IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received December 1, 2021, accepted December 15, 2021, date of publication December 23, 2021,
date of current version December 31, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3137767

Boosting Perturbation-Based Iterative Algorithms to Compute the Median String

PEDRO MIRABAL[1], JOSÉ ABREU[2], DIEGO SECO[3,4], ÓSCAR PEDREIRA[5], AND EDGAR CHÁVEZ[6]

[1]Informatics Engineering Department, Universidad Católica de Temuco, Temuco 4813302, Chile
[2]Instituto Universitario de Investigación Informática, Universidad de Alicante, 03690 Alicante, Spain
[3]Computer Science Department, Universidad de Concepción, Concepción 3349001, Chile
[4]Millennium Institute for Foundational Research on Data, Chile
[5]Database Laboratory, Universidade da Coruña, 15001 A Coruña, Spain
[6]Centro de Investigación Científica y de Educación Superior de Ensenada, Ensenada 22860, México

Corresponding author: Pedro Mirabal (pedro.sanchez@uct.cl)

This work was supported in part by the Comisión Nacional de Investigación Científica y Tecnológica - Programa de Formación de Capital Humano Avanzado (CONICYT-PCHA)/Doctorado Nacional/2014-63140074 through the Ph.D. Scholarship, in part by the European Union's Horizon 2020 under the Marie Skłodowska-Curie under Grant 690941, in part by the Millennium Institute for Foundational Research on Data (IMFD), and in part by the FONDECYT-CONICYT under Grant 1170497. The work of ÓSCAR PEDREIRA was supported in part by the Xunta de Galicia/FEDER-UE refs under Grant CSI ED431G/01 and Grant GRC: ED431C 2017/58, in part by the Office of the Vice President for Research and Postgraduate Studies of the Universidad Católica de Temuco, VIPUCT Project 2020EM-PS-08, and in part by the FEQUIP 2019-INRN-03 of the Universidad Católica de Temuco.

ABSTRACT The most competitive heuristics for calculating the median string are those that use perturbation-based iterative algorithms. Given the complexity of this problem, which under many formulations is NP-hard, the computational cost involved in the exact solution is not affordable. In this work, the heuristic algorithms that solve this problem are addressed, emphasizing its initialization and the policy to order possible editing operations. Both factors have a significant weight in the solution of this problem. Initial string selection influences the algorithm's speed of convergence, as does the criterion chosen to select the modification to be made in each iteration of the algorithm. To obtain the initial string, we use the median of a subset of the original dataset; to obtain this subset, we employ the Half Space Proximal (HSP) test to the median of the dataset. This test provides sufficient diversity within the members of the subset while at the same time fulfilling the centrality criterion. Similarly, we provide an analysis of the stop condition of the algorithm, improving its performance without substantially damaging the quality of the solution. To analyze the results of our experiments, we computed the execution time of each proposed modification of the algorithms, the number of computed editing distances, and the quality of the solution obtained. With these experiments, we empirically validated our proposal.

INDEX TERMS Approximate median string, algorithm initialization, half space proximal neighbors.

I. INTRODUCTION

The median string problem has attracted the attention of the scientific community in different domains, from early work by Kohonen [10], through word recognition and prototyping. An example of the above is encoding as strings of representative shapes [31], handwritten character recognition [2] or prototyping as a way to condense a dataset [50]. In classification tasks, an approximation to the median string is better as a prototype compared with taking as a prototype a string from the set accumulating the least distance from the rest [15]. Although they show rapid convergence, some heuristics hardly manage to improve the quality of the approximation to the median string concerning the starting point. In contrast, heuristics that converge to better solutions to problems such as character string classification [50] can take tens of hours to find a solution [28], [42]. So work must be done to improve the speed of these algorithms.

Even though there are algorithms that manage to find the exact solution for this problem [11], the computational cost is exponential. For a set of strings of size $|S|$ and length l, a time $\mathcal{O}(l^{|S|})$ is needed. Furthermore, various authors have shown that the mean chain problem has W [1] - Hard complexity in $|S|$ even for binary alphabets for the Levenshtein

The associate editor coordinating the review of this manuscript and approving it for publication was Diego Oliva.

distance case. Examples of these approaches are greedy algorithms as in [4], [26].

Many heuristics can be classified as disturbance-based iterative refinement algorithms. An initial seed or string is modified through editing operations (also named perturbations in the literature) to get closer to the median string. There are different strategies for selecting the editing operation to be tested. For example, testing possible insertions, deletions, and substitutions in a pre-established order without estimating the probability that the operation will be successful [15]. In [1] authors proposes a different approach, ranking operations by an index allowing testing the most promising first, a better variant of the said index was proposed in [28]. We cover both approaches in Section II.

A. SUMMARY OF CONTRIBUTIONS

This work proposes two variations to the perturbation-based iterative refinement algorithms to compute the median string that improves their performance. The first variation affects the algorithm's initialization since an alternative to the median string belonging to the dataset is sought, which was the best initialization proposed in previous works. The new initialization consists in computing the median of a subset of the original dataset. We propose this subset to by the Half Space Proximal neighbors of the median of the string set. The second proposed variation modifies the stop condition; we propose trimming the list of possible edits in each iteration. The above trimming befalls when the expected quality of the operations is less than a threshold; in our case, we took 0 as the threshold.

This paper continues with Section II, where we review relevant concepts and the related works. In Section III, we describe our proposal for a new initialization of the algorithms and the modification in the stop condition. In Section IV, we present the set of experiments carried out and discuss the results, comparing them with other relevant algorithms. Ending with Section V, where the conclusions we arrived at are presented.

II. PRELIMINARIES AND RELATED WORK

As described in [1], [28], the Median String problem can be defined as follows. Let Σ be an alphabet, let Σ^* be the set of all strings over Σ, and let ϵ the empty symbol over this alphabet. For two strings S_i, $S_j \in \Sigma^*$, an edit operation is a pair $(a, b) \neq (\epsilon, \epsilon)$, written $a \rightarrow b$, which transforms a string S_i into S_j, if $S_i = \sigma a \tau$ and $S_j = \sigma b \tau$, where σ and τ represent substrings.

We denote as $E_{S_i}^{S_j} = \{e_1, e_2, \ldots, e_n\}$ the sequence of edit operations transforming S_i into S_j. The cost of $E_{S_i}^{S_j}$ is $\omega(E) = \sum_{e_i \in E} \omega(e_i)$ and the edit distance from S_i to S_j is defined as $d(S_i, S_j) = argmin_E\{\omega(E_{S_i}^{S_j})\}$. The *median string* is a string that minimizes $\sum_{S_i \in S} d(\hat{S}, S_i) | \hat{S} \in \Sigma^*$.

There are multiple approaches to tackle the problem of the median string, in this paper, we focus on those algorithms based on perturbations. Kohonen originally conceived this idea in [10], who proposed, starting from the median of the set, systematically mutate each of its symbols, verifying if the sum of the distances decreased. In that work, the authors established no criteria to evaluate the feasibility of each of the possible modifications.

Perturbations are called each of the possible editing operations defined for the Levenshtein editing distance [13], that is, the substitution of a a symbol for a b symbol, the elimination of a symbol a or inserting a b symbol. We will denote each of these operations, as well as their cost, by $w(a, b)$, $w(a, \epsilon)$, and $w(\epsilon, b)$. One of the interpretations of the edit distance is to find the sequence of transformations of the string s_i to s_j in such a way that the sum of the cost of each operation involved is the minimum.

Some authors such as [4], take the empty string as the initial string $\hat{s}$, perform a greedy search adding at each iteration at the end of the string the symbol that is estimated to lead to the lowest sum of distances. First, $\sum_{s_i \in S} d(\hat{s} + c_j, s_i)$ for each $c_j \in \Sigma$ is calculated using dynamic programming [49], storing the row relative to c_j from the dynamic programming array. The sum of the minimum value in each row yields the estimate of how promising c_j is; the lower, the better. The process is repeated until neither c_j leads to improvement or the length of $\hat{s}$ matches the largest string in S. In [12] this procedure is enhanced with a different quality estimate, as well as a procedure for solving cases where more than one symbol has the same quality.

In [5] authors introduced the idea of performing multiple operations simultaneously to speed up calculation. The algorithm starts from the median established as the initial approximation $\hat{s}$. In each iteration, they compute the distances from $\hat{s}$ to all the strings in S, registering the list of edit operations in each case. After that, for each position, i of $\hat{s}$ the most frequent operation is applied. This process is repeated until there are no further improvements. Experiments show that the algorithm is faster than [10]; however, they provide no details on the quality of the solution. Later the above strategy was revisited in [44], where they show that there may be an inverse relationship between convergence speed and solution quality since as the number of simultaneous operations increases, the quality of the solution deteriorates. The authors also stated that this could slightly increase the probability of finding the global optimum since algorithms that change only one symbol at a time can get stuck at a local optimum.

In [14] authors propose one operation at a time approach, which appraises a specific order to perform operations. For each position i of $\hat{s}$, each possible substitution of the i-th symbol is applied to identify $\hat{s}_{sub}$, the string with the smallest sum of distances. The string $\hat{s}_{ins}$ is calculated in a similar way and $\hat{s}_{del}$ results from the elimination of the symbol i. The new candidate is the best among the three options and $\hat{s}$; then the process continues for $i+1$. The authors also evaluated two optimizations to reduce the algorithm's $\mathcal{O}(|\sigma| \times |S| \times L^3)$ time. The division technique explored

dividing each string in S into d substrings. In this way, starting from a string $s \in S$ results in strings $s_1, s_2, .., s_d$. The procedure independently calculates the median for each resulting set $\hat{s}^1, \hat{s}^2, \ldots, \hat{s}^d$ that they are concatenated to obtain the median of S. Another improvement is related to inserts and substitutions, which account for most of the computation time. For substitutions, only the two symbols closest to the one at position i are evaluated, and for inserts, only the symbol at position $i-1$ and its two closest symbols. The above allows avoiding the $|\Sigma|$ factor. They compare the three choices in a classification task where the median is a prototype for the classification k -NN, but without considering the sum of distances. Regardless, the results suggest that splitting optimization leads to faster convergence but lower quality prototypes.

Another work that advocates simultaneous operations is [3]. Even though the proposal is not an algorithm for the median string, it can be easily adapted. The main loop is similar to [5] calculating the distance from $\hat{s}$ to the strings in S to get the frequency of edit operations. However, only operations with a frequency more significant than a threshold of η are considered. The authors focused on prototyping for classification, they report no experiments to assess the quality of the median chain.

More recently, [1] improved the results in [14] by applying perturbations one at a time. When calculating the distance from $\hat{s}$ to strings in S, the authors also take into account the editing operations. The hypothesis is that the operations on both sequences have a better chance of improving the solution given the results in [2], which states that if we apply to $\hat{s}$ one of the operations of the edit sequence $d(\hat{s}, s_i)$ to get $\hat{s}'$ then $d(\hat{s}', s_i) <= d(\hat{s}, s_i)$. The above is the underlying idea in [3], [5] but the authors also took into account the cost of the operation to estimate how much the sum of distances would reduce by applying the operations. The estimated probability, from highest to lowest, determines the order in which operations are evaluated. In cases where an edit operation is not present in all edit sequences, the heuristic is optimistic as it does not consider how the application of the operation will affect the distance from $\hat{s}'$ to the strings that do not request this operation. In [28] authors tackled the problem offering an estimate for some of the cases with a significant improvement in convergence rate without affecting the quality of the solution. A comparison between algorithms performing multiple operations simultaneously [3], [5] and one operation at a time [1], [14], [28] suggests that the first ones have fast convergence with a lower approximation quality. The above can be explained by the side effect of editions in other positions, as suggested by [5].

Recently, [46] describes the applications of median string in DNA motif classification, where the median string is computed using Markov chains. Also, a prototype generation in the string space via approximate median for data reduction in nearest neighbor classification is presented in [48].

III. OUR PROPOSAL

Most algorithm start from the empty string [1], [5] or from the string belonging to the dataset that accumulates the least distance to the rest, understood as the median of the dataset [1], [4], [10], [14], [28], [42]–[44]. Our goal is to propose an initialization alternative since, from the works above, initialization affects the speed of convergence of the algorithms.

A. SELECTING A BETTER INITIALIZATION USING THE HALF SPACE Proximal(HSP) TEST

The Half Space Proximal (HSP) graph was originally defined by Chávez *et al.* [19], and it is a sparse subgraph of the complete graph in a metric space. For the construction of this graph, it needs to separately apply the *HSP test* to each object in $S_i \in S$. The output of the HSP test on S_i is a partition of S, $\mathcal{P} = \{P_1, P_2, \ldots\}$, where each P_j has a representative object p_{jr}, that is connected to S_i in the HSP graph. One advantage of this test is reducing the number of edges of the complete graph. The connection of the vertices in the HSP guarantees two desirable properties for our work: proximity and diversity.

The HSP test proceeds iteratively. Let S' be the set of strings in iteration i (when $i = 0$, $S' = S$). In each iteration i, the algorithm finds the nearest neighbor of m in S', $p_{ir} = kNN(m, S')$. All the strings that are closer to p_{ir} than to m form a new subset P_i of S', where p_{ir} is the representative of P_i. All the strings in P_i are removed from S'. This process repeats until S' is empty. After this, every string is reassigned to the subset corresponding to the closest representative.

In Fig. 1, we show a pseudo-code of the implementation used by us of the Half Space Proximal test. In our case, we previously removed from the dataset the element m. The output of this algorithm is a set of disjoint subsets P_i derived from the original dataset. All elements of $P_i \in P$ has a representative p_{ir} that is the closest element to m within P_i. We initialize P as an empty set (line 1), and iterates until every string in S is added to some subset $P_i \in P$ (lines 2-15). In each iteration, we search for the nearest neighbor of m, $nn \in S$ (lines 4-7), then, all closer to nn than to m are removed from S and added to Q (lines 8-13), including nn thats become the representative of Q. The set Q is added to P (line 14) and a new iteration begins if S is not empty. From line 16 to line 31, each element $p_{ij} \in P_i$, such that $p_{ij} \neq p_{ir}$, is reassigned to the nearest representative. This part of the algorithm takes $\mathcal{O}((n-p) \times p)$ where n is the size of S and p is the number of subsets in P.

In Fig. 2 we illustrate an example of the Half Space Proximal test. To simplify, we show a possible spatial relation of the different strings in the set S in terms of distance between them. In Fig. 2(a), the star represents the set median $m \in S$. Then, in Fig. 2(b), string s_1, represented by a white square, is selected as the closest to m, and space is split into two parts, any string closer to s_1 than to m is assigned to P_1 and $p_{1r} = s_1$. In Fig. 2(c), the string $s_2 \notin \bigcup_{i=1}^{n} P_i$, represented by a white triangle, is selected as the closest to m and the previous

Algorithm 1: HSP test pseudo-code.

```
Input  : Set S, Set Median m
Output : Set of Sets P
1  P = ∅
2  while S ≠ ∅
3   do
4      Q = ∅
5      nn = argmin{distance(S, m)}
6      Q.add(nn)
7      S.remove(nn)
8      foreach S_i ∈ S do
9          if distance(S_i, nn) < distance(S_i, m) then
10             Q.add(S_i)
11             S.remove(S_i)
12         end if
13     end foreach
14     P.add(Q)
15 end while
16 foreach P_i ∈ P do
17     foreach P_ij ∈ P_i; j > 1 do
18         minDistance = distance(P_ij, P_i0)
19         minP = i
20         foreach P_k ∈ P do
21             if distance(P_ij, P_k0) < minDistance then
22                 minDistance = distance(P_ij, P_k0)
23                 minP = k
24             end if
25         end foreach
26         if minP ≠ i then
27             P_minP.add(P_ij)
28             P_i.remove(P_ij)
29         end if
30     end foreach
31 end foreach
32 return P
```

FIGURE 1. HSP test pseudo-code.

process is repeated to build P_2. This process continues, selecting the nearest string s_3, represented by a white diamond, and building P_3 and so on, until all strings become part of the set $P = \{P_1, P_2, \ldots, P_n\}$. In our example, we can see in Fig. 2(d) that all strings are already assigned. The last stage of the test is to reassign all strings $s_i \neq p_{ir}$ to the P_i with the p_{ir} closest to s_i. The final reassignment is shown in Fig. 2(e).

We propose Algorithm 3 for calculating the median string when strings in the dataset have different weights. This algorithm takes as input an instance set S, an initialization string R, and a weight set W that is a vector that represents the corresponding weight of each element in S. The algorithm iterates through the same steps until no editions applied to $\hat{S}$ improve the result. First, the distances between R' and each S_i and the respective involved editions $E_{S_i}^{R'}$ are computed (lines $4-7$). Each edition in $E_{S_i}^{R'}$ have an associated weight W_i used to update the statistics (line 6). In lines, $8-14$, the repercussion of each edition affecting the same position is computed, generating a goodness index of editions. All editions are inserted in a priority queue Q, sorted by goodness index. Then, we discard from Q all editions q_i with $q_i.goodnessIndex \leq 0$ (line 16). Next, we dequeue editions from Q to obtain a new candidate R', applying e_k to $\hat{S}$ (lines $18-19$). These two steps are repeated while the new candidate R' is worse than $\hat{S}$ and Q is not empty. Finally, the algorithm returns $\hat{S}$. It is worth noting that the vector W for strings in S can be computed using any weighting procedure. In our particular setup, we use the output of Algorithm 1 as follows. The set $S = \{p_{ir} \in P_i\}$ and $W = \{|P_i|\}$.

This algorithm iterates by considering one perturbation at a time until it does not improve during the iteration. Each iteration may consider several different editions. In the worst case, this is upper-bounded by $O(l \times \Sigma^2)$, where l is the length of the longest string. The experimental evaluation shows that this bound is rather pessimistic and that our heuristic usually needs just a few editions per iteration. The above is a crucial difference with the algorithm in [1], which uses more operations per iteration.

For each edition explored during an iteration, the algorithm computes the distance of the new candidate R' to all the elements in S (lines $17-20$), which takes time $O(N \times dc)$, where dc is the time to compute the edit distance (Levenshtein in our experiments). By providing a better ranking, we save on the number of operations explored per iteration, and thus, on the number of times this distance is computed, which is expensive. In the case of Levenshtein, for example, it is $O(l^2)$. However, to do that, we expend some computations to bound the *repercussion* (lines $8-14$). This takes $O(l \times \Sigma^2)$ time and it is usually worth it as $\sqrt{l} \geq \Sigma$ in most applications.

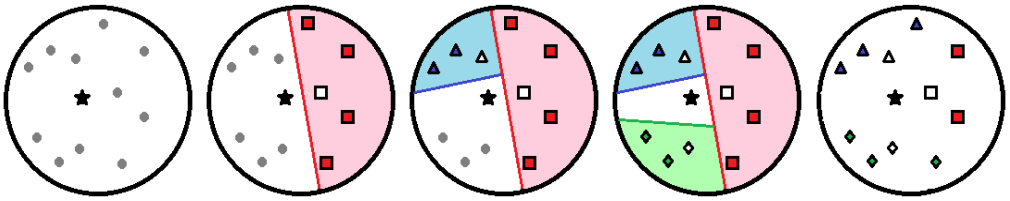
B. TRIMMING THE LIST OF POSSIBLE EDIT OPERATIONS

The algorithm presented by [28], generates a goodness index for each edition, taking into account how this edition impacts other editing alternatives affecting the same position. This goodness index is more precise than considering only editions frequency or the frequency multiplied by the cost as proposed in [1].

In Fig. 3, it is possible to see on line 15 that their goodness index sorts the operations but, unlike in [1], [28], we propose to trim Q discarding those editing operations that have a negative value of goodness index, line 16. This modification can reduce the size of Q, speeding up the while loop in lines 17-20 because it runs until an improvement is achieved or Q is empty.

The basis of the algorithm in Fig. 4, was presented in [28]. The original algorithm considered all strings $S_i \in S$ with the same relevance. We have made the necessary modifications so that strings $S_i \in S$ can have different weights. It is important to notice that, unlike in [1], [14], [15], [28], for the calculation of the median of S' we use an approach in which each S'_i is weighted according to the size of the subset P_i that it represents. The idea of weighting strings has been studied in [27], but only when computing the median of two strings.

Combining the modifications described in Section III-A and in Section III-B, we have four different algorithms,



(a) Starting point. (b) 1st neighbor. (c) 2nd neighbor. (d) 3rd neighbor. (e) Reassignment.

FIGURE 2. HSP test step by step.

Algorithm 2: Median String Repercussion with weighs.

Input : instance set S, initialization string R, weight set W
Output : approximate median string $\hat{S}$

$R' = R$
repeat
  $\hat{S} = R'$
  foreach $S_i \in S$ **do**
    compute $d(R', S_i)$ and the respective $E^{R'}_{S_i}$
    statistics.update($E^{R'}_{S_i}, W_i$)
  end foreach
  foreach *position* j *of* R' **do**
    foreach *symbol* $si \in \Sigma$ **do**
      foreach *symbol* $sj \in \Sigma$ **do**
        $Rep[j][si] += \omega(R[j] \rightarrow si) - \omega(si \rightarrow sj)$
      end foreach
    end foreach
  end foreach
  Q: a priority queue of editions sorted by goodness index
  while $\sum_{S_i \in S} d(\hat{S}, S_i) \leq \sum_{S_i \in S} d(R', S_i)$ **and** $Q \neq \emptyset$ **do**
    $e_k = Q.dequeue$
    obtain a new candidate R' applying e_k to $\hat{S}$
  end while
until *no editions* e_k *applied to* $\hat{S}$ *improve the result*
return $\hat{S}$

FIGURE 3. Median String Repercussion with weighs.

Algorithm 3: AppMedianStringRepercussion(S,R) :$\hat{S}$

Input : instance set S, initialization string R
Output : approximate median string $\hat{S}$

$R' = R$
repeat
  $\hat{S} = R'$
  foreach $S_i \in S$ **do**
    compute $d(R', S_i)$ and the respective $E^{R'}_{S_i}$
    statistics.update($E^{R'}_{S_i}$)
  end foreach
  foreach *position* j *of* R' **do**
    foreach *symbol* $si \in \Sigma$ **do**
      foreach *symbol* $sj \in \Sigma$ **do**
        $Rep[j][si] += \omega(R[j] \rightarrow si) - \omega(si \rightarrow sj)$
      end foreach
    end foreach
  end foreach
  Q: a queue of operations sorted by goodness index
  while $\sum_{S_i \in S} d(\hat{S}, S_i) \leq \sum_{S_i \in S} d(R', S_i)$ **and** $Q \neq \emptyset$ **do**
    $e_k = Q.dequeue$
    obtain a new candidate R' applying e_k to $\hat{S}$
  end while
until *no operation* e_k *applied to* $\hat{S}$ *improve the result*
return $\hat{S}$

FIGURE 4. AppMedianStringRepercussion(S,R) :$\hat{S}$.

labeled as *Median-all*, *HSP-all*, *Median-trimmed* and *HSP-trimmed*. The first part of the algorithms name refers to the initial string, having *Median* for those that have as initial string the set median and *HSP* for the ones that apply the modifications described in Section III-A. The second part of the names refers to how we deal with the operation list. We use *all* for those that test the whole operation list and *trimmed* for the ones that apply the modifications described in Section III-B.

IV. EXPERIMENTAL RESULTS

Our experimental evaluation uses different alphabets, set sizes, and string lengths. In Eq. 1, we show the ratio used to evaluate the quality of the obtained median string $\hat{S}$, where S^M is the set median. Besides, we compare the number of edit distances required by the algorithms to converge. Also, we took into consideration the execution time for each experiment. As expected, in all the experiments, time was proportional to the number of edit distances calculated.

$$\frac{\sum_{S_i \in S} d(\hat{S}, S_i)}{\sum_{S_i \in S} d(S^M, S_i)} \tag{1}$$

Three different datasets were used. The first one corresponds to the Freeman Chain Code of Eight Directions [6], which represents contours of letters from the *NIST-3* database. This codification is also used in [1], [7], [8], [16], [28]. In this dataset, we evaluated set sizes of {45, 90, 180, 270, 360} and, for each one, 26 independent samples were drawn.

The second dataset considers 23 symbols representing different amino acids. We selected 175 samples of orthologous of insulin protein, representing 70 species, obtained from eggNog online application [1] with length ranging in [100, 300] and average 150. With them, we prepared 26 different sets in total, 5 different for each of the sets of size {20, 40, 80, 120, 160} strings, respectively, and 1 set with size 175 with all the data available. We use the well-known BLOSUM62 [18] cost function.

We also generated a third dataset containing synthetic Freeman chain codes as in [1], [16], [28]. With these data, we aimed to study how algorithms scale for sets with sizes of {45, 90, 180, 270, 360}, with the average length of the strings of {20, 40, 80, 160, 320} symbols, respectively. The length variation among strings in the same set was 10%. We generated 5 different sets for each possible combination of set size and string length, making a total of 125 independent sets.

We designed experiments to compare our proposal with the best algorithm described in [28], labeled as *Median-all*, and in [1], labeled *Frequency* and *Frequency*Cost*, in terms of edit distances calculated, average distance to median string and time.

In Fig. 5, we can differentiate three groups of algorithms, at the top, we see the algorithms labeled as *Frequency* and *Frequency*Cost*, exposed in [1]. These two algorithms are those with the highest number of edit distances calculated, which increases very rapidly as the size of the dataset grows. In the central region, we see the algorithm labeled as *Median-all*, presented in [28], and a variant that takes as a starting point the one described in Section III, labeled *HSP-all*. These two algorithms perform better in comparison with those mentioned above. As the dataset size grows, differences between them are more evident. Finally, in the bottom part of Fig. 5, the algorithms *Median-trimmed* and *HSP-trimmed* are shown. These two algorithms are the ones that perform the best.

In Fig. 6 the quality of the solution achieved by the same algorithms is studied. As can be seen, the quality of the two algorithms computing fewer edit distances is slightly worse. In Fig. 6, we can differentiate two groups of algorithms, at the top, we see the algorithms labeled as *Median-trimmed* and *HSP-trimmed*. Except for these two algorithms, the others behave similarly regarding the quality of the obtained median string. Finally, as expected, Fig. 7 shows a similar behavior to Fig. 5, i.e. the running time of the method is proportional to the edit distances that they compute.

[1] http://eggnogdb.embl.de/#/app/home

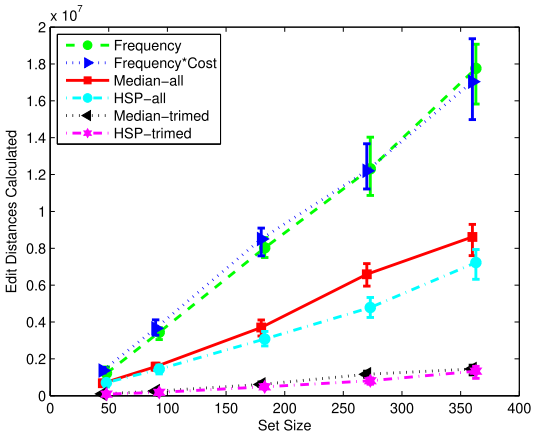


FIGURE 5. Synthetic Freeman Chain Codes: Edit Distances Calculated.

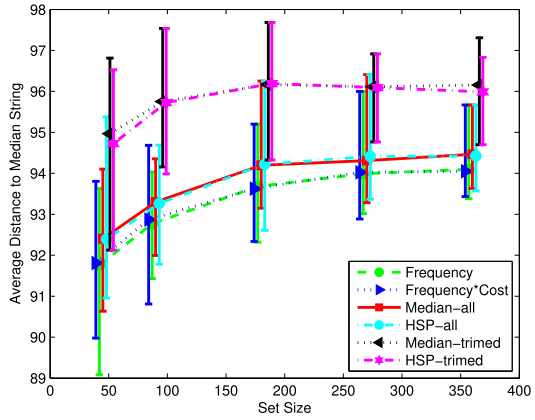


FIGURE 6. Synthetic Freeman Chain Codes: Average Distance to Median String.

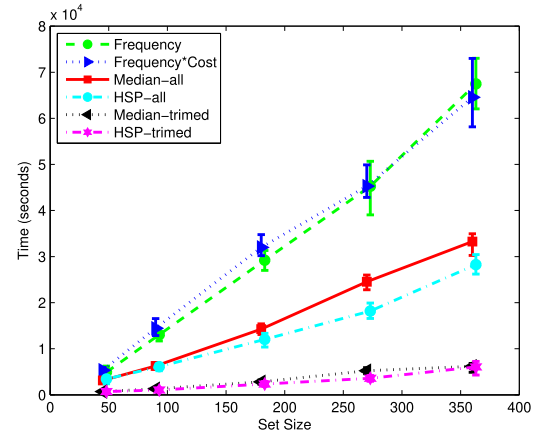


FIGURE 7. Synthetic Freeman Chain Codes: Time.

Next, we expose in detail the effect of each modification when applied independently. From Fig. 8 to Fig. 16, we see the comparison between the algorithm that takes as a starting

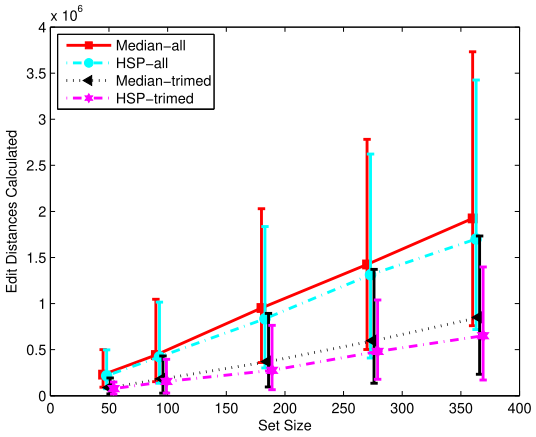


FIGURE 8. Edit Distances Calculated: NIST Freeman Chain Codes.

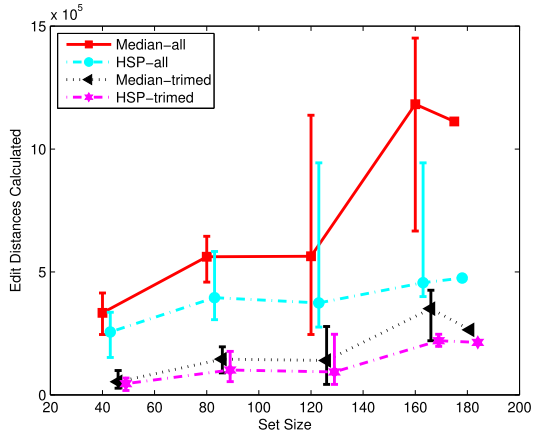


FIGURE 9. Edit Distances Calculated: Proteins. String.

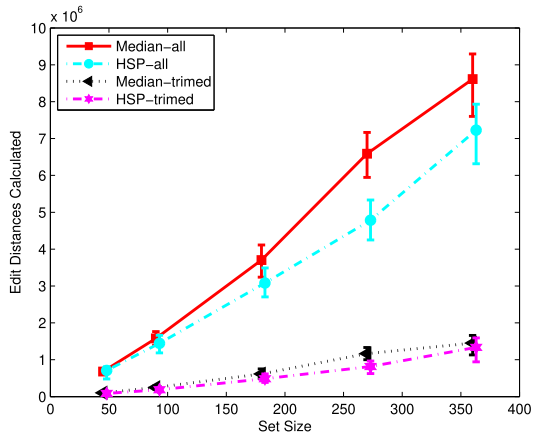


FIGURE 10. Edit Distances Calculated: Synthetic Freeman Chain Codes.

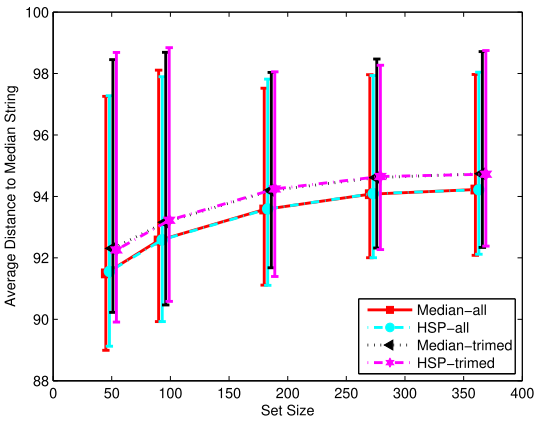


FIGURE 11. Average Distance to Median String: NIST Freeman Chain Codes.

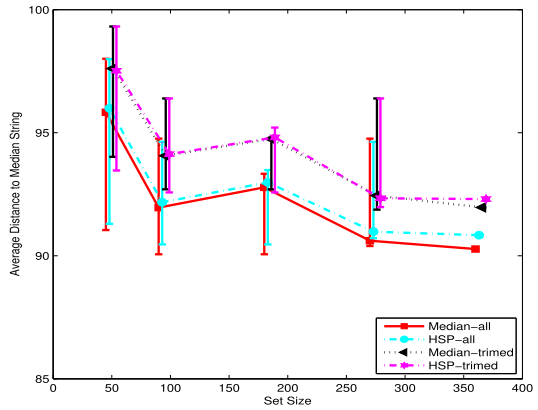


FIGURE 12. Average Distance to Median String: Proteins. String.

point the one proposed in Section III, labeled as *HSP-all*, comparing it with the same algorithm starting from the set median, labeled as *Median-all*. It is essential to clarify that we include the edit distances calculated to obtain the starting point for those algorithms using the HSP test. For all the datasets, results show that in most of the cases our proposal requires fewer operations, while, as can be seen in Fig. 11 and in Fig. 13, the quality of the median string obtained, in the Freeman Chain Codes datasets, is equivalent in both cases. Fig. 12 shows that, in the proteins dataset, the median string is slightly worse for *HSP-all*.

We show the comparison between algorithms with the same starting point, trimming the list of operations and without trimming it. In this analysis, two new algorithms are incorporated besides the *Median-all* and the *HSP-all*, explained in detail in the previous section. We label as *Median-trimmed* the algorithm that takes as a starting point the median of the set and trims the list of operations when the expected quality of the operation is zero. We label as *HSP-trimmed* the algorithm that takes as a starting point the one proposed previously in Sec. III and trims the list of operations when the expected quality of the operation is zero.

In Fig. 8, Fig. 9, Fig. 10, Fig. 14, Fig. 15, and Fig. 16 we notice that trimming can reduce the number of calculated edit

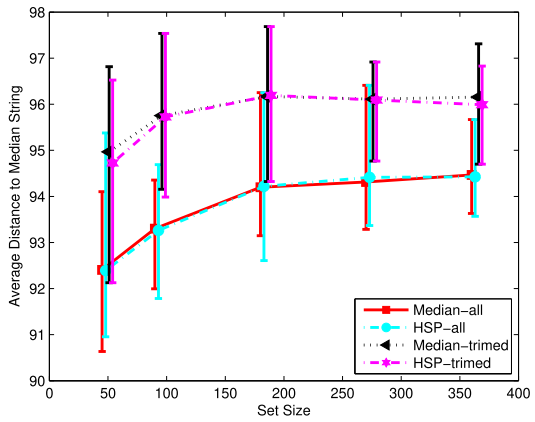


FIGURE 13. Average Distance to Median String: Synthetic Freeman Chain Codes.

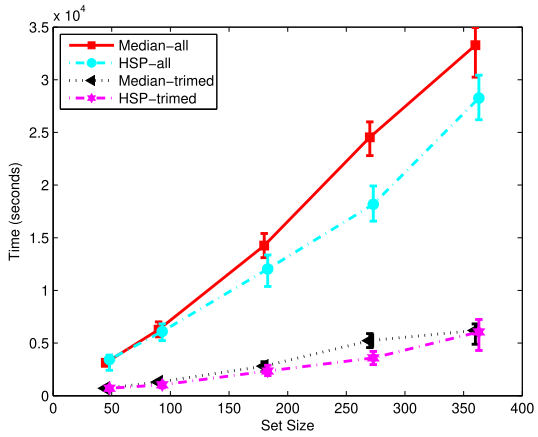


FIGURE 16. Time: Synthetic Freeman Chain Codes.

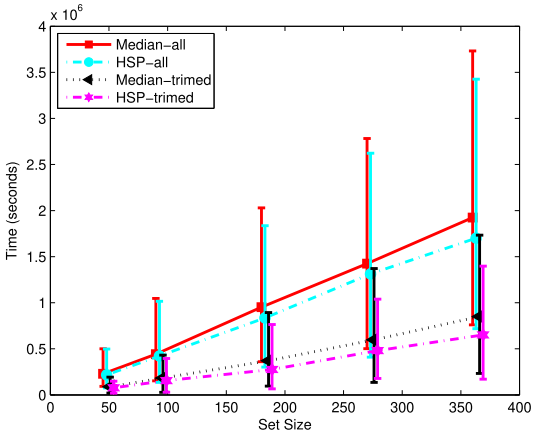


FIGURE 14. Time: NIST Freeman Chain Codes.

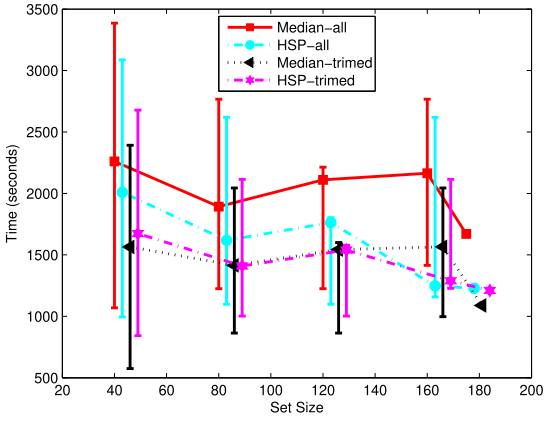


FIGURE 15. Time: Proteins. String.

distances, and thus, lead to a decrease in the execution time. However, the quality of the median is slightly worse when trimming, as it can be seen in Fig. 11, Fig. 12, and Fig. 13.

Finally, we can see the significant difference that exists, concerning the edit distances calculated and execution time, between the current state of the art, *Median-all*, and our new proposal, *HSP-trimmed*. We can also compare the quality of the median string achieved for each of the different algorithms. The results show that the loss of quality of the median string in *HSP-trimmed* is small, and can be assumed for cases in which a high speed of convergence is required.

V. CONCLUSION

A new starting point can be used with satisfactory results in perturbation-based iterative refinement algorithms to compute the median string. We obtain this new starting point from computing the median of a subset of the original dataset. The string subset consists of the Half Space Proximal neighbors of the median string. The above modification implied weighting the elements of the subset depending on the number of instances they represented. We also show that trimming the list of operations improved the stop condition of these algorithms. The above trimming occurs when the expected quality of the operations is less than a threshold, 0 in our case.

The combination of the two heuristics above in our approach produce a more competitive solution than SOTA algorithms. Comparing *Median-all* and *HSP-trimmed* we reduced edit distance computations by 86% on average. Similarly, we decreased execution time 82% on average. Reductions in execution time and the number of computed edit distances induced a slight increase of 2% in the average distance to the median string.

REFERENCES

[1] J. Abreu and J. R. Rico-Juan, "A new iterative algorithm for computing a quality approximate median of strings based on edit operations," *Pattern Recognit. Lett.*, vol. 36, pp. 74–80, Jan. 2014.

[2] H. Bunke, X. Jiang, K. Abegglen, and A. Kandel, "On the weighted mean of a pair of strings," *Pattern Anal. Appl.*, vol. 5, no. 1, pp. 23–30, May 2002.

[3] R. A. M. Cardenas, "A learning model for multiple-prototype classification of strings," in *Proc. 17th Int. Conf. Pattern Recognit. (ICPR)*, vol. 4, 2004, pp. 420–423.

[4] F. Casacuberta and M. D. Antonio, "A greedy algorithm for computing approximate median strings," in *Proc. VII Sim. Nac. de Reconocimiento de Formas y Análisis de Imágenes*, 1997, pp. 193–198.
[5] I. Fischer and A. Zell, "String averages and self-organizing maps for strings," in *Proc. ICSC Neural Comput.*, 2000, pp. 208–215.
[6] H. Freeman, "Computer processing of line-drawing images," *ACM Comput. Surv.*, vol. 6, no. 1, pp. 57–96, 1974.
[7] S. García-Díez, F. Fouss, M. Shimbo, and M. Saerens, "A sum-over-paths extension of edit distances accounting for all sequence alignments," *Pattern Recognit.*, vol. 44, no. 6, pp. 1172–1182, Jun. 2011.
[8] A. K. Jain and D. Zongker, "Representation and recognition of handwritten digits using deformable templates," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 19, no. 12, pp. 1386–1390, Dec. 1997.
[9] X. Jiang, H. Bunke, and J. Csirik, "Median strings: A review," *Data Mining Time Ser. Databases*, pp. 173–192, 2004, doi: 10.1142/9789812565402_0008.
[10] T. Kohonen, "Median strings," *Pattern Recognit. Lett.*, vol. 3, no. 5, pp. 309–313, 1985.
[11] J. B. Kruskal, "An overview of sequence comparison: Time warps, string edits, and macromolecules," *SIAM Rev.*, vol. 25, no. 2, pp. 201–237, Apr. 1983.
[12] F. Kruzslicz, "Improved greedy algorithm for computing approximate median strings," *Acta Cybern.*, vol. 14, no. 2, pp. 331–339, 1999.
[13] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet Phys. Doklady*, vol. 10, no. 8, pp. 707–710, 1966.
[14] C. D. Martínez-Hinarejos, A. Juan, F. Casacuberta, and R. Mollineda, "Reducing the computational cost of computing approximated median strings," in *Proc. Joint IAPR Int. Workshops SSPR and SPR*, 2002, pp. 47–55.
[15] C. D. Martínez-Hinarejos, A. Juan, and F. Casacuberta, "Median strings for *k*-nearest neighbour classification," *Pattern Recognit. Lett.*, vol. 24, nos. 1–3, pp. 173–181, Jan. 2003.
[16] J. R. Rico-Juan and J. M. Iñesta, "New rank methods for reducing the size of the training set using the nearest neighbor rule," *Pattern Recognit. Lett.*, vol. 33, no. 5, pp. 654–660, Apr. 2012.
[17] F. Nicolas and E. Rivals, "Hardness results for the center and median string problems under the weighted and unweighted edit distances," *J. Discrete Algorithms*, vol. 3, nos. 2–4, pp. 390–415, Jun. 2005.
[18] J. G. Henikoff and S. Henikoff, "Blocks database and its applications," in *Methods in Enzymology*, vol. 266. Amsterdam, The Netherlands: Elsevier, 1996, pp. 88–105.
[19] E. Chavez, S. Dobrev, E. Kranakis, J. Opatrny, L. Stacho, H. Tejeda, and J. Urrutia, "Half-space proximal: A new local test for extracting a bounded dilation spanner of a unit disk graph," in *Proc. Int. Conf. Princ. Distrib. Syst.*, vol. 1, 2005, pp. 235–245.
[20] X. Jiang, J. Wentker, and M. Ferrer, "Generalized median string computation by means of string embedding in vector spaces," *Pattern Recognit. Lett.*, vol. 33, no. 7, pp. 842–852, May 2012.
[21] M. Hayashida and H. Koyano, "Finding median and center strings for a probability distribution on a set of strings under Levenshtein distance based on integer linear programming," in *Proc. Int. Joint Conf. Biomed. Eng. Syst. Technol.*, 2016, pp. 108–121.
[22] M. Hayashida and H. Koyano, "Integer linear programming approach to median and center strings for a probability distribution on a set of strings," in *Proc. 9th Int. Joint Conf. Biomed. Eng. Syst. Technol.* Rome, Italy: SCITEPRESS, 2016, pp. 35–41.
[23] X. Jiang and H. Bunke, "Optimal lower bound for generalized median problems in metric space," in *Structural, Syntactic, and Statistical Pattern Recognition*, vol. 2396. Portugal: Sci. Technol. Publications, 2002, pp. 143–151.
[24] C. Olivares-Rodríguez and J. Oncina, "A stochastic approach to median string computation," in *Proc. SSPR SPR*, in Lecture Notes on Computer Science, vol. 5342, 2008, pp. 431–440.
[25] E. S. Ristad and P. N. Yianilos, "Learning string-edit distance," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 20, no. 5, pp. 522–532, May 1998.
[26] L. Franek and X. Jiang, "Evolutionary weighted mean based framework for generalized median computation with application to strings," in *Structural, Syntactic, and Statistical Pattern Recognition*. USA: Springer, 2012, pp. 70–78.
[27] G. Sánchez, J. Lladós, and K. Tombre, "A mean string algorithm to compute the average among a set of 2D shapes," *Pattern Recognit. Lett.*, vol. 23, nos. 1–3, pp. 203–213, Jan. 2002.
[28] P. Mirabal, J. Abreu, and D. Seco, "Assessing the best edit in perturbation-based iterative refinement algorithms to compute the median string," *Pattern Recognit. Lett.*, vol. 120, pp. 104–111, Apr. 2019.
[29] R. Corral-Corral, E. Chavez, and G. Del Rio, "Machine learnable fold space representation based on residue cluster classes," *Comput. Biol. Chem.*, vol. 59, pp. 1–7, Dec. 2015.
[30] S. Budalakoti, A. N. Srivastava, and M. E. Otey, "Anomaly detection and diagnosis algorithms for discrete symbol sequences with applications to airline safety," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 39, no. 1, pp. 101–113, Jan. 2009.
[31] X. Jiang, L. Schiffmann, and H. Bunke, "Computation of median shapes," in *Proc. 4th Asian Conf. Comput. Vis.*, Taipei, Taiwan, 2000, pp. 300–305.
[32] A. Lourenço and A. Fred, "Ensemble methods in the clustering of string patterns," in *Proc. 7th IEEE Workshops Appl. Comput. Vis. (WACV/MOTIONS)*, Jan. 2005, pp. 143–148.
[33] T. Kohonen and P. Somervuo, "Self-organizing maps of symbol strings," *Neurocomputing*, vol. 21, nos. 1–3, pp. 19–30, Nov. 1998.
[34] R. Popovici and R. Andonie, "Sensor signal clustering with self-organizing maps," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2015, pp. 1–8.
[35] B. G. Chern, I. Ochoa, A. Manolakos, A. No, K. Venkat, and T. Weissman, "Reference based genome compression," in *Proc. IEEE Inf. Theory Workshop (ITW)*, Sep. 2012, pp. 427–431.
[36] P. Bille, "A survey on tree edit distance and related problems," *Theor. Comput. Sci.*, vol. 337, nos. 1–3, pp. 217–239, Jun. 2005.
[37] P. Bille, P. H. Cording, I. L. Gørtz, F. R. Skjoldjensen, H. W. Vildhøj, and S. Vind, "Dynamic relative compression," *CoRR*, 2015. [Online]. Available: https://arxiv.org/abs/1504.07851
[38] S. Kuruppu, S. J. Puglisi, and J. Zobel, "Reference sequence construction for relative compression of genomes," in *Proc. Int. Symp. String Process. Inf. Retr.*, 2011, pp. 420–425.
[39] S. Wandelt, M. Bux, and U. Leser, "Trends in genome compression," *Current Bioinf.*, vol. 9, no. 3, pp. 315–326, May 2014.
[40] R. Giancarlo, S. E. Rombo, and F. Utro, "Compressive biological sequence analysis and archival in the era of high-throughput sequencing technologies," *Briefings Bioinf.*, vol. 15, no. 3, pp. 390–406, May 2014.
[41] L. P. Dinu and R. T. Ionescu, "Clustering based on median and closest string via rank distance with applications on DNA," *Neural Comput. Appl.*, vol. 24, no. 1, pp. 77–84, Jan. 2014.
[42] P. Mirabal, J. Abreu, and O. Pedreira, "Pivot selection for median string problem," in *Proc. 38th Int. Conf. Chilean Comput. Sci. Soc. (SCCC)*, Nov. 2019, pp. 1–5.
[43] P. Mirabal, J. Abreu, D. Seco, O. Pedreira, and E. Chavez, "New initialization for algorithms to solve median string problem," in *Proc. 39th Int. Conf. Chilean Comput. Sci. Soc. (SCCC)*, Nov. 2020, pp. 1–7.
[44] J. Abreu and P. Mirabal, "Multi-edition approach for median string problem," in *Proc. 39th Int. Conf. Chilean Comput. Sci. Soc. (SCCC)*, Nov. 2020, pp. 1–6.
[45] S. Mirabal and P. Aniel, "Algoritmos para el cálculo de la cadena media," M.S. thesis, Facultad de Ingeniería, Univ. Concepción, Concepción, Chile, 2019.
[46] M. S. Kaysar and M. I. Khan, "A modified median string algorithm for gene regulatory motif classification," *Symmetry*, vol. 12, no. 8, p. 1363, Aug. 2020.
[47] D. Chakraborty, D. Das, and R. Krauthgamer, "Approximating the median under the Ulam metric," in *Proc. ACM-SIAM Symp. Discrete Algorithms (SODA)*. Philadelphia, PA, USA: SIAM, 2021, pp. 761–775.
[48] F. J. Castellanos, J. J. Valero-Mas, and J. Calvo-Zaragoza, "Prototype generation in the string space via approximate median for data reduction in nearest neighbor classification," *Soft Comput.*, vol. 25, pp. 15403–15415, Sep. 2021.
[49] R. A. Wagner and M. J. Fischer, "The string-to-string correction problem," *J. ACM*, vol. 21, no. 1, pp. 168–173, 1974.
[50] J. Abreu and J. R. Rico-Juan, "An improved fast edit approach for two-string approximated mean computation applied to OCR," *Pattern Recognit. Lett.*, vol. 34, no. 5, pp. 496–504, Apr. 2013.

PEDRO MIRABAL received the Ph.D. degree in computer science from the University of Concepción, Chile, in 2019. He is currently a Professor with the Department of Informatics Engineering, Faculty of Engineering, Universidad Católica de Temuco, Chile. His research interests include NLP, data structures, and algorithms.

JOSÉ ABREU is currently a Researcher with the Institute for Computing Research, University of Alicante. He has been a member of the Cuban Chapter of the International Association of Pattern Recognition, and a full-time Professor with the University of Matanzas, and the Catholic University of the Most Holy Conception. His research interests include data-driven solutions in natural language processing and instance selection and prototype construction algorithms.

DIEGO SECO received the Ph.D. degree in computer science from the University of A Coruña, Spain, in 2009. He is currently an Associate Professor with the Department of Computer Science, University of Concepción, Chile. His research interests include geographic information retrieval, geographic information systems, and compressed data structures and algorithms for textual and geographic data.

ÓSCAR PEDREIRA received the M.Sc. and Ph.D. degrees in computer science from the University of A Coruña, Spain. He has been an Associate Professor with the University of A Coruña, since 2008. He is currently a Researcher of the Database Laboratory. He has coauthored many articles published in journals and conferences relevant for the research areas mentioned. His research interests include databases (algorithms for similarity search, data structures and algorithms for graph databases, geographic information systems), and in software engineering (process improvement, testing, MDE, and SPL). He has continuously participated in research projects and technology and knowledge transfer projects with different companies.

EDGAR CHÁVEZ received the Ph.D. degree in computer science from the Centro de Investigacion en Matematicas (CIMAT), in 1999. He is currently a Full Professor with the Centro de Investigacion Cientifica y de Educacion Superior de Ensenada (CICESE), Ensenada, Mexico. His research interests include multimedia information retrieval, similarity search, indexing, and clustering algorithms.