prototype selection is computationally cheaper than the methods discussed above, it satisfies all the three criteria not particularly well.

3.2. Computation of the median vector

Once all the strings have been embedded in the vector space, the median vector is computed. To this end we use the concept of Euclidean median. Given a set $X = \{x_1, x_2, \ldots, x_n\}$ of n points with $x_i \in \Re^m$ for $i = 1, 2, \ldots, n$, the Euclidean median is defined as

$$\arg\min_{y \in \Re^m} \sum_{i=1}^{n} \|x_i - y\|,$$

where $\|x_i - y\|$ denotes the Euclidean distance between the points $x_i,\ y \in \Re^m$. That is, the Euclidean median is a point $y \in \Re^m$ that minimizes the sum of the Euclidean distances between itself and all the points in X. It corresponds with the definition of the median string, but in the vector domain.

This problem is the classical Weber problem from Mathematical Economics and has been studied for long time; see (Wesolowsky, 1993) for a survey. It is well-known that if the data points x_i are not collinear, the objective is strictly convex and therefore has a unique minimum. In the collinear case at least one of the points x_i is optimal and it can be found in linear time (Chandrasekaran and Tamir, 1989).

Unfortunately, the Euclidean median cannot be calculated in a straightforward way. Indeed, the exact location of the Euclidean median is not solvable by radicals over the field of rationals; i.e. for $n \geqslant 5$ there exists no exact algorithm under models of computation where the root of an algebraic equation is obtained using arithmetic operations and the extraction of kth roots (Bajaj, 1988). No algorithm is known for exactly computing the Euclidean median in polynomial time, nor has the problem been shown to be $\mathcal{NP}$-hard (Hakimi, 2000). In this work we will use the most common approximate algorithm for the computation of the Euclidean median, that is the Weiszfeld algorithm (Weiszfeld, 1937). It is an iterative procedure that converges to the Euclidean median. To this end, the algorithm first chooses an initial estimate solution y (this initial solution is often chosen randomly). Then, the algorithm defines a set of weights that are inversely proportional to the distances from the current estimate to the samples, and creates a new estimate that is the weighted average of the samples according to these weights (Chandrasekaran and Tamir, 1989):

$$y_{k+1} = \sum_{i=1}^{n} \frac{x_i}{\|x_i - y_k\|} \Bigg/ \sum_{i=1}^{n} \frac{1}{\|x_i - y_k\|}.$$

The algorithm may finish either when a predefined number of iterations is reached or under some other criteria, for instance, when the difference between the current estimate and the previous one is less than some predefined threshold.

3.3. Inverse transformation to string domain

The last step in order to obtain the median string is to transform the Euclidean median from the vector space back into a string. Such a string will be considered as an approximation of the median string of the set S. In the following we summarize three different strategies for this purpose. The foundation of all these heuristic approaches is the concept of weighted mean of a pair of strings (Bunke et al., 2002) and the edit path between two given strings.

Definition 2. Let x and z be strings. The weighted mean of x and z is a string y such that

$$d(x, y) = a; \quad 0 \leqslant a \leqslant d(x, z),$$
$$d(x, z) = a + d(y, z),$$

The computed string y is somewhere between the strings x and z controlled by the "weight" parameter a. Conceptually, this corresponds to the weighted mean of two numbers and for this reason y is named weighted mean string.

Given the optimal edit sequence from x to z, the fundamental idea behind the algorithm from Bunke et al. (2002) for constructing y that has the desired fraction a of the total distance $d(x,z)$ is as follows. Select a subset of all the edit operations so that the sum of their costs (approximately) amounts to a. The selected subset is applied to x (the remaining cost $d(x,z) - a$ is then realized by the remaining edit operations for obtaining z from y). Note that the weighted mean is generally not unique. However, it can be made unique if we select a subset of total cost a based on the optimal edit operations between x and z in their natural order from the edit distance computation.

Linear interpolation. Given the m-dimensional points, each of them corresponding to a string in S, and the Euclidean median vector v, this simple strategy (Ferrer et al., 2009) uses two auxiliary points in the vector space; see Fig. 2. We choose the two closest points v_1 and v_2 of v (Note that we know the corresponding strings of these two points). Then, we compute the median vector v' of v_1 and v_2. This point v' is used to obtain the approximate median string. To this end, we first compute the distance of v_1 and v_2 to v' and then, with these distances, we apply the weighted mean of a pair of strings between strings s_1 and s_2 (which correspond to v_1 and v_2, respectively) to obtain s', the approximate median string.

Triangulation. This strategy (Ferrer et al., 2010) first selects the three closest points to the Euclidean median (v_1, v_2, and v_3; see Fig. 2). Then, the median vector v' of these three points is computed. Note that v' is in the plane formed by v_1, v_2 and v_3. The basic idea here is to compute a string corresponding to v'. This is done in two steps. First, we arbitrarily choose two out of the three points. Without loss of generality we can assume that v_1 and v_2 are selected and we then project the remaining point v_3 onto the line joining v_1 and v_2 through v'. In this way, we obtain a point v_i in between v_1 and v_2. Using the two-point based strategy from linear interpolation, we can determine a string corresponding to v_i. A second use of the same strategy delivers a string corresponding to v', which is regarded the approximation of the generalized median of S.

Recursive approach. In essential this strategy is an extension of the triangulation approach. In the following we give a brief outline and the full details can be found in (Ferrer et al., 2011). The main idea here is to expand the triangulation strategy to any number of points less than or equal to n. The use of all n points in S implies an $(n-1)$-dimensional geometric body. For $n = 4$, this is simply a tetrahedron; see Fig. 3. The point v' represents the Euclidean median of all four points. Using the fact that the Euclidean median always lies inside the geometric body (here the tetrahedron), the point v_4 can be chosen (without loss of generality) to be projected

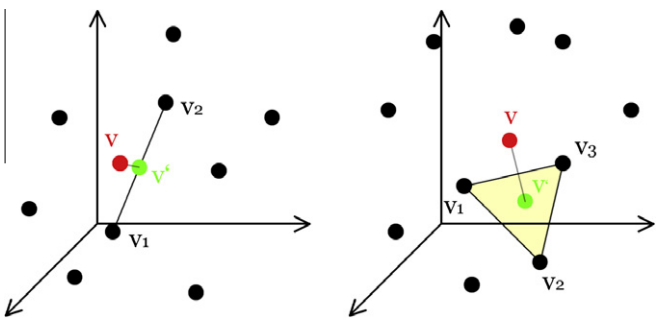

Fig. 2. Illustration of inverse transformation: two-point based linear interpolation (left) and three-point based triangulation (right).

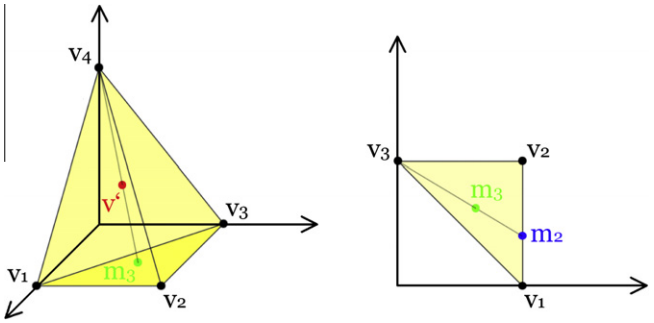

Fig. 3. Illustration of inverse transformation: recursive approach.

onto an $(n-2)$-dimensional body. In this case it is a triangle and therefore the triangulation strategy can be applied; Fig. 3 (right). So, after the projection points m_3 and m_2 are computed recursively, the weighted means have to be determined in the reverse direction. This also includes the Euclidean median v, which belongs to the generalized median string. Note that only the use of all n points prevents an approximation of v respectively the generalized median string.

3.4. Discussion on the approximations

The approximate embedding procedure is composed of three steps: the string embedding into a vector space, the median vector computation, and the inverse transformation to the string domain. Each of these steps introduces some kind of approximation to the final solution. In the first step, a certain amount of error is due to the use of $m(<n)$ prototypes. Also the median vector computation is not free of error, since the Weiszfeld method obtains approximations for the median vector only. In case of triangulation, for instance, this factor may lead to choosing three points that might not be the best points to go back to the string domain. Finally, when the weighted mean between two points is computed, the string edit path is composed of a set of discrete steps, each of them with its own cost. In the return to the string domain, the percentage of distance needed to obtain the weighted mean of a pair of strings may fall in between two of these edit operations. Since we choose only one of them, small errors may also be introduced in this step.

Although all these approximations may seem severe to obtain good medians, in the next section we will show that the embedding method is able to obtain reasonably good approximations for the median string. But, at the same time, these well defined entry points of approximation can be used to improve the medians obtained. That is, each of these three steps can be further studied and other options for each of them can be proposed in order to minimize the error introduced and obtain better medians.

4. Experimental results

We have conducted an experimental evaluation of the proposed approach. In the following we describe the generation of test data sets, the assessment of median quality, and the results of several test series.

4.1. Generation of test data sets

For test purpose we have used data sets of synthetic strings based on the German alphabet of 59 letters and 10 digits. Given an initial string, a desired number n of distorted versions are generated. This generation process is controlled by the following parameters:

- Each symbol of the initial string is distorted with a probability p.
- If a symbol should be distorted, then the three basic operations substitution, deletion, and insertion are chosen by probability p_s, p_d, and p_i, respectively.

Due to the non-deterministic nature of the generation process, we always generate 100 data sets for one initial string and report the average performance measures.

4.2. Assessment of median quality

We consider two means of evaluating the quality of the obtained median strings. First, as the set median string is the string belonging to the input set with minimum SOD, it is a good reference to evaluate the generalized median string quality by comparing its SOD with the SOD of the set median.

A second means is based on a lower bound for the generalized median problem. Given a suboptimal solution $\tilde{p}$ for a set of strings $S = \{s_1, s_2, \ldots, s_n\}$, the following relationship holds:

$$\mathrm{SOD}(\tilde{p}) = \sum_{i=1}^{n} d(\tilde{p}, s_i) \geqslant \sum_{i=1}^{n} d(\bar{p}, s_i) = \mathrm{SOD}(\bar{p}),$$

where $\bar{p}$ stands for the true generalized median string. The quality of $\tilde{p}$ can be measured by $\mathrm{SOD}(\tilde{p})/\ \mathrm{SOD}(\bar{p})$. Since $\bar{p}$ and thus $\mathrm{SOD}(\bar{p})$ are unknown in general, we have to resort to a lower bound Γ with

$$0 \leqslant \Gamma \leqslant \mathrm{SOD}(\bar{p}) \leqslant \mathrm{SOD}(\tilde{p})$$

and measure the quality of $\tilde{p}$ by $\Delta = \ \mathrm{SOD}(\tilde{p})/\Gamma(\geqslant 1)$ instead.

We require Γ to be as close to $\mathrm{SOD}(\bar{p})$ as possible. In (Jiang and Bunke, 2002) a lower bound based on linear programming is proposed for metric spaces. Assuming a metric distance function $d(x,y)$, the lower bound for the generalized median problem is specified by the solution Γ of the following linear program:

$$\begin{aligned}
&\text{minimize } x_1 + x_2 + \cdots + x_n \text{ subject to}\\
&\forall i,j \in \{1,2,\ldots,n\}, \quad i \neq j, \quad \begin{cases} x_i + x_j \geqslant d(s_i,s_j), \\ x_i + d(s_i,s_j) \geqslant x_j, \\ x_j + d(s_i,s_j) \geqslant x_i, \end{cases}\\
&\forall i \in \{1,2,\ldots,n\}, \quad x_i \geqslant 0.
\end{aligned}$$

Note that the three conditions behind the brace should be satisfied simultaneously. Given a suboptimal solution $\tilde{p}$ and the computed lower bound, the deviation Δ can thus give a hint of the absolute accuracy of $\tilde{p}$. In particular, if $\Delta \approx 1$, then it can be safely claimed that there is hardly room for further improvement (for the particular data set at hand).

In (Jiang et al., 2001) another graph-based lower bound is proposed. It is however believed that this lower bound is equivalent to the linear program one Γ from Jiang and Bunke (2002). Therefore, we apply Γ in our work.

It is important to note that although an initial string is applied to generate a test data set, that string may not necessarily be the ground truth (i.e. the true solution of SOD minimization) for the computed generalized median. This is particularly true in case of large distortions. Therefore, we do not use the initial string as test reference, but instead the lower bound Γ.

4.3. Test series

We defined a standard set of parameter values as follows:

- $n = 40$: Number of strings in input set S for the generalized median computation.
- subset size = 25%: this fraction of strings from S are used as prototypes for the embedding process. This parameter indirectly determines the number of prototypes, m.
- $p = 12\%$: distortion probability for each symbol.
- $p_s = 87\%$, $p_d = 9\%$, $p_i = 4\%$: probability of the three basic operations substitution, deletion, and insertion for each symbol, which should be distorted. For a more realistic modeling of typical OCR errors, we defined five similar symbols for substituting each symbol of the alphabet. For instance, the symbol "*i*" should be replaced by "*l*", "*I*", "*j*", etc.
- $c(a \to b) = c(\epsilon \to a) = c(a \to \epsilon) = 1$: equal costs for the three fundamental edit operations.
- The k-medians prototype selector is defined as default.

Then, we varied some of these parameters in turn (by keeping the other parameters unchanged) to investigate the algorithm behavior. These parameter values are certainly subjective to some extent. However, they can be justified from typical real situations. For instance, we expect to encounter mostly reading errors (substitution) in OCR, then followed by deletion and insertion, as reflected by the much higher probability of substitution. The cost 1 for the three fundamental edit operations is popular in the literature. The k-medians prototype selector is not only the favorite from the previous study (Spillmann et al., 2006), but also validated in our own experiments (see below).

Prototype selection. Fig. 4 shows the performance of the four prototype selection methods for data sets generated from the initial string "Andalusien" as a function of the distortion probability p. We consider p up to about 20% as reasonable for practical situations. Thus, the k-medians selector turns out to be the most effective one. The center selector may be a good choice for larger distortion levels. Due to the large number of set median computations, however, this variant of prototype selection is slower compared to the k-medians (in our experiments typically a factor of 3–4). Overall, the k-medians is a good compromise. The same test has been run for many strings of various lengths and similar behavior can be observed in all test cases. For this reason k-medians is fixed for other test series.

The following test series compare the different inverse transformation methods (linear interpolation, triangulation, and recursive approach) by varying one of the standard parameters each time. In order to increase the test basis the following three tests include strings of various lengths. Since similar performance was observed for different strings of the same length, we decided to include one string only for each length. Five lengths (8, 10, 12, 14, 16) were considered and the corresponding strings were "Scotland", "Birmingham", "Philadelphia", "TristanDaCunha", and "WesternPatagonia". For each string, the prespecified number n of distorted variants are generated and their median string is computed. This procedure is repeated 100 times and the average performance is measured.

Number of strings. This test series investigates the performance as a function of the size of input set S; see Fig. 5. Clearly, the recursive approach performs best. From about $n = 25$ on the performance tends to approach the lower bound, thus the unknown ground truth. In contrast the set median solution can in no way compete with the computed generalized median, even against linear interpolation and triangulation. The measures given in Fig. 5 are averaged values over 100 times. To give the readers an impression of the individual behavior, we also list the standard deviation of SOD/(Lower Bound) for the recursive method: 0.274, 0.286, 0.269, 0.277, 0.258, 0.238, 0.234, 0.228, 0.224, 0.233, 0.168, 0.158, 0.148, and 0.123 (corresponding to the 14 different numbers of strings).

Subset size. This number is another source of potential errors introduced by the embedding process (see Section 3.4). Fig. 6 demonstrates again the superior performance of the recursive approach. This performance can even be achieved with a small fraction of all input strings as prototypes only. Here the constant discrepance between set median and the recursive solution is remarkable.

Symbol distortion level. The robustness against distortions in the input strings is studied in Fig. 7. Up to 20% the recursive approach dominates the other methods. Between 20% and 40% the linear interpolation delivers the best results. This may be interpreted by its robustness against outliers. Indeed, linear interpolation is based on two strings near the median vector only and thus has the potential of avoiding outliers. Obviously, the same argument applies to the case of triangulation as well and this explains the similar performance of triangulation in Fig. 7.

String length. Fig. 8 shows the performance for various string lengths. Again the recursive approach clearly outperforms the other methods and achieves results very near to the unknown ground truth (about 5% over the lower bound only).

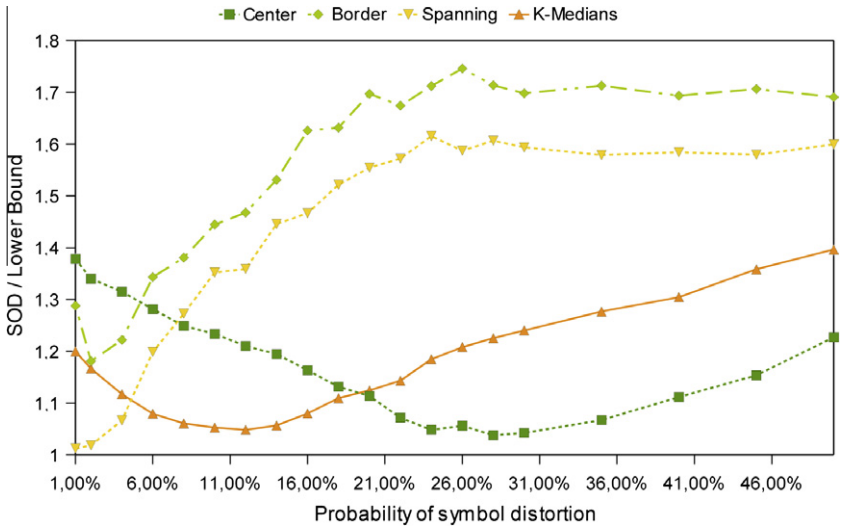

Fig. 4. Performance of four prototype selectors.

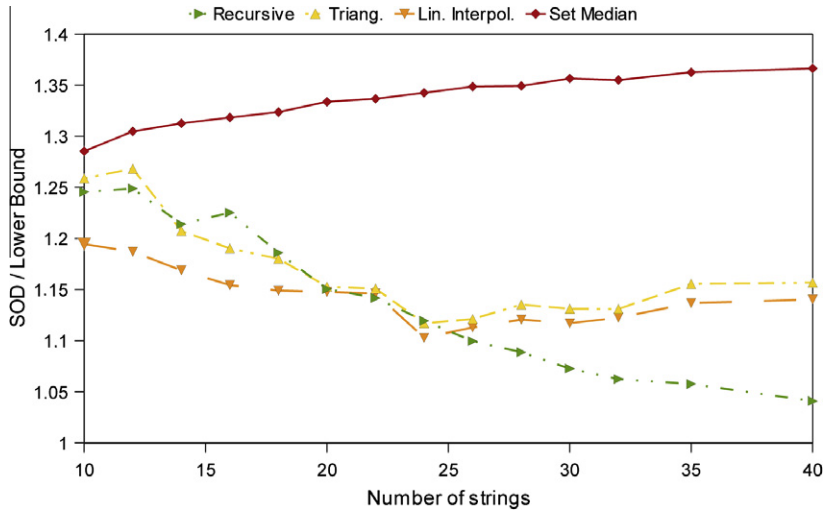

Fig. 5. Performance as a function of input size.

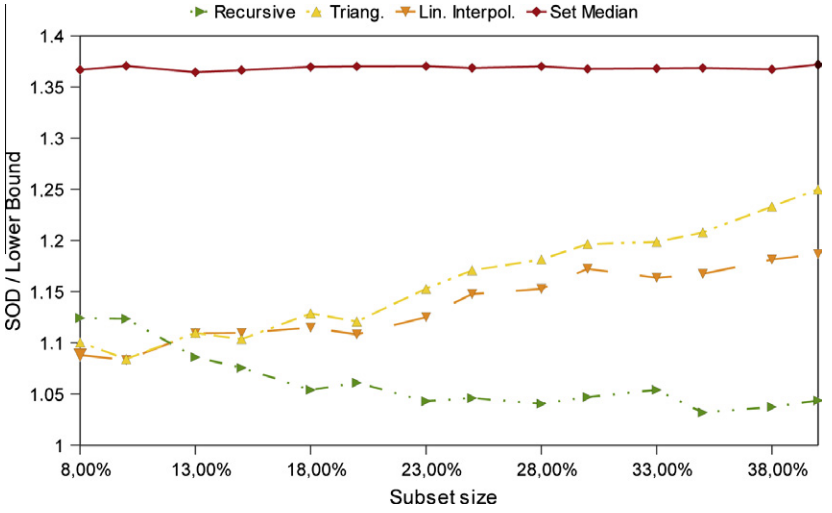

Fig. 6. Performance as a function of number of prototypes.

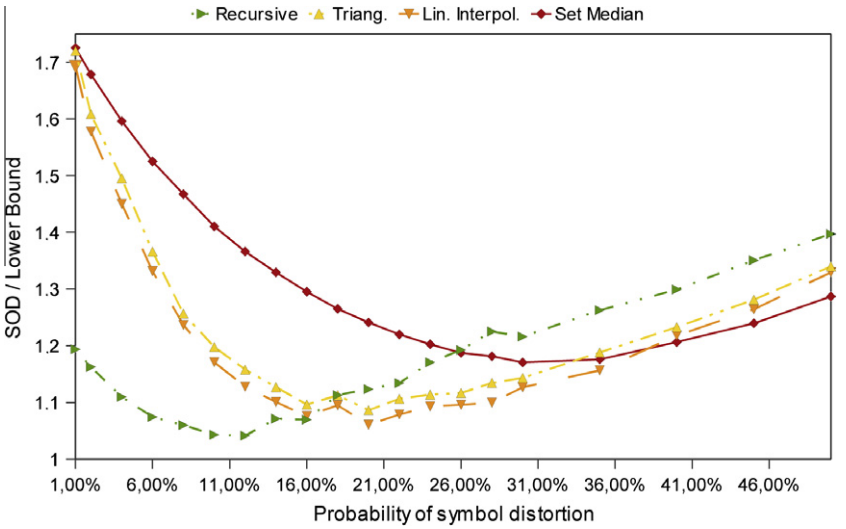

Fig. 7. Performance as a function of distortion level.

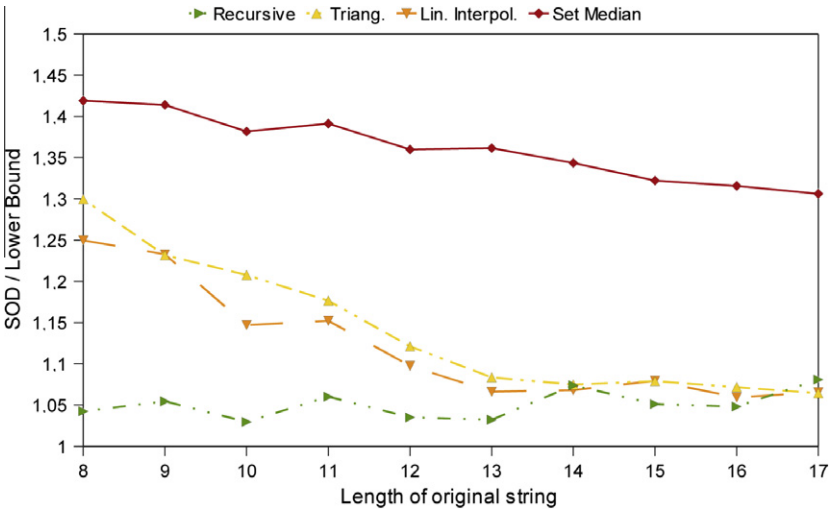

Fig. 8. Performance as a function of string length.

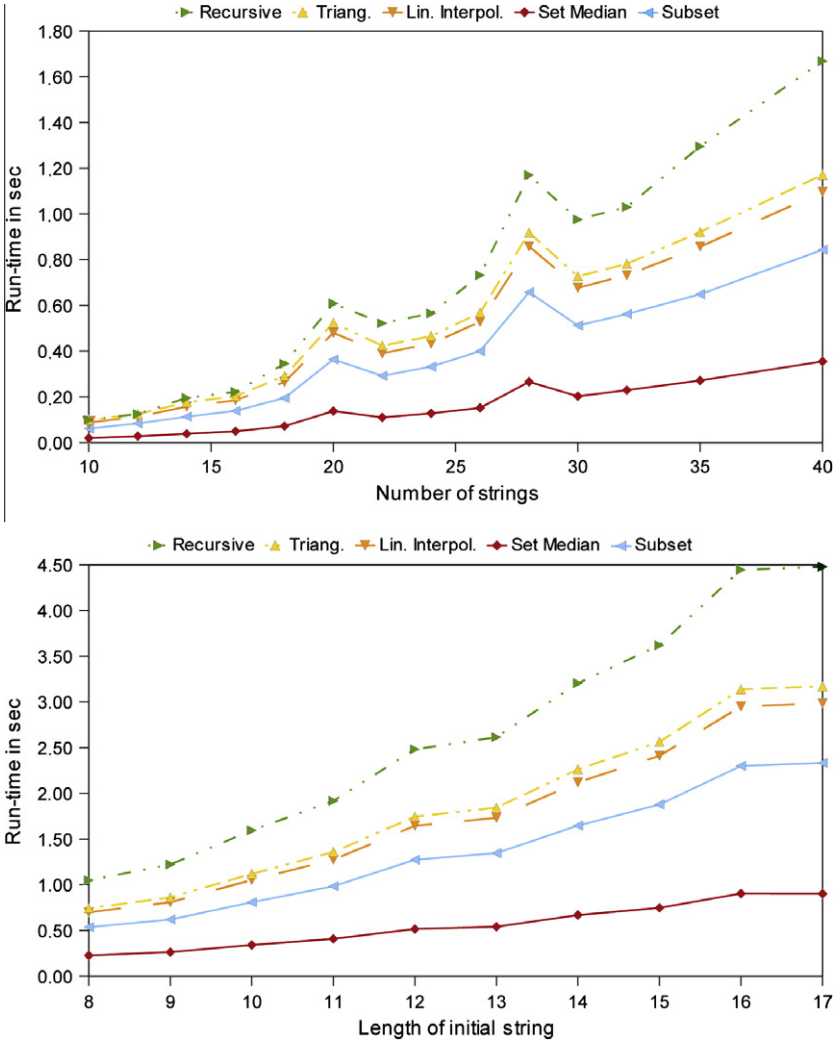

Fig. 9. Computational time for test series *number of strings* (top) and *string length* (bottom).

4.4. Computational time

The experiments were performed on a 2.66 GHz quad-core PC. The program was written in Matlab, which was restricted to operate on one core only. Fig. 9 shows the computational time for the test series *number* of strings and *string* length. A large part of the time is needed by prototype selection (see the curve in blue). Generally, the recursive approach is more costly than linear

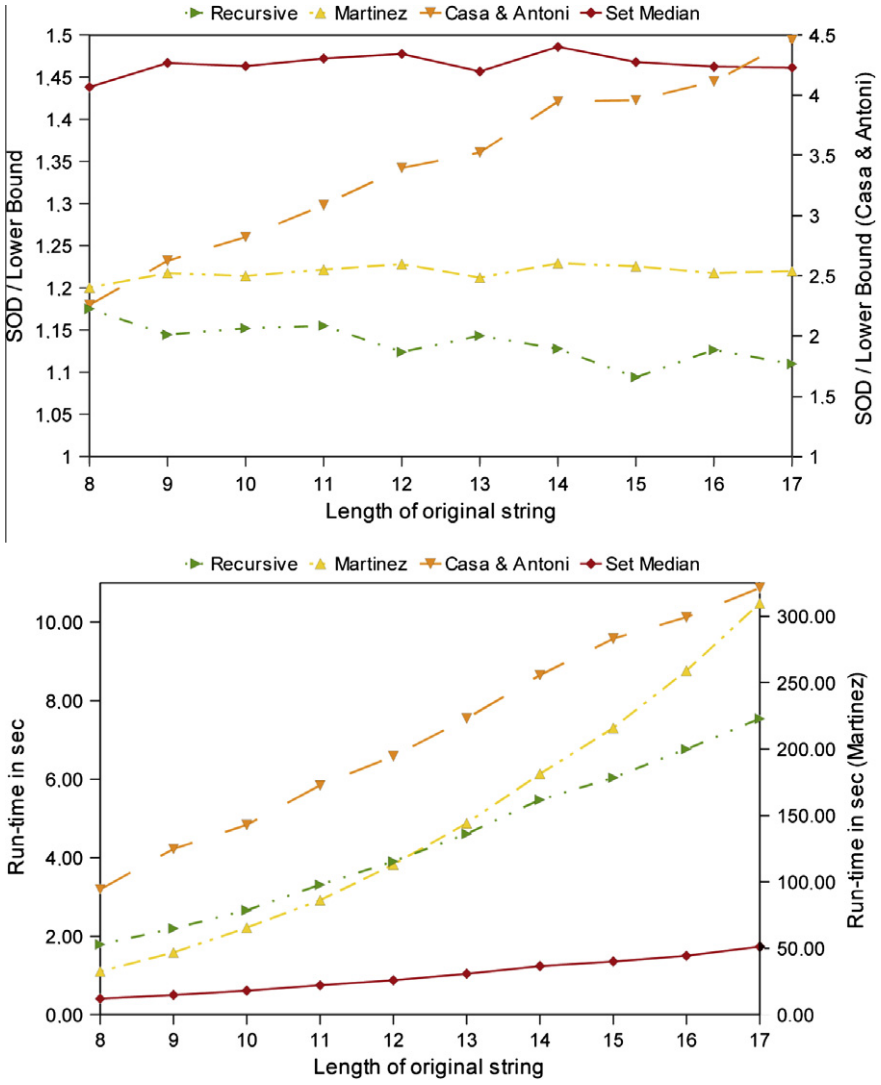

Fig. 10. Comparative study: performance (top) and computational time (bottom) as a function of string length. Please note the different scale for (Casa & Antoni) and (Martinez), respectively, on the right side.

interpolation and triangulation. But the additional time may be justifiable because of the improved performance. Currently, no efforts were made in the Matlab implementation in terms of efficiency. There is certainly substantial potential here.

4.5. Discussions

The embedding method with all its variants almost consistently substantially beats the set median in terms of SOD. This is a clear sign of quality of this novel method for generalized string computation. Absolutely speaking, the achieved quality can be as low as less than 5% over the lower bound (thus in a close neighborhood of the unknown ground truth).

The experimental results reveal good compromise decisions. The k-medians can be the choice for prototype selection. The recursive approach achieved in average the best performance in terms of median quality. However, there is no absolute winner. For instance, Fig. 4 also shows that the Center prototype selection may be helpful in case of large distortion levels (if we disregard its higher computational burden). The situation in Fig. 7 is similar. Starting from about 20% distortion level, the recursive approach may not be the first choice. This observation is typical in pattern recognition. There is simply no classifier (or other decision procedure), which can guarantee the best performance in all cases. A reasonable approach is thus decision fusion, which, for example, has turned out to be extraordinary powerful in pattern classification. The same fundamental idea can be applied in our case as well. We may run more than one method and select the best result in terms of SOD.

The experimental results reported above are based on the German alphabet of 59 letters and 10 digits. For other languages with a comparable alphabet size similar performance can be expected. However, it is difficult to make general statements about the influence of the alphabet size. Only a performance evaluation in a concrete problem case can give a reliable assessment of our algorithm with regard to this parameter.

4.6. Comparative study

We have conducted a comparative study with Casacuberta and de Antoni (1997) and Martinez-Hinarejos et al. (2000). Starting from an empty string, the greedy algorithm (Casacuberta and de Antoni, 1997) constructs an approximate generalized median string $\bar{p}$ symbol by symbol. When we are going to generate the lth symbol a_l ($l \geqslant 1$), the substring $a_1 \cdots a_{l-1}$ (ϵ for $l = 1$) has already been determined. Then, each symbol from Σ is considered as a candidate for a_l. All the candidates are evaluated by means of SOD and

the final decision of a_l is made by selecting the best candidate. The process is continued until a termination criterion is fulfilled. In (Martinez-Hinarejos et al., 2000) an approximate solution $\bar{p}$, e.g. the set median, is further improved by an iterative process of systematic perturbations. For each position i, the following operations are performed:

1. build perturbations
   - substitution: replace the ith symbol of $\bar{p}$ by each symbol of Σ in turn and choose the resulting string x with the smallest SOD relative to S.
   - insertion: insert each symbol of Σ in turn at the ith position of $\bar{p}$ and choose the resulting string y with the smallest SOD relative to S.
   - deletion: delete the ith symbol of $\bar{p}$ to generate z.
2. Replace $\bar{p}$ by the one from $\{\bar{p}, x, y, z\}$ with the smallest SOD relative to S.

There is a particularly favorite situation for the perturbation algorithm from Martinez-Hinarejos et al. (2000). If there is one input string in S with one edit error from the ground truth only, then this string tends to be the set median, which can be fully corrected by the perturbation approach. For our tests we have excluded such input strings by requiring at least two edit operations in the data generation step.

We report one test series related to varying string length in Fig. 10, where the measures are averaged values over 25 times (in contrast to 100 times in all other test series)[1]. The performance of the greedy algorithm is far lower than the other ones. This is not surprising since the local termination criteria tend to stop early and thus fail to obtain the correct length. On the other hand, the perturbation method is only slightly worse than our recursive method, but needs much more computation time (factor of 40–50).

5. Conclusions

In the present paper we have proposed a novel technique to obtain approximate solutions for the median string. This new approach is based on embedding of strings into vector spaces. First, the strings are turned into points of vector spaces using the string edit distance paradigm. Then, the crucial point of obtaining the median of the set is carried out in the vector space, not in the string domain, which substantially simplifies this operation. Finally, using the string edit distance again we can transform the obtained median vector to a string by means of the weighted mean of a pair of strings. This embedding approach allows us to get the main advantages of both the vector and string representations. That is, we compute the more complex parts in real vector spaces but keeping the representational power of strings. The experimental evaluation has clearly demonstrated its superiority to the set median. More importantly, the computed median is shown to closely approach the lower bound (thus the unknown ground truth).

From a practical point of view, our approach is certainly a more suitable choice than the exact methods (Kruskal, 1983; Lopresti and Zhou, 1997). In contrast to greedy methods (Casacuberta and de Antoni, 1997; Kruzslicz, 1999) our global approach avoids the risk of making wrong local decision at each symbol. In addition, the computational time is independent of the alphabet size.

The most critical part of the algorithm is the inverse transformation back to the string space. We have tested three different variants in this work and the recursive approach turns out to be a reasonable choice for most situations. But the inverse transformation is certainly worth of further consideration, not only from the median string but also from a more general point of view in the context of embedding based median computation in structural pattern recognition. Also interesting is to apply the proposed embedding method to distance functions different from the edit distance used in this work and to deal with weighted generalized median problems.

[1] This comparison test series was run on a slower PC than for all other test series, which leads to longer computation time compared to Fig. 9 (bottom). This is also the reason why only 25 times were run instead of 100.

References

Bajaj, C., 1988. The algebraic degree of geometric optimization problems. Discrete Comput. Geom. 3 (2), 177–191.

Bunke, H., Sanfeliu, A., 1990. Syntactic and Structural Pattern Recognition Theory and Applications. World Scientific.

Bunke, H., Jiang, X., Abegglen, K., Kandel, A., 2002. On the weighted mean of a pair of strings. Pattern Anal. Appl. 5 (1), 23–30.

Casacuberta, F., de Antoni, M., 1997. A greedy algorithm for computing approximate median strings. In: Proceedings of National Symposium on Pattern Recognition and Image Analysis, pp. 193–198.

Chandrasekaran, R., Tamir, A., 1989. Open questions concerning weiszfelds algorithm for the Fermat–Weber location problem. Math. Program. 44, 293–295.

de la Higuera, C., Casacuberta, F., 2000. Topology of strings: Median string is NP-complete. Theor. Comput. Sci. 230 (1/2), 739–748.

Ferrer, M., Valveny, E., Serratosa, F., Bardaj, I., Bunke, H., 2009. Graph-based k-means clustering: A comparison of the set median versus the generalized median graph. In: Jiang, X., Petkov, N. (Eds.), Proceedings of CAIP, pp. 342–350.

Ferrer, M., Valveny, E., Serratosa, F., Riesen, K., Bunke, H., 2010. Generalized median graph computation by means of graph embedding in vector spaces. Pattern Recognit. 43, 1642–1655.

Ferrer, M., Karatzas, D., Valveny, E., Bardaj, I., Bunke, H., 2011. A generic framework for median graph computation based on a recursive embedding approach. Computer Vision and Image Understanding 115 (7), 919–928.

Fred, A., Leitao, J., 1998. A comparative study of string dissimilarity measures in structural clustering. In: Proceedings of International Conference on Document Analysis and Recognition, pp. 385–394.

Gusfield, D., 1997. Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology. Cambridge University Press.

Hakimi, S., 2000. Location Theory. CRC Press.

Jiang, X., Bunke, H., 2002. Optimal lower bound for generalized median problems in metric space. In: Caelli, T., Amin, A., Duin, R., Kamel, M., de Ridder, D. (Eds.), Structural, Syntactic, and Statistical Pattern Recognition. pp. 143–151.

Jiang, X., Bunke, H., 2010. Learning by generalized median concept. In: Wang, P. (Ed.), Pattern Recognition and Machine Vision. River Publishers, pp. 1–16.

Jiang, X., Münger, A., Bunke, H., 2001. On median graphs: Properties, algorithms, and applications. IEEE Trans. Pattern Anal. Machine Intell. 23 (10), 1144–1151.

Jiang, X., Abegglen, K., Bunke, H., Csirik, J., 2003. Dynamic computation of generalized median strings. Pattern Anal. Appl. 6 (3), 185–193.

Jiang, X., Bunke, H., Csirik, J., 2004. Median strings: A review. In: Last, M., Kandel, A., Bunke, H. (Eds.), Data Mining in Time Series Databases. World Scientific, pp. 173–192.

Juan, A., Vidal, E., 1998. Fast median search in metric spaces. In: Amin, A., Dori, D. (Eds.), Advances in Pattern Recognition. Springer, pp. 905–912.

Kohonen, T., 1985. Median strings. Pattern Recognition Lett. 3, 309–313.

Kruskal, J., 1983. An overview of sequence comparison: Time warps, string edits, and macromolecules. SIAM Reviews 25 (2), 201–237.

Kruzslicz, F., 1999. Improved greedy algorithm for computing approximate median strings. Acta Cybernetica 14, 331–339.

Lopresti, D., Zhou, J., 1997. Using consensus sequence voting to correct ocr errors. Computer Vision and Image Understanding 67 (1), 39–47.

Marti, V., Bunke, H., 2001. Use of positional information in sequence alignment for multiple classifier combination. In: Kittler, J., Roli, F. (Eds.), Proceedings of Multiple Classifier Combination. Springer, pp. 388–398.

Martinez-Hinarejos, C., Juan, A., Casacuberta, F., 2000. Use of median string for classification. In: Proceedings of International Conference on Pattern Recognition, pp. 907–910.

Marzal, A., Vidal, E., 1993. Computation of normalized edit distance and applications. IEEE Trans. Pattern Anal. Mach. Intell. 15 (9), 926–932.

Mico, L., Oncina, J., 2001. An approximate median search algorithm in nonmetric spaces. Pattern Recognition Lett. 22 (10), 1145–1151.

Nicolas, F., Rivals, E., 2005. Hardness results for the center and median string problems under the weighted and unweighted edit distances. J. Discrete Algor. 3, 390–415.

Olivares-Rodriguez, C., Oncina, J., 2008. A stochastic approach to median string computation. In: Proceedings of SSPR/SPR, pp. 431–440.

Sankoff, D., Kruskal, J. (Eds.), 1983. Time Warps, String Edits, and Macromolecules: The Theory and Practice of String Comparison. Addison-Wesley.

Sim, J., Park, K., 2003. The consensus string problem for a metric is NP-complete. J. Discrete Algor. 1 (1), 111–117.

Spillmann, B., Neuhaus, M., Bunke, H., Pekalska, E., Duin, R., 2006. Transforming strings to vector spaces using prototype selection. In: Proceedings of SSPR/SPR, pp. 287–296.
Stephen, G., 1994. String Searching Algorithms. World Scientific.
Vidal, E., Marzal, A., Aibar, P., 1995. Fast computation of normalized edit distances. IEEE Trans. Pattern Anal. Mach. Intell. 17 (9), 899–902.
Wagner, R., Fischer, M., 1974. The string-to-string correction problem. J. ACM 21 (1), 168–173.
Wang, L., Jiang, T., 1994. On the complexity of multiple sequence alignment. J. Comput. Biol. 1 (4), 337–348.
Weiszfeld, E., 1937. Sur le point pour lequel la somme des distances de n points donnés est minimum. Tohoku Math. J. 43, 355–386.
Wesolowsky, G., 1993. The Weber problem: History and perspective. Location Sci. 1, 5–23.