Aggregation of results in Crowdsourcing by means of an Evolutionary Algorithm that calculates the Approximate Median String

LUIS GOMEZ
Electronic Engineering, Faculty of Engineering,
Central Unit of the Valley of Cauca.
Tuluá, COLOMBIA
Email: lgomez@uceva.edu.co

ANDRES REY
Electronic Engineering, Faculty of Engineering,
Central Unit of the Valley of Cauca.
Tuluá, COLOMBIA
Email: arey@uceva.edu.co

ANGEL LOZADA
Electronic Engineering, Faculty of Engineering,
Central Unit of the Valley of Cauca.
Tuluá, COLOMBIA
Email: alozada@uceva.edu.co

Abstract: - In Crowdsourcing, the aggregation of results consists of introducing redundancy by asking several workers to perform the same task, and then adding the answers given by the workers seeking to obtain results that are more reliable. The aggregation of results is a very promising approach that can be easily implemented when having numerical entries, but it is quite complex when the entries correspond to strings. This article details how the concept of Median String was applied to perform the aggregation of Crowdsourcing strings, developing an evolutionary algorithm for this purpose. The results obtained show that our algorithm can calculate the Median String, which is a NP-Hard problem, of correct way but when applied to correct errors in Crowdsourcing calculating the correct answer (hidden truth) depends on the quality of the inputs that must present low dissimilarity

Key-Words: - aggregation of results, evolutionary algorithms, median String, error correction, crowdsourcing.

1 Introduction

The concept of Crowdsourcing is widely disseminated nowadays, which refers, according to its creator Jeff Howe, to the "act of a company or institution taking a function once performed by employees and outsourcing it to an undefined (and generally large) network of people in the form of an open call" [1]. This definition proposed by Howe, although it is quite concrete, provides elements to suggest that Crowdsourcing can be seen as a new business model that is considered particularly useful in tasks that require a large number of points of view or different solutions to problems [2].

Crowdsourcing is generally carried out through the Web and over the years has proven to be an effective and scalable approach to solve different problems in various fields [2], even to be used to solve problems that are computationally expensive or impossible to solve for machines, but which are rather simple for human beings; hence, sometimes the tasks proposed to the multitude are called Human Intelligence Task (HITs).

Among the technical challenges faced by crowdsourcing, one of the most important is to control the quality of the results obtained, because given the open nature of crowdsourcing, the data collected through a process of this type are potentially noisy [3]. To overcome this challenge, several approaches or mechanisms have been proposed, such as: (i) aggregation of results, (ii) gold data, (iii) multilevel reviews, (iv) expert review, (v) pre-selection and (vii) reputation.

With regard to this study, the aggregation of results approach is of interest, which consists of introducing redundancy by asking several workers to perform the same task, and then adding the

answers or results delivered by the workers seeking a more reliable result. The aggregation of results is a very promising approach that can be easily implemented when having numerical entries, but it is quite complex when the entries correspond to strings [4].

Bearing in mind that in the last decades several problems derived from the treatment of strings have been the subject of research in fields such as artificial vision, speech recognition, spell checking, detection of plagiarism, extensive searches in databases, Internet searches, pattern recognition, and bioinformatics, [5], [6]. In this study, it is proposed that by performing an adequate processing of the strings, it may be possible to strengthen the Crowdsourcing mechanisms of aggregation of results, which seek to find the hidden truth in the set of responses given by the multitude for the proposed task. What is specifically proposed is to perform the aggregation of strings in Crowdsourcing applying the concept of Median String to perform correction of errors presented in the input strings. The concept of Median String in mention is equivalent to the concept of average vector, but having in this case a set of data conformed by character strings what causes that its calculation is not simple and takes to face a NP-Hard problem [7].

This article details how the concept of Median String was applied to aggregate a set of strings generated in Crowdsourcing tasks, developing an evolutionary algorithm that provides good results with a reasonable computational cost. The structure of the document is as follows: section 2, deals with the fundamentals; in section 3, the proposed evolutionary algorithm for calculating the median string of Crowdsourcing entries is presented; Section 4 shows the experimental results; Section 5 contains the discussion of results; and section 6, presents the conclusions.

2 Background

In this section, the main concept for the work that corresponds to the Median String is described and the problem that is addressed when trying to find just the median string of a set of given strings. In addition, a brief description of the evolutionary algorithms proposed as a heuristic for the solution of the mentioned problem and some related works are presented

2.1 Median String

The initial definition of median string was presented by Kohonen [8] who defines the concept for the first time, as follows: given a set of strings, the median string is defined as the string that minimizes ther sum of distances to the strings of the set [8]. The above definition is represented by the following mathematical expression (1):

$$m_{S=} \ min_{s \in \sum *} \sum_{i=1}^{n} d\,(s, s^i) \quad (1)$$

In the above equation mS denotes the median string to find given a set of strings S1= {S1, S2, … , Sn} formed by elements of an alphabet Σ, where Σ * indicates that the strings have finite length and d is the measure of dissimilarity or distance defined according to the application.

The concept of median string is equivalent to the average vector but in this case, the data set is made up of strings of characters, so its calculation is not simple. To find the median string of a set of given strings, it is necessary to fulfill the condition that the sum of the distances of this with all the given strings must be minimal.

The distances can be measured in several ways but the best known is with the Levenshtein metric, which allows to find the distance between two strings, that is, it is a measure that represents how different the strings are. Table 1 shows examples where the Levestein distance is calculated, which basically corresponds to the minimum number of operations required to transform one character string into another; it is understood by operation, either an insertion, elimination or the substitution of a character.

Table 1. Example of distance between strings

String 1	String 2	Distance
casa	casa	0
casa	ca**z**a	1
c**es**a	c**az**a	2
casa	**perro**	5
abcd	**dcba**	4

When trying to find the Median String, a search space must be generated and explored, consisting of all the strings that can be created with all possible combinations of the elements of the alphabet Σ. The problem then is to look in this space until finding a string whose sum of distances with respect to the entry set of string is the minimum.

In spite of the definition of Median String is apparently simple, it was shown that the calculation of the exact median string is a NP-Hard problem [7], which in practice leads to the need for approaches that calculate an approximate solution through algorithms that have a computational cost polynomial and not exponential as when the exact median string is found.

To address the solution of problems such as finding the median string, evolutionary algorithms can be used, which is precisely what is discussed next.

2.1 Evolutionary algorithms

Historically, evolutionary algorithms are considered a branch of artificial intelligence, since they were used in the 1960s to add intelligence to finite-state machines, but today they are used to solve a wide variety of problems that are computationally difficult to solve. That is to say, those that do not have a solution in polynomial time, such as: the problem of the traveling agent, the problem of the backpack, the problem of finding a Hamiltonian cycle, etc. All these problems for relatively small entries, the response time executed on the best computer in the world would take the time when the universe was created until today. That is why it is necessary to attack these problems with alternative strategies where the response time is reasonable.

There are several different ways or approaches to implement evolutionary algorithms. Among the best known are the genetic algorithms created by John Henry Holland in the 70s, which gave rise to evolutionary computation; these algorithms are characterized because the chromosome is encoded in zeros and ones, and all the functions are applied to expressions of zeros and ones, besides having a function of decoding.

In this work, another existing approach was used, genetic programming, which is very similar to the genetic algorithm but the chromosome is not coded, the phenotypic part is the same as the genotypic one and the operators are defined according to the case and application. The general operation of an evolutionary algorithm consists of the following steps:

Initialization: a set of chromosomes are created with random values, where each chromosome is a possible solution to the problem that is to be solved or optimized. The set of chromosomes generated in the initialization are stored in a memory space called search space.

Evaluation: in this phase, a fitness value or a classification is given to the chromosome in order to know how close the solving the problem is.

Selection for reproduction: a subset of the population of existing chromosomes in the search space is selected to apply subsequently a series of functions generally known as reproduction (crossing or mutation) or survival functions. There are several strategies to select the chromosomes after establishing how many or in what percentage they will pass for reproduction, for example: choosing the ten best chromosomes, that is to say the ten ones with the best fitness function; randomly choosing 10 chromosomes from the search space; make a sampling by lottery, where the chromosomes with the best fitness function are more likely to be chosen; using the roulette method, a random value is generated and from this value are formed (through a simple recurrence relation) other missing random values; and by tournament, selecting three chromosomes randomly and from these ones, the one with the best fitness value is chosen and the procedure is repeated until the number of fixed chromosomes is chosen.

Recombination: different reproductive functions are applied to the selected chromosomes with the purpose of generating new descendants. Among the functions to be applied are those of crossing and mutation. Not all descendant chromosomes are going to be good, but the recombination process allows some better chromosomes to be generated. This procedure is done until reaching a point where the new chromosomes do not improve those found in the search space, a signal that serves to stop the evolutionary algorithm.

Replacement: there are many ways to replace old chromosomes in the search space with the new descendant chromosomes, among the most used are: (a) the ten chromosomes generated will replace the ten worst chromosomes in the search space; (b) if many chromosomes are generated then the entire search space is replaced with all the descendant chromosomes; (c) only the chromosomes that overcome or gain the worst chromosomes are replaced (many of the chromosomes generated are worse than the worst chromosomes in the search space.

When an evolutionary algorithm is implemented, the elements shown in Figure 1 must be generated. These elements appear during the execution of the evolutionary algorithm as described below.

The search space is created in the initialization containing the chromosomes to which the fitness function in the evaluation is applied. Subsequently, during the selection, chromosomes that are selected

to proceed with reproduction are copied into the mating pool where new chromosomes are created, which during the replacement can take the place of the worst chromosomes in the search space by completing the process that can be repeated until the new chromosomes do not improve those in the search space or a defined number of iterations is completed. As in the search space, there are n possible solutions; all the algorithms that are implemented following this scheme must be of low computational cost.

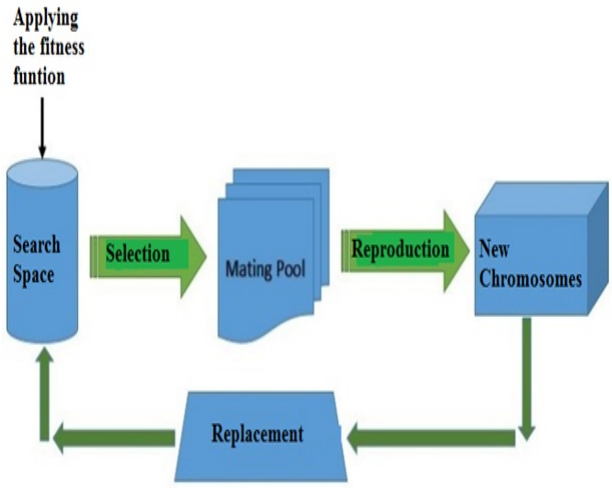

Fig. 1. Main elements to be implemented in evolutionary computation

3 Evolutionary algorithm developed to calculate the median string

Evolutionary algorithms using the genetic programming approach carried out the solution to the problem of finding the median string.

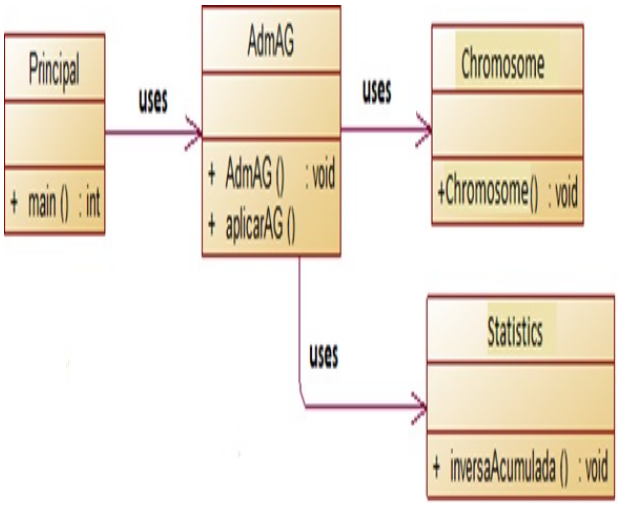

Fig. 2. Class diagram of the implemented solution

The evolutionary algorithm developed was implemented in the C ++ programming language, which is a language recognized for its speed of execution.

To start the description of the evolutionary algorithm developed, Figure 2 is provided, which corresponds to the class diagram of the implemented solution.

As it is observed in the previous figure there are four classes: (1) Main class, in charge of starting the execution of the evolutionary algorithm; (2) a class called AdmAG that implements the steps defined in evolutionary programming such as initialization, evaluation, selection, reproduction and replacement; (3) Chromosome class, used to model the chromosomes; (4) Statistics class, which contains methods to support various statistical operations that are commonly used in evolutionary algorithms that use the sampling technique for the selection of the best chromosomes.

The four previous classes allow to find the median string and the process starts when the main class creates an instance of the evolutionary algorithm class (AdmAG) passing as parameters the set of strings to which the median string is going to be calculated; the input strings correspond to raw data that is loaded by the main class from an input file.

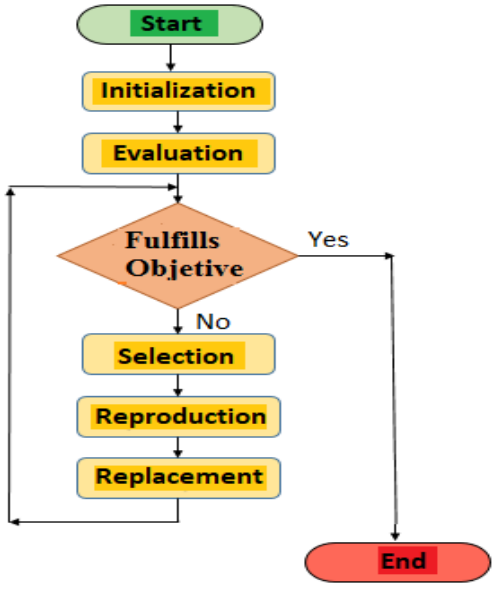

Fig. 3. Steps followed in the evolutionary computation

At the time of creating the evolutionary algorithm class instance the constructor method is responsible for creating the table of symbols to be used, including only the characters that appear in the input strings passed as an argument, this in order to optimize the search space. Once this is done, the

AG method is invoked, which is responsible for executing the steps defined in the evolutionary computation shown in Figure 3.

With the previous steps, evolutionary programming allows to create and explore a large search space made up of chromosomes that contain strings created from the input strings. A description of what happens in each step is presented below:

Initialization: Consists of filling the search space with 100 chromosomes. The search space is implemented as a vector of chromosomes formed basically with random strings created according to the table of symbols. Each chromosome represents a possible solution to the problem of the median string.

Evaluation: Consists of ordering the chromosomes of the search space from least to greatest according to the evaluation value obtained when comparing the current string (that conforms to the chromosome) with respect to the received input strings. The evaluation value represents how close a chromosome is to the input strings. The evaluation values are obtained for each chromosome at the time of ordering and are stored as part of its structure.

Selection: With the evaluation values, it is created a vector that passes as a parameter to create an instance of the Statistics class, which is responsible for choosing the statistically most suitable chromosomes according to their evaluation value. It is important to highlight that before creating this instance, negative numbers or zeros must be avoided, so before using the statistics class, the minimum of the evaluation function is calculated and if this value is not positive it is necessary a displacement.

That is to say, a kind of prenormalization is done because the statistics class makes the true normalization to apply the cumulative inverse function (main function of the class) that is responsible for randomly choosing the most suitable chromosomes that are copied to a vector called "Mating Pool". The chromosomes with higher evaluation value are more likely to be chosen, but all have some probability including the least fit, because for their selection in the cumulative inverse function a strategy called sampling draw is used; the number of chromosomes that pass to the Mating Pool corresponds to 10% of the size of the search space.

Reproduction: Once in the Mating Pool, two chromosomes that are located next to each other are already chosen and the percentage is decided if they are going to mutate or if they are going to cross each other. If it mutates, it is decided randomly, which of the four defined mutation functions is going to be applied and if it is going to cross, it is decided randomly which of the two crossing functions will be chosen. The new chromosomes that result from the mutation and/or crossing are taken to a new vector of chromosomes and when all the reproductions are finished and the vector is complete, it is ordered according to the evaluation function; the reproduction ends when all the chromosomes of the Mating Pool are taken.

Replacement: To replace chromosomes from the search space with the new chromosomes that result from reproduction, the criterion is used: if it is better than the worst of those found in the search space, then it enters the search space. This implies that to enter one of the new chromosomes into the search space, it is required that its evaluation function be better than that of the worst chromosome located in the search space; new chromosomes that do not enter the search space will be discarded because they are not "better" than those already existing are. After entering the new chromosomes in the search space, it is ordered and the process starts again until it is observed that new chromosomes are not entering the search space, for this reason the algorithm is stopped and the highest chromosome of the search space is returned which will be the one that contains the median string intended to find.

To finish the description of the evolutionary algorithm implemented, three things need to be detailed first, how the representation of individuals was handled, secondly, the function of evaluation or defined fitness (one of the most important elements) and third, the crossing and mutation operations designed.

Regarding the representation of individuals, the chromosome is made up of a single gene that corresponds to a string of characters. Each chromosome represents a possible solution to the problem of the median string and when creating the strings that comprise it, the following considerations are taken into account: the sizes of these strings range from the minimum size of the input strings to the largest size of the input strings; all characters are chosen from the symbol table that is created only by using characters that appear in the input strings. The above restrictions correspond to heuristics that are totally logical since the median string can not be smaller than the minimum one, nor even larger than the maximum one.

On the other hand, the evaluation function defined to measure the fitness of the chromosomes in the evolutionary algorithm is presented and described below:

$$\textit{Evaluation Value} = \textit{Similarity by Column} - \textit{Total Dissimilarity} \quad (2)$$

Similarity by column. Corresponds to the evaluation that is made of the similarity of the characters of the strings evaluated in each column or position in the following way: if the characters of the column are equal, a value of 1 is assigned, if they are different -1 and if it is a blank space-2. The final value of the evaluation of the similarity by column corresponds to the sum of the results of the evaluation of each column when comparing the string that conforms to the chromosome against all the input strings. A high evaluation value represents that the strings are similar, but it may obtain negative values that represent that the strings are not similar.

Total dissimilarity. This measure represents how different the evaluated strings are. To measure the dissimilarity of the strings, the Levenshtein's method was used to find a value called editing distance, which represents how different the strings are. When the editing distance is zero, it means that the strings are equal but as it increases, it represents a greater dissimilarity. The measure of total dissimilarity is always positive.

Finally, with regard to crossing operations two forms were designed, crossing 1 that creates a new chromosome by crossing two chromosomes in a single randomly chosen point and crossing 2, which creates a new chromosome crossing two chromosomes in two points chosen in a random way. Moreover, for the mutation, four operations were designed:

To move. Where a new chromosome is created by moving all the characters of the string one position to the left or to the right. The character that comes out of the chromosome is introduced on the opposite side, leaving the string of the same size.

To remove. Creates a new chromosome by removing a character from a random position. To the Levestein's distance, the amount of removed elements is added, to assure that the gain in distance, not only because the string diminishes in length, but also because an element that was doing that the distance grew was removed. To remove can be taken as a mutation process.

To insert. For creating a new chromosome by inserting a character from the ones in the symbol table. The symbol is chosen randomly from the symbol table and the place where it is to be inserted is chosen randomly from the string, this being a larger unit. To insert can be taken as a mutation process.

To modify. Creates a new chromosome by modifying one character for another. The position to be modified is chosen randomly and the character to be inserted is chosen randomly from the symbol table.

When applying any of the above functions, the symbol table that is created from the characters that make up the input strings is used.

4 Experimental results

In order to evaluate the developed evolutionary algorithm that allows finding the median string, two different types of tests were carried out: first, testing the evolutionary algorithm to find the median string of several given input strings generated in Crowdsourcing tasks. Secondly, evaluating the algorithm evolutionary versus a performed implementation of an algorithm that calculates the exact median string to compare response times.

4.1 Evaluation of the evolutionary algorithm finding the median string

The first group of tests (Test No.1, Test No.2 and Test No.3) were intended to evaluate the functioning of the evolutionary algorithm by finding the median string for several different input strings. Numerous tests of this type were carried out varying the length and number of entries, and the number of symbols that made up the strings.

Below are the results of three of the tests performed where they were used as input strings, responses generated by Crowdsourcing tasks of transcription of information when it was a question of collecting a data corresponding to the word "Fotocopiado". For each of the tests ten executions were made using the same machine (Lenovo G40 computer with Windows 7, 2.16 GHz and 2 GB of memory) and without making variations to the algorithm or input strings. The inputs provided to the algorithm shown in Table 2 and the outputs delivered by algorithm, the times required to perform the computation, the editing distance of the calculated median string with respect to the input strings were recorded for each execution, these data are shown in Tables 3, 4 and 5.

Table 2. Input strings used in the tests of the calculation of the median string

Input string	Test No.1	Test No.2	Test No.3
String No.1	fotocopiado	fotocoPiado	fotoCoPiado
String No.2	Fotocopiad	Fotoopiad	Ftoopiad
String No.3	Fotocopado	Foocopado	oocopado
String No.4	Foticopiado	Foticopiadu	Foyicopiadu

Table 3. Strings calculated by the algorithm

Execution	Test No.1		
	Median String	**Editing Distance**	**Time (Sec)**
No. 1	Fotocopiado	4	1,17
No. 2	Fotocopiado	4	1,07
No. 3	Fotocopiado	4	1,12
No. 4	Fotocopiado	4	1,23
No. 5	Fotocopiado	4	1,96
No. 6	Fotocopiado	4	1,11
No. 7	Fotocopiado	4	1,16
No. 8	Fotocopiado	4	1,11
No. 9	Fotocopiado	4	1,31
No. 10	Fotocopiado	4	2,00

Table 4. Strings calculated by the algorithm

Execution	Test No.2		
	Median String	**Editing Distance**	**Time (Sec)**
No. 1	Fotocopiado	8	14,13
No. 2	Fotocopiado	8	14,13
No. 3	Fotocopiado	8	13,10
No. 4	Fotocopiado	8	14,18
No. 5	Fotocopiado	8	14,14
No. 6	Fotocopiado	8	14,12
No. 7	Fotocopiado	8	14,16
No. 8	Fotocopiado	8	14,18
No. 9	Fotocopiado	8	11.66
No. 10	Fotocopiado	8	14,13

Table 5. Strings calculated by the algorithm

Execution	Test No.3		
	Median String	**Editing Distance**	**Time (Sec)**
No. 1	Fooopado	15	15,89
No. 2	Fooopiada	15	4,32
No. 3	Focopiada	15	4,32
No. 4	Fotocopiado	12	4,01
No. 5	Fotocopiado	12	2,91
No. 6	Fotopopiado	14	10,74

4.2 Evaluation of evolutionary algorithm versus naive algorithm

The second part of the tests was aimed at evaluating the response time of the evolutionary algorithm; it was specifically intended to verify if the evolutionary algorithm could continue finding the median string in a reasonable time, even though the input strings were increasing their length. In order to perform this test, another so-called naive algorithm had to be implemented that was designed to calculate the exact median string (the implementation of this algorithm is included as a method of the AdmAG class) and served as a reference point to compare the performance of the evolutionary algorithm in regard to the response time. It was called a naive algorithm precisely because it calculates the exact median string, but it can only be used when the input strings have a small length. For example to calculate the median string corresponding to the following input strings: minima, Minim, Miima and Manima. The naive algorithm takes 0.764 seconds, which is a short time in the case of a complex calculation, but as the length of the strings increases so does time, for the strings: estatuto, Estatut, Estatoto and Esatuto. The naive algorithm takes 397 seconds, equivalent to 6 minutes, which is a longer time but still continues to be acceptable.

However, if the length of the string increases and the symbols that make it up, it is possible to have times that are not acceptable. For example having to calculate the exact median string for entries as "fotocopiadora" would take approximately 29 days. This is why this algorithm that calculates the exact median string was called naive, since it is not viable to use it for the large amount of time it takes to perform the calculations.

If the input string is "fotocopiadora", then a symbol table (or alphabet) consisting of: "f", "o", "t", "c", "p", "i", "a", "d", "r" (9 different characters). The naive algorithm has to process 913 possible combinations that is of the order of 1012 operations, more exactly 2541865828329, this figure when dividing it by the number of operations that a computer makes per second, gives the number of seconds that is required to process all combinations. For example, if you have a computer that does one million operations per second, then to calculate this input string would be taken 2541865828329 / 106 = 2541866 seconds= 29 days.

The test No.4 consisted of comparing the response times of both algorithms, the evolutionary and the so-called naive, using the same input strings that changed in length starting with 3 characters that were incremented until reaching 7 characters of an execution to another, but always keeping the same number of symbols that make them up. The input strings used in the test were taken from a corpus of tests called "abecede" that was proposed and used in

[9] where it was extensively inquired about different ways of finding the median string. Both algorithms were evaluated when processing input strings formed by combinations of the following strings: dac, bdac, caaac, ccaadc, aadaaac.

Table 4 shows the response times obtained in test No.4, meanwhile in Figure 4 the behavior of each algorithm is plotted.

Table 6. Response times, evolutionary versus naive algorithm

Execution	Response Time, Naive Algorithm	Response Time, Evolutionary Algorithm
Execution No.1	0,015	0,02
Execution No.2	0,015	0,13
Execution No.3	0,047	0,161
Execution No.4	0,203	0,198
Execution No.5	0,978	0,25
Execution No.6	4,623	0,333
Execution No.7	21,804	0,374

To investigate the behavior of the evolutionary algorithm, several regression models were generated from the available data. After testing with some different types of regression, two models were identified that fit the response delivered by the evolutionary algorithm, as shown in Figure 5.

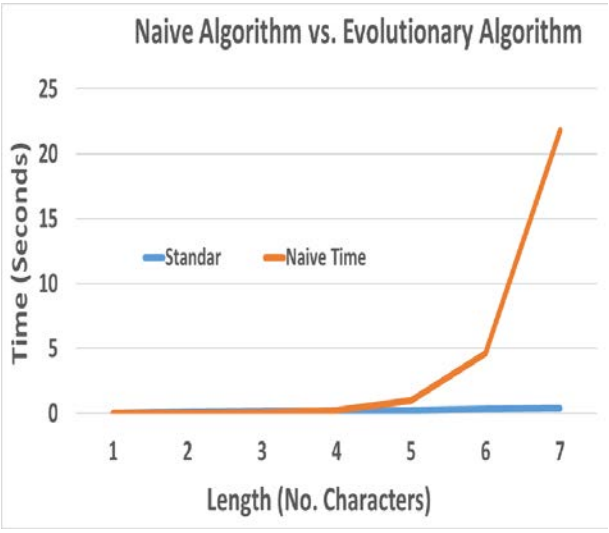

Fig. 4. Response time, Naive Algorithm versus Evolutionary Algorithm

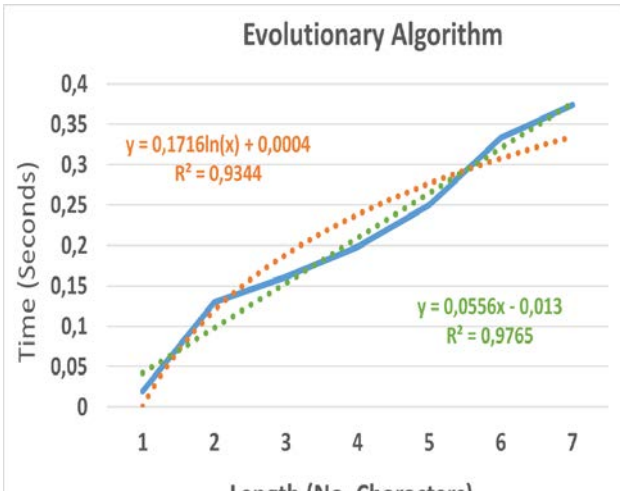

Fig. 5. Evolutionary Algorithm Time Trend

The models shown in Figure 5 were obtained from the data in Table 6. The first model that is drawn in orange corresponds to a logarithmic model and in green, the second model that corresponds to a linear model. The linear model is the one that best fits the response delivered by the evolutionary algorithm (which is drawn in blue) according to the coefficient of determination R2 that has a value of 0.9795.

5 Discussion of results

The results presented of the tests carried out on the developed evolutionary algorithm show that the algorithm can calculate the median string correctly and in a low or reasonable time that increases as the input strings have greater dissimilarity. The response time of the algorithm also increases when the input strings present a greater extension; this is previously known as combinatorial problems.

For the test No.1, the input strings presented the lowest level of dissimilarity that is reflected in the editing distance calculated by the evolutionary algorithm, 4 in all the executions performed in this test. In tests No.3 and No.4, the dissimilarity of the input string was increased, a situation that is identified when performing a visual inspection of them, or when observing the editing distance indicated by the algorithm that for test No. 2 was 8 in all the executions and in the No.3 test it varied between performances from 12 to 15. This last result allows indicating that for a set of input strings that present greater dissimilarity, several strings that have the same editing distance can be found or calculated concluding that in these cases the median string is not unique, unlike what is observed in the

first two tests where a single median string is obtained for a low dissimilarity in the input strings.

It should also be noted that in tests No.1 and No.2, the obtained results show that the median string found in all the executions was equal to the exact value or "hidden truth" that was wanted to be found that corresponded to the "Fotocopiado" string. In these cases, it can be indicated that the action of the evolutionary algorithm can be assimilated as an action of correction of the errors presented by the input strings. In contrast, in test No.3 the median string delivered by the evolutionary algorithm did not coincide in most executions with the exact value that was expected. These results indicate that the evolutionary algorithm can perform correction of errors present in the input strings when they present low dissimilarity, but otherwise the error correction action of the algorithm is not as effective.

On the other hand, the results of the test No.4, which correspond to the behavior of the evolutionary algorithm in its response time, allow to suggest that for input strings whose length does not exceed 4 characters, the response time is low in both Algorithms being faster the naive algorithm. Nevertheless, as the input strings increase in length by exceeding four characters, the evolutionary algorithm is much faster. The response time of the naive algorithm grows exponentially, while the response time of the evolutionary algorithm does so linearly with a very low slope. Hence, it is suggested that the evolutionary algorithm can find the median string in a reasonable time that does not grow exponentially.

Among the constraints identified in the operation of the algorithm, it must be reported that the use of the C ++ language generates a limitation in the size of the input strings which can not be greater than 12 characters, because memory overflow is caused in some related variables with the handling of the combinations that are generated in the search space of the solution. This situation was identified once the implementation of the evolutionary algorithm was available and running tests were performed.

Finally, it should be noted that the application of the concept of Median String to perform the aggregation of strings in Crowdsourcing and its implementation with evolutionary algorithms presents an innovative nature for the developments in this field.

6 Conclusions

In Crowdsourcing, the aggregation of results is one of the most important approaches or mechanisms used to obtain results that are more reliable. The implementation of this mechanism is simple when having numeric entries but it is quite complex when the entries correspond to strings.

In this study, it was showed how to apply the concept of Median String to perform the aggregation of strings in Crowdsourcing, developing an evolutionary algorithm for this purpose.

The obtained findings provide elements to suggest that the aggregation of strings in Crowdsourcing applying the concept of Median String allows strengthening the aggregation mechanisms by providing correction of the errors presented in the input strings. However, it is important to emphasize that the results that can be obtained depend largely on the input strings that must not present large deviations from the exact value to be captured.

The above is a restriction associated to the use of the evolutionary algorithm and not of the algorithm as such that does present a restriction in the size of the input strings that cannot exceed 12 characters. To overcome this restriction, it is proposed as future work to carry out a new implementation of the algorithm using a programming language, different from the C language, and that is not typed as Racket (Scheme) seeking to have fewer limitations on the memory size that is assigned to the variables, then the ability to process longer input strings could be expanded. In the same way, within the work to be followed, it is intended to expand the test cases and compare the performance of the evolutionary algorithm developed against other algorithms used for this same task.

References:

[1] J. Howe, The rise of crowdsourcing, *Wired Mag*., Vol 14, No.6, 2006, pp. 1–4.

[2] D. Brabham, Crowdsourcing as a model for problem solving: leveraging the collective intelligence of online communities for public good. *The University of Utah*, 2010.

[3] A. Quinn & B.Bederson, Human computation: a survey and taxonomy of a growing field, *Proceedings of the SIGCHI conference* on human factors in computing systems, 2011, pp.1403–1412.

[4] L. Duan, S. Oyama, H. Sato, & M. Kurihara, Separate or joint? Estimation of multiple labels From crowdsourced annotations, *Expert Syst*. Appl.,Vol.41, No.13, 2014, 5723–5732.

[5] S. Altschul, W. Gish, W. Miller, E. W. Myers, & D. Lipman, *Basic local alignment search tool*, J. Mol. Biol., Vol.215, No.3, 1990, pp.

403–410.
[6] D. Sankoff & J. Kruskal, Time warps, string edits, and macromolecules: the theory and practice of sequence comparison, *Read*. Addison-Wesley Publ, 1, 1983.
[7] C. de la Higuera & F. Casacuberta, Topology of strings: Median string is NP-complete, *Theor. Comput. Sci*., Vol.230, No.1, 2000, pp.39–48.
[8] T. Kohonen, Median strings, *Pattern Recognit*. Lett., Vol.3, No.5, 1985, pp. 309–313.
[9] C. Martínez-Hinarejos, La cadena media y su aplicación en reconocimiento de formas, *Phd. Thesis*. UPV, 2003.