Reducing the Computational Cost of Computing Approximated Median Strings[⋆]

Carlos D. Martínez-Hinarejos, Alfonso Juan,
Francisco Casacuberta, and Ramón Mollineda

Departament de Sistemes Informàtics i Computació
Institut Tecnològic d'Informàtica, Universitat Politècnica de València
Camí de Vera s/n, 46022, València, Spain

Abstract. The k-Nearest Neighbour (k-NN) rule is one of the most popular techniques in Pattern Recognition. This technique requires good prototypes in order to achieve good results with a reasonable computational cost. When objects are represented by strings, the Median String of a set of strings could be the best prototype for representing the whole set (i.e., the class of the objects). However, obtaining the Median String is an **NP**-Hard problem, and only approximations to the median string can be computed with a reasonable computational cost. Although proposed algorithms to obtain approximations to Median String are polynomial, their computational cost is quite high (cubic order), and obtaining the prototypes is very costly. In this work, we propose several techniques in order to reduce this computational cost without degrading the classification performance by the Nearest Neighbour rule.

1 Introduction

Many pattern classification techniques, such as k-Nearest Neighbour (k-NN) classification, require good prototypes to represent pattern classes. Sometimes, clustering techniques can be applied in order to obtain several subgroups of the class training data, where each subgroup has internal similarities [1]. Each cluster is usually represented by a prototype, and several prototypes can be obtained for each class (one per cluster). One important problem in Pattern Recognition is the selection of an appropiate prototype for a given cluster of data points.

Although the feature vector is the most common data point representation, there are many applications where strings (sequences of discrete symbols) are more appropriate as data representation (e.g., chromosomes, character contours, shape contours, etc.). The optimal prototype of a cluster of strings is the *(generalized) median string*. The median string of a given set of strings is defined as a string which minimizes the sum of distances to each string of the set. The problem of searching the median string is a **NP**-Hard problem [2]. Therefore, only approximations to median string can be achieved in a reasonable time.

⋆ Work partially supported by Spanish Ministerio de Ciencia y Tecnología under projects TIC2000-1703-C03-01 and TIC2000-1599-C02-01 and by European Union under project IST-2001-32091.

One of these approximations is the *set median string*. In this case, the search for the string is constrained to the given input set and is a polynomial problem [3,4]. In some cases, the set median string cannot be a good approximation to a median string (as in the extreme case of a set of two strings).

Other heuristic approaches were proposed in [5,6,7]. Based on the proposal presented in [5] (systematic perturbation of the set median), a new greedy simple algorithm was proposed in [8] to efficiently compute a good approximation to the median string of a set. This algorithm was improved by iterative refinement as described in [9]. Exhaustive experimentation with this algorithm and k-Nearest Neighbour classifiers is reported in [15]. The results presented in [15] show that the algorithm provides prototypes which give better classification results than set median.

In this work, we propose several methods to reduce the computational cost of the proposed algorithm. NN classifier results are presented to show the performance of the obtained prototypes.

2 Approximations to Median String

In this section, we present different methods to obtain median string approximations. Let Σ^* be the free monoid over the alphabet Σ. Given a finite set S of strings such that $S \subset \Sigma^*$, the median string of S is defined by:

$$m_S = \underset{t \in \Sigma^*}{\operatorname{argmin}} \sum_{r \in S} d(t, r) \tag{1}$$

where d is the distance used to compare two strings (usually, edit distance or normalized edit distance [10]). In other words, m_S is the string with the lowest accumulated distance to the set S. However, with this definition, no adequate median strings can be obtained (e.g., in a set with two strings, both of them achieve the minimum accumulated distance to the set). Therefore, considering the definition of the mean vector for Euclidean spaces, which uses square distances, an alternative definition (as proposed in [11]) could be:

$$m_S = \underset{t \in \Sigma^*}{\operatorname{argmin}} \sum_{r \in S} (d(t, r))^2 \tag{2}$$

As we pointed out above, computing m_S is a **NP**-Hard problem, and only approximations to m_S can be built in a reasonable time by using heuristics which attempt to optimize the accumulated distance for one of the previous definitions.

The set median string can be used as an alternative to median string. Given the set S, the set median string of S is defined as:

$$sm_S = \underset{t \in S}{\operatorname{argmin}} \sum_{r \in S} d(t, r) \tag{3}$$

that is, the search space is reduced to the set S, and not to all the free monoid Σ^*. The edit distance can be computed in a time $O(|t| \cdot |r|)$, where $|t|$ is the length of the string t. Therefore, obtaining the set median has a computational complexity of $O(|S|^2 \cdot l_S^2)$, where l_S is the maximum length of the strings from S.

Another approximation to median string can be obtained by using a refinement process. This process is based on applying the editing operations (insertion, deletion and substitution) over each position on the string, looking for a reduction of the accumulated distance defined in Equation 1 or Equation 2. This process is repeated until there is no improvement, and it needs an initial string which can be the set median string. Given a set of strings S, the specification of the process is:

For each position i in the current approximated median string M

1. Build alternatives
   Substitution: Make $M_{sub} = M$. For each symbol $a \in \Sigma$
   - Make M'_{sub} the result string of substituting the ith symbol of M by symbol a.
   - If the accumulated distance of M'_{sub} to S is lower than the accumulated distance from M_{sub} to S, then make $M_{sub} = M'_{sub}$.

   Deletion: Make M_{del} the result string of deleting the ith symbol of M.
   Insertion: Make $M_{ins} = M$. For each symbol $a \in \Sigma$
   - Make M'_{ins} the result of adding a at position i of M.
   - If the accumulated distance from M'_{ins} to S is lower than the accumulated distance from M_{ins} to S, then make $M_{ins} = M'_{ins}$.
2. Choose an alternative: from the set $\{M, M_{sub}, M_{del}, M_{ins}\}$, take the string M' with the least accumulated distance to S. Make $M = M'$.

M is the returned string at the end of the process. The time complexity of this optimization process is $O(|\Sigma| \cdot |S| \cdot l_S^3)$ for each iteration, which is a very high cost although the number iteration is low. Therefore, it will be convenient to use techniques which could reduce this cost.

3 Techniques for Reducing the Computational Cost

In this section, we introduce two techniques which allow us to reduce the computational complexity of approximated median string computations. These techniques are called division and local optimization.

3.1 The Division Technique

As we showed in Section 2, the time complexity of the process to obtain the approximated median string is cubic with the length of the strings. Therefore, it seems that reducing the length of the strings would be the most influential action on the complexity.

Following this idea, the division technique acts by dividing the strings of S into d substrings. Therefore, given a string s, this string is divided to give the strings s^1, s^2, ..., s^d such as $s = s^1 \cdot s^2 \cdots s^d$. From the set of strings S, this division provides d sets of strings S^1, S^2, ..., S^d as result. Then, an approximated median string can be obtained for each set S^1, S^2, …, S^d, that is, M^1, M^2, …, M^d and a final approximated median string of S will be $M = M^1 \cdot M^2 \cdots M^d$.

More formally, the process is:

1. $S^i = \emptyset$ for $i = 1, \ldots, d$
2. For each string $s \in S$
   (a) Divide s into s^1, s^2, …, s^d
   (b) Make $S^i = S^i \bigcup \{s^i\}$ for $i = 1, \ldots, d$
3. Compute the approximated median string M^i of S^i for $i = 1, \ldots, d$
4. $M = M^1 \cdot M^2 \cdots M^d$

The main complexity of this procedure is due to the approximated median string computation, whose complexity is still $O(|\Sigma| \cdot |S| \cdot l_S^3)$ for each iteration. Nevertheless, with the previous division, the approximated median string is computed for each S^i, where $|S^i| = |S|$ and, therefore, a total number of $d \cdot |S|$ strings are involved in the process. Futhermore, the maximum length of the strings of each S^i is $\frac{l_S}{d}$.

The final time complexity is proportional to $|\Sigma| \cdot d \cdot |S| \cdot \frac{l_S}{d}^3$, and the complexity is then $O(|\Sigma| \cdot |S| \cdot l_S^3 \frac{1}{d^2})$; that is, the real complexity is reduced by a factor d^2.

3.2 The Local Optimization Technique

From the algorithm presented in Section 2, the substitution and insertion are the edit operations which involve the majority of the edit distance calculations. This is due to the use of all the symbols in the alphabet Σ; that is, we try to substitute the current symbol by all the symbols in Σ and the insertion on the current position of all the symbols in Σ.

However, in practice, the natural symbol sequence is usually correlated so that a symbol can preceed or follow another symbol with a certain probability. This fact leads to modifying the possible symbols to be inserted or substituted on the string, using only the most likely symbols in these operations. The only way we have to determine the chosen symbols without an exhaustive study of the corpus is by using the weight matrix which is used in the edit distance calculation. In our current approach, we propose:

– For substitution: to use only the two closest symbols (according to the weight matrix) to the current one.
– For insertion: to use only the previous position symbol and its two closest symbols.

Therefore, the algorithm is modified in the following way:

For each position i in the current approximated median string M

1. Build alternatives
   Substitution: Make $M_{sub} = M$. For each symbol $a \in \text{nearest}(M_i)$
      – Make M'_{sub} the result string of substituting the ith symbol of M by symbol a.

    - If the accumulated distance of M'_{sub} to S is lower than the accumulated distance from M_{sub} to S, then make $M_{sub} = M'_{sub}$.

  Deletion: Make M_{del} the result string of deleting the ith symbol of M.

  Insertion: Make $M_{ins} = s$. For each symbol $a \in \{M_{i-1}\} \bigcup \text{nearest}(M_{i-1})$
    - Make M'_{ins} the result of adding a at position i of M.
    - If the accumulated distance from M'_{ins} to S is lower than the accumulated distance from M_{ins} to S, then make $M_{ins} = M'_{ins}$.

2. Choose an alternative: from the set $\{M, M_{sub}, M_{del}, M_{ins}\}$, take the string M' with the least accumulated distance to S. Make $M = M'$.

where nearest(M_i) gives the two nearest symbols to symbol M_i according to the weight matrix used and M is the returned string at the end of the process. The definition of nearest can be easily extended to more than two symbols.

With this modification, we avoid the factor $|\Sigma|$ in the complexity (because we only try with a low number of symbols). Therefore, the final time complexity of this approximation is $O(|S| \cdot l_S^3)$ for each iteration, which gives an assymptotic complexity reduction.

4 Experimental Framework

This section is devoted to describing the corpus we used and the experiments we carried out to compare the cost and performance of the different approximation methods described in Sections 2 and 3.

4.1 The Chromo Corpus

The data used in this paper was extracted from a database of approximately $7,000$ chromosome images that were classified by cytogenetic experts [12]. Each digitized chromosome image was automatically transformed into a string through a procedure that starts with obtaining an idealized, one-dimensional density profile that emphasizes the band pattern along the chromosome. The idealized profile is then mapped nonlinearly into a string composed of symbols from the alphabet $\{1, 2, 3, 4, 5, 6\}$. Each symbol in this alphabet represents a different absolute density level. Then, the resulting string is difference coded to represent signed differences of successive symbols, using the alphabet $\Sigma = \{=, A, B, C, D, E, a, b, c, d, e\}$ ("=" for a difference of 0; "A" for +1; "a" for -1; etc.). For instance, the string "1221114444333" is difference coded as "AA=a==C====a==". A total of 4400 samples were collected, 200 samples of each of the 22 non-sex chromosome types. See [14,13] for details about this preprocessing.

The chromosome dataset comprises 200 string samples for each of the 22 non-sex chromosome types, i.e. a total of 4400 samples. An additional piece of information contained in this dataset is the location of the *centromere* in each chromosome string. However, this position is difficult to accurately determine in a fully automatic way and, thus, we have decided not to use it in this work.

4.2 Experiments and Results

The experiments to compare the different approximations were carried out with a 2-fold cross-validation of the chromo corpus. Each class of chromosomes was divided into several clusters in order to get several prototypes for each class. Different number of clusters were obtained for each class, from 1 (i.e., no clustering) up to 9 clusters and from 10 up to 90 clusters. The set median and different approximated median strings (for both definition in Equations 1 and 2) were obtained, using the normalized edit distance and the same weights as in [15]. The optimization techniques described in section 3 were also applied, using 2 and 3 divisions in the division technique.

In this prototype extraction process, the length of the compared strings was taken as the basic cost unit, i.e., when two strings s and t are compared, the total cost is incremented in $|s| \cdot |t|$. The comparison for the different approximations using the median string definition of Equation 1 is given in Figure 1. The results using the definition given in Equation 2 are very similar. You can see the large difference from set median to all the approximated median strings (at least one order of magnitude), and also the large difference between the original method (not optimized), the division method and the local optimization method (from 3 to 5 times lower cost).

After obtaining the prototypes, several classification experiments were performed using a classical NN classifier [1] in order to quantify the degradation of the prototypes by the use of the different techniques. The results obtained

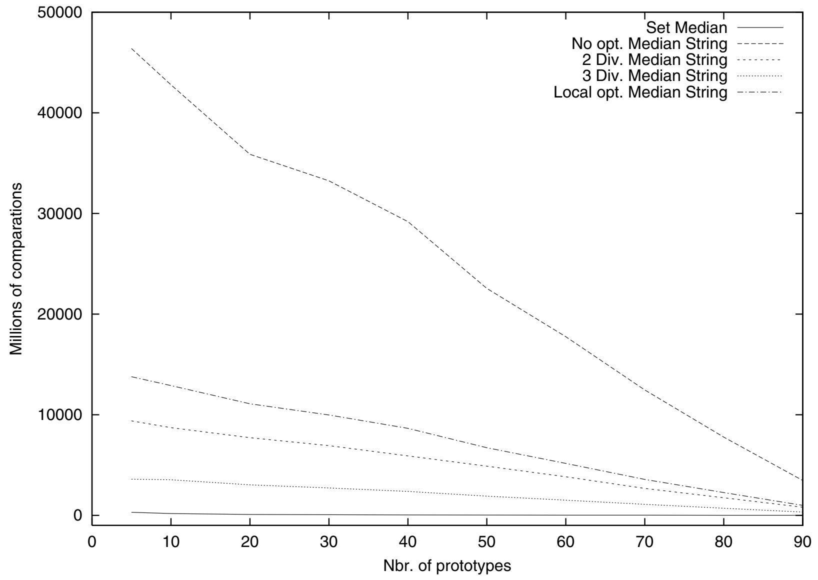

Fig. 1. Time cost in millions of comparisons performed for the different approximations to median string using the definition of Equation 1

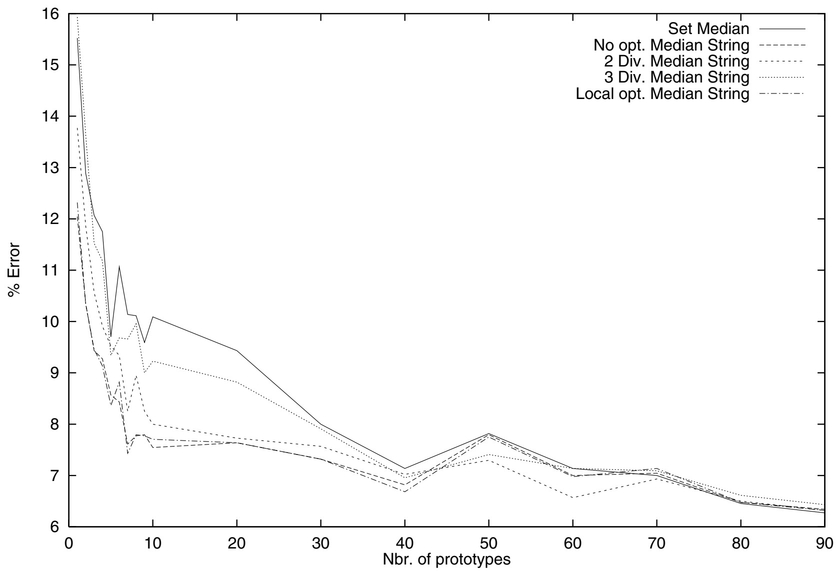

Fig. 2. Set median and approximated median string results using a NN classifier using the prototype given by the definition in Equation 1

are shown in Figure 2 (using the definition given in 1) and Figure 3 (using the definition given in 2). In both graphics, you can see that in general the median strings perform much better than set median and that the local optimization technique is the one that obtains the most similar results to the not optimized approximated median string.

5 Conclusions and Future Work

In this work, we have proposed two different techniques to reduce the computational cost to obtain an approximated median string. Even for a reduced number of divisions, the division technique provides a high reduction in the number of comparisons, although the classification results show us a clear degradation in the prototypes quality when increasing the number of divisions. The local optimization technique gives a lower reduction than the division technique, but the given prototypes are as good as those obtained by the not optimized process.

However, these conclusions are limited because the experiments were carried out with only the chromosome corpus and using only a NN classifier. Therefore, future work is directed to extending these conclusions using other corpora and more powerful classifiers (as k-NN classifiers), and to verify the effects of combining both techniques.

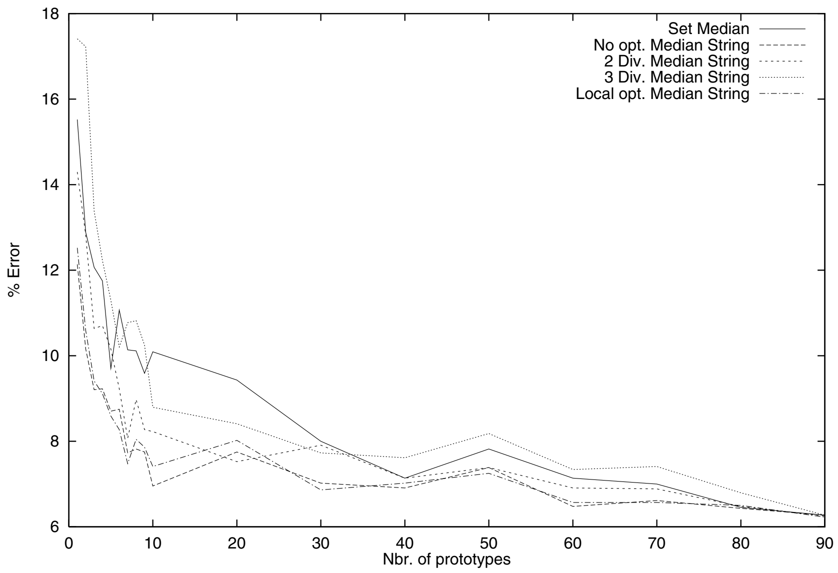

Fig. 3. Set median and approximated median string results using a NN classifier using the prototype given by the definition in Equation 2

Acknowledgements

The authors wish to thank Dr. Jens Gregor for providing the preprocessed chromosome data used in this work, and the anonymous reviewers for their criticism and suggestions.

References

1. Duda, R. O., Hart, P., Stork, D. G., 2001. Pattern Classification. John Wiley. 47, 52
2. de la Higuera, C., Casacuberta, F., 2000. The topology of strings: two np-complete problems. Theoretical Computer Science 230, 39–48. 47
3. Fu, K. S., 1982. Syntactic Pattern Recognition. Prentice-Hall. 48
4. Juan, A., Vidal, E., 1998. Fast Median Search in Metric Spaces. In: Proceedings of the 2nd International Workshop on Statistical Techniques in Pattern Recognition. Vol. 1451 of Lecture Notes in Computer Science. Springer-Verlag, Sydney, pp. 905–912. 48
5. Kohonen, T., 1985. Median strings. Pattern Recognition Letters 3, 309–313. 48
6. Kruzslicz, F., 1988. A greedy algorithm to look for median strings. In: Abstracts of the Conference on PhD Students in Computer Science. Institute of informatics of the József Attila University. 48
7. Fischer, I., Zell, A., 2000. String averages and self-organizing maps for strings. In: Proceeding of the Second ICSC Symposium on Neural Computation. pp. 208–215. 48

8. Casacuberta, F., de Antonio, M., 1997. A greedy algorithm for computing approximate median strings. In: Proceedings of the VII Simposium Nacional de Reconocimiento de Formas y Análisis de Imágenes. pp. 193–198. 48
9. Martínez, C. D., Juan, A., Casacuberta, F., 2000. Use of Median String for Classification. In: Proceedings of the 15th International Conference on Pattern Recognition. Vol. 2. Barcelona (Spain), pp. 907–910. 48
10. Vidal, E., Marzal, A., Aibar, P., 1995. Fast computation of normalized edit distances. IEEE Transactions on Pattern Analysis and Machine Intelligence 17 (9), 899–902. 48
11. Martínez, C., Juan, A., Casacuberta, F., 2001. Improving classification using median string and nn rules. In: Proceedings of IX Simposium Nacional de Reconocimiento de Formas y Análisis de Imágenes. pp. 391–394. 48
12. Lundsteen, C., Philip, J., Granum, E., 1980. Quantitative Analysis of 6895 Digitized Trypsin G-banded Human Metaphase Chromosomes. Clinical Genetics 18, 355–370. 51
13. Granum, E., Thomason, M., 1990. Automatically Inferred Markov Network Models for Classification of Chromosomal Band Pattern Structures. Cytometry 11, 26–39. 51
14. Granum, E., Thomason, M. J., Gregor, J. On the use of automatically inferred Markov networks for chromosome analysis. In C Lundsteen and J Piper, editors, *Automation of Cytogenetics*, pages 233–251. Springer-Verlag, Berlin, 1989. 51
15. Martínez-Hinarejos, C. D., Juan, A., Casacuberta, F., Median String for k-Nearest Neighbour classification, *Pattern Recognition Letters*, acepted for revision. 48, 52