Integer Linear Programming Approach to Median and Center Strings for a Probability Distribution on a Set of Strings

Morihiro Hayashida[1] and Hitoshi Koyano[2]
[1]*Bioinformatics Center, Institute for Chemical Research, Kyoto University, Gokasho, Uji, Kyoto 611-0011, Japan*
[2]*Graduate School of Medicine, Kyoto University, 54 Kawahara-cho, Shogoin, Sakyo-ku, Kyoto 606-8397, Japan*

Keywords: Median String, Center String, Integer Linear Programming.

Abstract: We address problems of finding median and center strings for a probability distribution on a set of strings under Levenshtein distance, which are known to be NP-hard in a special case. There are many applications in various research fields, for instance, to find functional motifs in protein amino acid sequences, and to recognize shapes and characters in image processing. In this paper, we propose novel integer linear programming-based methods for finding median and center strings for a probability distribution on a set of strings under Levenshtein distance. Furthermore, we restrict several variables to a region near the diagonal in the formulation, and propose novel integer linear programming-based methods also for finding approximate median and center strings for a probability distribution on a set of strings. For evaluation of our proposed methods, we perform several computational experiments, and show that the restricted formulation reduced the execution time.

1 INTRODUCTION

It is a fundamental statistical method for understanding a data set to take an average. In this paper, we focus on a set of strings. For instance, nucleotide sequences of DNAs and RNAs are represented by strings as well as protein amino acid sequences. The number of such sequences has rapidly increased, and analytical methods are required. In the field of evolutionary studies of organisms, it would be an aim to find genetic information, nucleotide sequences of common ancestors. In the field of protein science, it is essential to find functional motifs in protein amino acid sequences. Also in the field of image recognition, there are several applications such as post-processing of optical character recognition (OCR) results (Bunke et al., 2002) and shape recognition (Chen et al., 1998). Furthermore, it can be applied to classification and clustering of strings and biological sequences (Martínez-Hinarejos et al., 2003).

Several definitions have been proposed for representing an average of strings because the average is not uniquely determined. One is a *median string*, which is defined as a string that minimizes the sum of distances with strings included in a set (Kohonen, 1985). One is a *center string*, which is defined as a string that minimizes the maximum of distances with strings (Gusfield, 1997). As distances between two strings, several distances such as Levenshtein distance (Levenshtein, 1965), Hamming distance (Hamming, 1950), and Jaro-Winkler distance (Winkler, 1990) have been proposed, where the Jaro-Winkler distance is known not to obey the triangle inequality. The Levenshtein distance between two given strings s and t allows three types of edit operations, insertion, deletion, substitution, and can be calculated in polynomial time $O(|s||t|)$ using dynamic programming, where $|s|$ denotes the length of s. The Hamming distance has been also used for closest strings and related problems (Gramm, 2003; Gramm et al., 2003). Data reduction techniques that reduce candidates of a center string under the Hamming distance were developed (Hufsky et al., 2011). However, they mentioned that their parameterized methods would be not applicable for finding center strings under the Levenshtein distance. A genetic algorithm for finding closest strings under rank distance was developed (Dinu and Ionescu, 2012), where the rank distance has been applied in biology, natural language processing, and authorship attribution.

The problems of finding the median and center strings for a finite set of strings under the Levenshtein distance have been proved to be NP-complete for an unbounded alphabet (de la Higuera and Casacuberta, 2000), and even for a binary alphabet (Nicolas and Rivals, 2003; Nicolas and Rivals, 2005). It has

Hayashida, M. and Koyano, H.
Integer Linear Programming Approach to Median and Center Strings for a Probability Distribution on a Set of Strings.
DOI: 10.5220/0005666400350041
In *Proceedings of the 9th International Joint Conference on Biomedical Engineering Systems and Technologies (BIOSTEC 2016) - Volume 3: BIOINFORMATICS*, pages 35-41
ISBN: 978-989-758-170-0

been proved that a related problem CSCE is also NP-complete when a penalty matrix is a metric (Sim and Park, 2003). An exact algorithm for finding the median string under the Levenshtein distance using dynamic programming was proposed (Kruskal, 1983), which requires N-dimensional array and $O(n^N)$ time and space for a set of N strings with length n, for example, it requires $10^{10} \cdot 4$ bytes $=$ 40GB memory for $n = N = 10$. For computing approximate median strings in practical time, several methods have been proposed. If given strings are all quite similar, the path by the optimal dynamic programming should be close to the main diagonal. Hence, the method to restrict candidate paths to a region near the diagonal was proposed (Lopresti and Zhou, 1997). A greedy algorithm starts from an empty string, and selects a letter that minimizes the exact consensus error (Casacuberta and de Antoni, 1997). An online algorithm takes the current approximate median string and a new string, and calculates a weighted mean of these strings (Jiang et al., 2003). In a stochastic approach, some conditional probability from a string to another was defined, and an approximate median string was obtained by expectation maximization technique (Olivares-Rodríguez and Oncina, 2008). An iterative algorithm applies the edit operation with some highest score to the current string until a better solution is not found (Abreu and Rico-Juan, 2014). These methods output approximate median strings, and there are a few methods to output optimal median strings. As far as we know, methods for finding optimal center strings have not been developed. In this paper, hence, we propose an approach using integer linear programming for finding optimal median and center strings because efficient solvers for integer linear programming problems have been developed. In addition, we introduce a probability distribution on a set of strings (Koyano and Kishino, 2010), and propose methods for finding median and center strings of such a probability distribution under the Levenshtein distance. Furthermore, we propose integer linear programming formulations restricted to a region near the diagonal for finding approximate median and center strings. We perform several computational experiments and verify the efficiency of our methods.

2 METHODS

We use the Levenshtein distance because it is often used and a fundamental edit distance. In this section, we briefly review the computation of the Levenshtein distance, median, center strings, and propose integer linear programming formulations for exact and approximate median and center strings. Let $\mathcal{A} = \{a_1, \ldots, a_z\}$ be an alphabet composed of z letters, for instance, $\mathcal{A} = \{A, T, G, C\}$ for DNA nucleotide sequences. We define $\mathcal{A}^*$ to be the set of all strings on $\mathcal{A}$ with varying lengths, and for a string $s \in \mathcal{A}^*$, $|s|$ denotes the length of s.

2.1 Levenshtein Distance

The Levenshtein distance $d(s,t)$ between two strings s and t is defined as the minimum cost of sequences of edit operations transforming $s = s_1 \cdots s_n$ into $t = t_1 \cdots t_m$, and can be calculated by the following dynamic programming (Wagner and Fischer, 1974).

$$D[0,0] = 0, \tag{1}$$

$$D[i,j] = \min \begin{cases} D[i-1,j-1] + \gamma(s_i \to t_j) \\ D[i-1,j] + \gamma(s_i \to \varepsilon) \\ D[i,j-1] + \gamma(\varepsilon \to t_j) \end{cases} \tag{2}$$

where ε denotes an empty letter, $\gamma(s_i \to t_j)$, $\gamma(s_i \to \varepsilon)$, and $\gamma(\varepsilon \to t_j)$ denote the costs of substitution, deletion, and insertion, respectively. Then, $D[n,m]$ is the Levenshtein distance $d(s,t)$.

2.2 Median and Center Strings

Given N strings $s^{(k)}$ with length n_k $(k = 1, \ldots, N)$ on $\mathcal{A}^*$, the median string is defined by

$$\mathrm{argmin}_{t \in \mathcal{A}^*} \sum_{k=1}^{N} d(t, s^{(k)}). \tag{3}$$

Similarly, the center string is defined by

$$\mathrm{argmin}_{t \in \mathcal{A}^*} \max_{k \in \{1,\ldots,N\}} d(t, s^{(k)}). \tag{4}$$

For a given probability distribution $p(s)$ on $\mathcal{A}^*$, we define median and center strings by

$$\mathrm{argmin}_{t \in \mathcal{A}^*} \sum_{s \in \mathcal{A}^*} p(s) d(t,s), \tag{5}$$

$$\mathrm{argmin}_{t \in \mathcal{A}^*} \max_{s \in \mathcal{A}^*} p(s) d(t,s), \tag{6}$$

respectively. If $p(s^{(k)}) = \frac{1}{N}$ for $k = 1, \ldots, N$ and $p(s) = 0$ for all $s \notin \{s^{(k)}\}$, Eqs (3) and (4) are equivalent to Eqs (5) and (6), respectively.

2.3 Integer Linear Programming Formulation

Since it is known that problems of finding median and center strings under the Levenshtein distance are NP-hard (Nicolas and Rivals, 2003; Nicolas and Rivals, 2005), we make use of integer linear programming

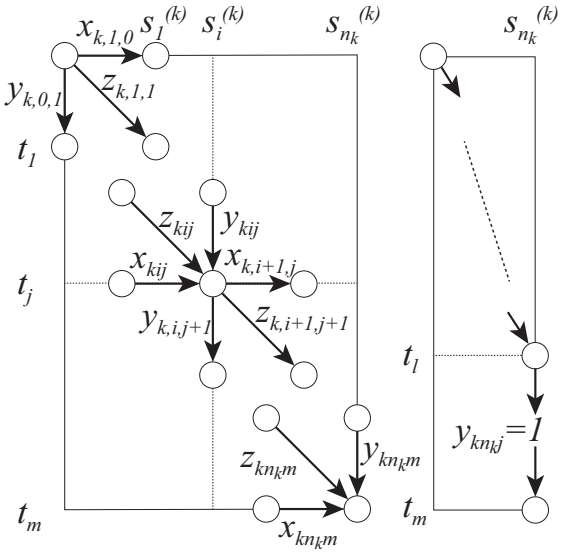


Figure 1: Illustration on variables appeared in our integer linear programming formulation. (Left) Variables x_{kij}, y_{kij}, and z_{kij} represent a path in dynamic programming for calculating the Levenshtein distance if the value of its variable is equal to 1. (Right) Variable l represents the length of string t. For all $j > l$, y_{kn_kj} is forced to be 1.

which efficient solvers have been developed. We can find a median string t by integer linear programming if the Levenshtein distance $d(t,s^{(k)})$ between t and $s^{(k)}$ can be calculated in linear formulas. It, however, is difficult to directly represent the array $D[i,j]$ in the dynamic programming by integer linear programming because it includes the selection of the minimum value in Eq. (2).

Suppose that a probability distribution $p(s)$ on $\mathcal{A}^*$ is given, where the number of strings s satisfying $p(s) > 0$ is finite, N, that is, $p(s^{(k)}) > 0$ $(k = 1,\ldots,N)$. We use integer numbers $1,\ldots,|\mathcal{A}|$ instead of letters in $\mathcal{A}$ because a variable takes a value in linear programming. $s_i^{(k)}$ $(i = 1,\ldots,n_k)$ is given as a constant value of $1,\ldots,|\mathcal{A}|$, and represents the i-th letter in $s^{(k)}$. t_j $(j = 1,\ldots,m)$ is a variable taking a value of $1,\ldots,|\mathcal{A}|$. Then, we propose the following integer linear programming formulation, called ILPMed, for finding the median string for $p(s)$ under the Levenshtein distance with costs C_{sub}, C_{del}, C_{ins} of substitution, deletion, insertion (see Fig. 1).

$$\min \sum_{k=1}^{N} p(s^{(k)}) \Big\{ \sum_{i=1}^{n_k} C_{del} x_{k,i,0} + \sum_{j=1}^{m} C_{ins} y_{k,0,j} + \sum_{i=1}^{n_k} \sum_{j=1}^{m} (C_{del} x_{kij} + C_{ins} y_{kij} + C_{sub} h_{kij}) \Big\} - C_{ins}(m-l)$$

subject to

for all $k = 1,\ldots,N$,

$$1 = x_{k,1,0} + y_{k,0,1} + z_{k,1,1}, \quad \text{(a1)}$$

$$x_{k,i,0} = x_{k,i+1,0} + y_{k,i,1} + z_{k,i+1,1} \quad \text{for all } i < n_k, \text{(a2)}$$

$$x_{k,n_k,0} = y_{k,n_k,1}, \quad \text{(a3)}$$

$$y_{k,0,j} = x_{k,1,j} + y_{k,0,j+1} + z_{k,1,j+1} \quad \text{for all } j < m, \text{(a4)}$$

$$y_{k,0,m} = x_{k,1,m}, \quad \text{(a5)}$$

$$x_{kij} + y_{kij} + z_{kij} = x_{k,i+1,j} + y_{k,i,j+1} + z_{k,i+1,j+1} \quad \text{for all } i < n_k,\ j < m, \text{(a6)}$$

$$x_{kn_kj} + y_{kn_kj} + z_{kn_kj} = y_{k,n_k,j+1} \quad \text{for all } j < m, \text{(a7)}$$

$$x_{kim} + y_{kim} + z_{kim} = x_{k,i+1,m} \quad \text{for all } i < n_k, \text{(a8)}$$

$$x_{kn_km} + y_{kn_km} + z_{kn_km} = 1, \quad \text{(a9)}$$

$$y_{kn_kj} \geq \tfrac{1}{m}(j-l) \quad \text{for all } j, \quad \text{(b)}$$

for all k, i, j,

$$s_i^{(k)} - t_j \leq |\mathcal{A}| g_{kij}, \quad \text{(c1)}$$

$$t_j - s_i^{(k)} \leq |\mathcal{A}| g_{kij}, \quad \text{(c2)}$$

$$h_{kij} \geq z_{kij} + g_{kij} - 1, \quad \text{(d1)}$$

$$h_{kij} \leq \tfrac{1}{2}(z_{kij} + g_{kij}), \quad \text{(d2)}$$

$$x_{kij}, y_{kij}, z_{kij}, g_{kij}, h_{kij} \in \{0,1\},$$

$$t_j \in \{1,\ldots,|\mathcal{A}|\},\ 0 \leq l \leq m,$$

where m is a sufficient large constant integer, that is, the sum of n_k, and l is the variable representing the length of median string.

In the formulation, variable x_{kij} takes 1 if $s_i^{(k)}$ is deleted, otherwise 0. y_{kij} takes 1 if t_j is inserted, otherwise 0. z_{kij} takes 1 if $s_i^{(k)}$ is substituted with t_j, otherwise 0. There must be exactly one path from the upper left to the lower right for each string $s^{(k)}$. If either of x_{kij}, y_{kij}, and z_{kij} is 1, either of $x_{k,i+1,j}$, $y_{k,i,j+1}$, and $z_{k,i+1,j+1}$ must be 1, which is represented by Eq. (a6). According to the position (i,j), Eqs (a1-9) are constructed. Eq. (b) represents the constraint that the length of median string t is l, and y_{kn_kj} is forced to be 1 if $j > l$. It is difficult to represent the Levenshtein distance $d(t,s^{(k)}) = \sum_{i=1}^{n_k} C_{del} x_{k,i,0} + \sum_{j=1}^{l} C_{ins} y_{k,0,j} + \sum_{i=1}^{n_k} \sum_{j=1}^{l} (C_{del} x_{kij} + C_{ins} y_{kij} + C_{sub} h_{kij})$ in the formulation because l is also a variable to be found. Hence, we use a constant integer m instead of l. Then, the sum includes the extra cost of $C_{ins}(m-l)$. We reduce the cost such that the objective function represents the sum in Eq. (5). It should be noted that for all $j > l$, y_{kn_kj} is forced to be 1. Eqs (c1-2) represent that g_{kij} becomes 1 if $s_i^{(k)}$ is the same as t_j. Eqs (d1-2) represent that h_{kij} becomes 1 if and only if both of z_{kij} and g_{kij} are 1. It means that the substitution cost from $s_i^{(k)}$ to t_j is C_{sub} if $s_i^{(k)} \neq t_j$, otherwise 0.

It is guaranteed that we can find the median string for $p(s)$ under the Levenshtein distance by solving this integer linear programming formulation because the objective function is equivalent to the sum in Eq. (5), t can be any string with length up to $m = \sum_{k=1}^{N} n_k$, and the sum for a string with length more than m is larger than that for the concatenated string of all $s^{(k)}$.

In a similar way to median strings, we propose the following integer linear programming formula-

tion, called ILPCen, for finding the center string for a probability distribution $p(s)$.

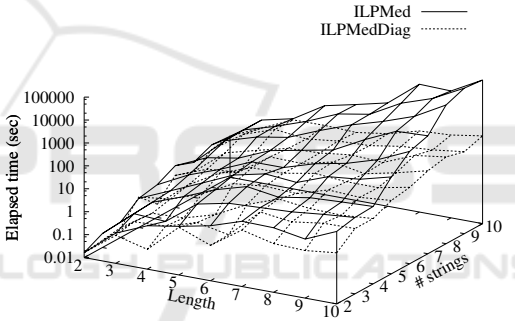
$$
\begin{aligned}
&\min \quad d \\
&\text{subject to} \\
&\quad \text{for all } k = 1,\dots,N, \\
&\quad p(s^{(k)})\Big\{\sum_{i=1}^{n_k} C_{del}x_{k,i,0} + \sum_{j=1}^{m} C_{ins}y_{k,0,j} + \sum_{i=1}^{n_k}\sum_{j=1}^{m} \\
&\quad (C_{del}x_{kij} + C_{ins}y_{kij} + C_{sub}h_{kij}) - C_{ins}(m-l)\Big\} \le d, \\
&\quad 1 = x_{k,1,0} + y_{k,0,1} + z_{k,1,1}, \\
&\quad x_{k,i,0} = x_{k,i+1,0} + y_{k,i,1} + z_{k,i+1,1} \quad \text{for all } i < n_k, \\
&\quad x_{k,n_k,0} = y_{k,n_k,1}, \\
&\quad y_{k,0,j} = x_{k,1,j} + y_{k,0,j+1} + z_{k,1,j+1} \quad \text{for all } j < m, \\
&\quad y_{k,0,m} = x_{k,1,m}, \\
&\quad x_{kij} + y_{kij} + z_{kij} = x_{k,i+1,j} + y_{k,i,j+1} + z_{k,i+1,j+1} \\
&\qquad\qquad \text{for all } i < n_k,\ j < m, \\
&\quad x_{kn_kj} + y_{kn_kj} + z_{kn_kj} = y_{k,n_k,j+1} \quad \text{for all } j < m, \\
&\quad x_{kim} + y_{kim} + z_{kim} = x_{k,i+1,m} \quad \text{for all } i < n_k, \\
&\quad x_{kn_km} + y_{kn_km} + z_{kn_km} = 1, \\
&\quad y_{kn_kj} \ge \tfrac{1}{m}(j-l) \quad \text{for all } j, \\
&\quad \text{for all } k,\ i,\ j, \\
&\quad s_i^{(k)} - t_j \le |\mathcal{A}| g_{kij}, \\
&\quad t_j - s_i^{(k)} \le |\mathcal{A}| g_{kij}, \\
&\quad h_{kij} \ge z_{kij} + g_{kij} - 1, \\
&\quad h_{kij} \le \tfrac{1}{2}(z_{kij} + g_{kij}), \\
&\quad x_{kij}, y_{kij}, z_{kij}, g_{kij}, h_{kij} \in \{0,1\}, \\
&\quad t_j \in \{1,\dots,|\mathcal{A}|\}, \\
&\quad 0 \le l \le m,\ d \ge 0.
\end{aligned}
$$

Here, d is a variable that represents the minimum in Eq.(6).

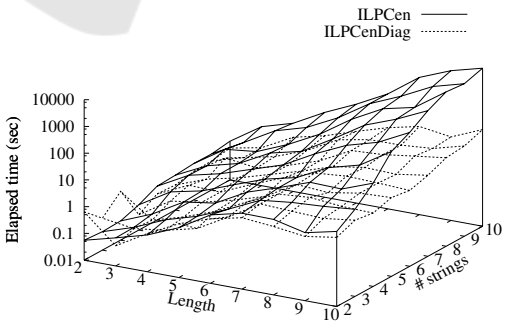
If strings $s^{(k)}$ are similar to each other, we can restrict candidate paths to a region near the diagonal without loss of optimality. We introduce an constant positive integer w, and propose integer linear programming formulations, called ILPMedDiag and ILPCenDiag, by reducing variables, x_{kij}, y_{kij}, z_{kij}, g_{kij}, and h_{kij} with $|i-j| > w$ from ILPMed and ILPCen, respectively.

3 COMPUTATIONAL EXPERIMENTS

For the evaluation of our methods, we performed several computational experiments. We used $C_{del} = C_{ins} = C_{sub} = 1$ to calculate the Levenshtein distance, and used an alphabet $\mathcal{A}$ with 4 letters as DNA and RNA nucleotide sequences. We randomly generated two types of probability distributions, $p_1(s)$ and $p_2(s)$, on $\mathcal{A}^*$. In $p_1(s)$, N strings $s^{(k)}$ with length n_k were generated as strings satisfying $p_1(s) > 0$ while varying $N = 2,\dots,10$ and $n_k = 2,\dots,10$, where n_k was the same for all $k = 1,\dots,N$. Each $s_i^{(k)}$ was generated as $\min(1 + \lfloor|\alpha|\rfloor, |\mathcal{A}|)$, where α followed the normal distribution with mean 0 and variance 1, and $\lfloor\alpha\rfloor$ is the largest integer not greater than α. The probability of $p_1(s^{(k)})$ was generated uniquely at random such that $\sum_{k=1}^{N} p_1(s^{(k)}) = 1$ holds. In $p_2(s)$, N strings $s^{(k)}$ were generated from a string of $a_1 \cdots a_1$ ($a_1 \in \mathcal{A}$) with length n by applying randomly selected edit operations of substitution, insertion, and deletion, three times, where we examined $N = 2,\dots,10$ and $n = 5,\dots,10$, and the length n_k of $s^{(k)}$ could be different according to k. The probability of $p_2(s^{(k)})$ was generated uniquely at random such that $\sum_{k=1}^{N} p_2(s^{(k)}) = 1$ holds. For each case of $p_1(s)$, $p_2(s)$, n_k, and N, we generated a set of N strings $s^{(k)}$ with $p_1(s^{(k)})$ or $p_2(s^{(k)})$ ten times, and took the average of execution times. We used CPLEX (version 12.5) as the integer linear programming solver under a linux operating system with Xeon 2.9GHz processor and 35GB memory.



(a) median string



(b) center string

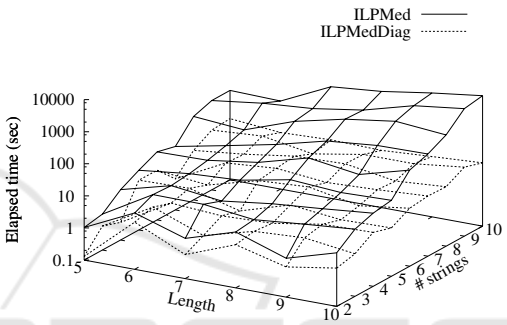
Figure 2: Results on the average execution time in seconds on a log scale by ILPMed, ILPMedDiag with $w = 2$, ILPCen, and ILPCenDiag with $w = 2$ for probability distributions $p_1(s)$ for $N = 2,\dots,10$ and $n_k = 2,\dots,10$. (a) For finding median strings. (b) For finding center strings.

Table 1: Results on the average and standard deviation of execution time in seconds by ILPMed, ILPMedDiag with $w = 2$, ILPCen, and ILPCenDiag with $w = 2$ for probability distributions $p_1(s)$ for $n = 10$ and $N = 2,\ldots,10$.

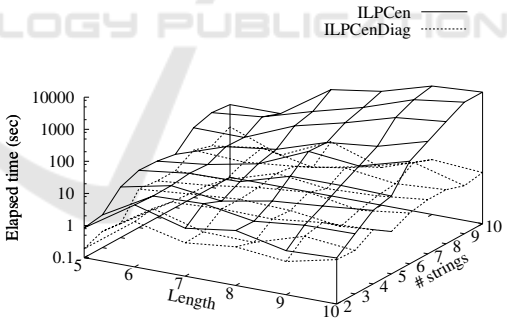
N	ILPMed		ILPMedDiag		ILPCen		ILPCenDiag	
	average	s.d.	average	s.d.	average	s.d.	average	s.d.
2	14.4	6.8	1.7	1.8	6.4	1.9	3.8	2.5
3	21.5	10.5	7.0	4.4	18.1	7.4	7.0	5.1
4	23.7	15.3	7.0	4.5	72.1	27.2	5.5	4.0
5	168.0	69.0	15.7	3.6	464.8	312.8	8.8	4.8
6	798.6	280.4	21.7	8.3	1015.6	580.6	6.9	3.0
7	1400.6	773.3	41.6	20.5	2810.3	1218.8	10.7	4.4
8	11269.8	9074.5	61.0	39.2	5936.2	3522.8	20.4	9.5
9	22438.1	16417.6	37.7	17.9	5086.0	1358.6	23.9	10.5
10	18467.8	16625.5	67.4	22.8	8120.3	4336.0	41.1	29.5

Fig. 2 shows results on the average execution time in seconds on a log scale by ILPMed, ILPMedDiag with $w = 2$, ILPCen, and ILPCenDiag with $w = 2$ for probability distributions $p_1(s)$ for $N = 2,\ldots,10$ and $n_k = 2,\ldots,10$. Table 1 shows the detailed average and standard deviation of execution time by ILPMed, ILPMedDiag, ILPCen, and ILPCenDiag for $n = 10$. We can see from these that the average execution times by ILPMed and ILPCen rapidly, almost exponentially, increased with both of the number N of strings and the length n_k because the problems are NP-hard. On the other hand, the average execution times by ILPMedDiag and ILPCenDiag were smaller than those by ILPMed and ILPCen, respectively, because candidate solutions for the problems were restricted to the region near the diagonal.

Fig. 3 shows results on the average execution time in seconds on a log scale by ILPMed, ILPMedDiag with $w = 2$, ILPCen, and ILPCenDiag with $w = 2$ for probability distributions $p_2(s)$ for $N = 2,\ldots,10$ and $n = 5,\ldots,10$. Table 2 shows the detailed average and standard deviation of execution time by ILPMed, ILPMedDiag, ILPCen, and ILPCenDiag for $n = 10$. Also for $p_2(s)$, the average execution times by ILPMedDiag and ILPCenDiag were smaller than those by ILPMed and ILPCen, respectively. The slopes of ILPMed and ILPCen along the length for $p_2(s)$ were smaller than those for $p_1(s)$, respectively. In addition, the average execution times for $p_2(s)$ were smaller than those for $p_1(s)$. It, however, is considered that ILPMed and ILPCen might be not sufficient to be applied to actual data for finding their optimal median and center strings. Fig. 4 shows results on the average objective value by ILPMed, ILPMedDiag with $w = 2$, ILPCen, and ILPCen with $w = 2$ for probability distributions $p_2(s)$ for $N = 2,\ldots,10$ and $n = 5,\ldots,10$. It is noted that the objective values by ILPMed and ILPCen for $p_1(s)$ were almost the same as those by ILPMedDiag and ILPCenDiag, respectively. In $p_2(s)$, three edit operations were applied to strings, and differences of objective values between ILPMed and ILPMedDiag with $w = 2$, and between ILPCen and ILPCenDiag with $w = 2$, occurred. We can obtain optimal strings using ILPMedDiag and ILPCenDiag by increasing the width w of diagonal.



(a) median string

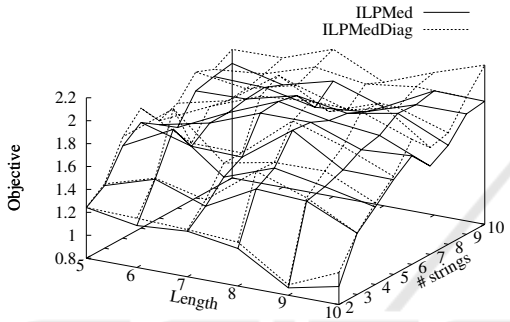


(b) center string

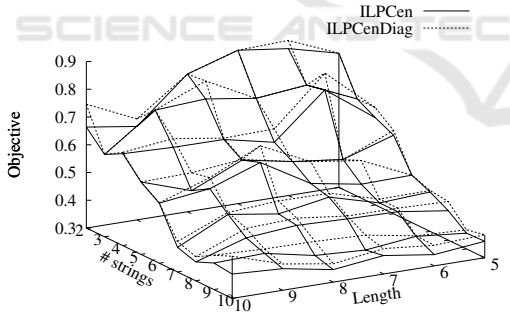
Figure 3: Results on the average execution time in seconds on a log scale by ILPMed, ILPMedDiag with $w = 2$, ILPCen, and ILPCenDiag with $w = 2$ for probability distributions $p_2(s)$ for $N = 2,\ldots,10$ and $n = 5,\ldots,10$.

Table 2: Results on the average and standard deviation of execution time in seconds by ILPMed, ILPMedDiag with $w = 2$, ILPCen, and ILPCenDiag with $w = 2$ for probability distributions $p_2(s)$ for $n = 10$ and $N = 2, \ldots, 10$.

N	ILPMed		ILPMedDiag		ILPCen		ILPCenDiag	
	average	s.d.	average	s.d.	average	s.d.	average	s.d.
2	4.7	3.2	1.6	2.4	2.7	2.8	2.4	2.8
3	12.4	7.8	2.1	2.5	7.5	4.5	2.3	2.6
4	18.9	11.5	4.8	5.3	19.5	13.9	4.9	3.1
5	23.3	23.2	3.2	2.5	26.6	20.0	2.1	2.4
6	98.6	76.1	6.4	3.9	157.0	110.7	3.8	2.6
7	154.3	120.0	7.5	4.0	320.4	257.5	4.9	4.1
8	293.7	170.0	7.1	6.5	641.3	425.0	4.8	4.4
9	943.4	747.9	14.1	8.4	1259.2	676.6	3.1	3.3
10	1160.4	276.1	9.4	5.5	1266.7	488.1	3.9	3.6



(a) median string



(b) center string

Figure 4: Results on the average objective value by ILPMed, ILPMedDiag with $w = 2$, ILPCen, and ILPCen with $w = 2$ for probability distributions $p_2(s)$ for $N = 2, \ldots, 10$ and $n = 5, \ldots, 10$.

4 CONCLUSION

We extended the definitions of median and center strings, which problems are known to be NP-hard, to those over a probability distribution $p(s)$ on a set of strings $\mathcal{A}^*$, and proposed novel integer linear programming-based methods, ILPMed, and ILPCen, for finding median and center strings for $p(s)$ on $\mathcal{A}^*$, and ILPMedDiag, ILPCenDiag for finding approximate median and center strings for $p(s)$ on $\mathcal{A}^*$ by restricting several variables of ILPMed and ILPCen to a region near the diagonal. We performed computational experiments, and confirmed that the execution times by ILPMedDiag and ILPCenDiag were smaller than those by ILPMed and ILPCen, respectively, and ILPMedDiag and ILPCenDiag reduced the execution times. ILPMed and ILPCen, however, might be not sufficient to be applied to actual data for finding their optimal median and center strings. It is considered because the number of candidate paths from the upper left to the lower right in ILPMed and ILPCen is enormous and should be selected by solvers although the Levenshtein distance between two strings can be calculated in polynomial time. On the other hand, ILPMedDiag and ILPCenDiag are considered to be useful if given strings are similar to each other because the number of such candidate paths in ILPMedDiag and ILPCenDiag is small. As future work, we need to analyze computational time and space complexities for our proposed methods. Furthermore, we would like to improve our methods by introducing other types of restriction to the variables than those in ILPMedDiag and ILPCenDiag. In addition, we will consider decomposition of strings, linear programming relaxation, and utilize approximate solutions obtained by ILPMedDiag and ILPCenDiag in order to find optimal solutions by ILPMed and ILPCen.

ACKNOWLEDGEMENTS

This work was partially supported by Grants-in-Aid #24500361, and #26610037 from MEXT, Japan.

REFERENCES

Abreu, J. and Rico-Juan, J. (2014). A new iterative algorithm for computing a quality approximate median of strings based on edit operations. *Pattern Recognition Letters*, 36:74–80.

Bunke, H., Jiang, X., Abegglen, K., and Kandel, A. (2002). On the weighted mean of a pair of strings. *Pattern Analysis and Applications*, 5:23–30.

Casacuberta, F. and de Antoni, M. (1997). A greedy algorithm for computing approximate median strings. pages 193–198.

Chen, S., Tung, S., Fang, C., Cherng, S., and Jain, A. (1998). Extended attributed string matching for shape recognition. *Computer Vision and Image Understanding*, 70:36–50.

de la Higuera, C. and Casacuberta, F. (2000). Topology of strings: Median string is NP-complete. *Theoretical Computer Science*, 230:39–48.

Dinu, L. and Ionescu, R. (2012). An efficient rank based based approach for closest string and closest substring. *PLoS ONE*, 7(6):e37576.

Gramm, J. (2003). *Fixed-parameter algorithms for the consensus analysis of genomic data*. PhD thesis, Universität Tübingen.

Gramm, J., Niedermeier, R., and Rossmanith, P. (2003). Fixed-parameter algorithms for closest string and related problems. *Algorithmica*, 37:25–42.

Gusfield, D. (1997). *Algorithms on strings, trees and sequences*. Cambridge University Press.

Hamming, R. (1950). Error detecting and error correcting codes. *The Bell System Technical Journal*, 29(2):147–160.

Hufsky, F., Kuchenbecker, L., Jahn, K., Stoye, J., and Böcker, S. (2011). Swiftly computing center strings. *BMC Bioinformatics*, 12:106.

Jiang, X., Abegglen, K., Bunke, H., and Csirik, J. (2003). Dynamic computation of generalised median strings. *Pattern Analysis and Applications*, 6:185–193.

Kohonen, T. (1985). Median strings. *Pattern Recognition Letters*, 3:309–313.

Koyano, H. and Kishino, H. (2010). Quantifying biodiversity and asymptotics for a sequence of random strings. *Physical Review E*, 81(6):061912.

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

Levenshtein, V. (1965). Binary codes capable of correcting deletions, insertions and reversals. *Doklady Adademii Nauk SSSR*, 163(4):845–848.

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

Martínez-Hinarejos, C., Juan, A., and Casacuberta, F. (2003). Median strings for k-nearest neighbour classification. *Pattern Recognition Letters*, 24:173–181.

Nicolas, F. and Rivals, E. (2003). Complexities of the centre and median string problems. *Lecture Notes in Computer Science*, 2676:315–327.

Nicolas, F. and Rivals, E. (2005). Hardness results for the center and median string problems under the weighted and unweighted edit distances. *Journal of Discrete Algorithms*, 3:390–415.

Olivares-Rodríguez, C. and Oncina, J. (2008). *A Stochastic Approach to Median String Computation*, pages 431–440. Springer, Berlin.

Sim, J. S. and Park, K. (2003). The consensus string problem for a metric is NP-complete. *Journal of Discrete Algorithms*, 1:111–117.

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

Winkler, W. (1990). String comparator metrics and enhanced decision rules in the Fellegi-Sunter model of record linkage. pages 354–359.