2009 Ninth International Conference on Intelligent Systems Design and Applications

Optimizing Multiple Sequence Alignment by Improving Mutation Operators of a Genetic Algorithm

Fernando José Mateus da Silva
Dept. of Informatics Engineering
Sch. of Tech. and Mgmt., Polytechnic Institute of Leiria
Leiria, Portugal
fernando.silva@estg.ipleiria.pt

Juan Manuel Sánchez Pérez, Juan Antonio Gómez Pulido, Miguel A. Vega Rodríguez
Dept. Tecnologías Computadores y Comunicaciones
Escuela Politécnica, Universidad de Extremadura
Cáceres, Spain
{sanperez, jangomez, mavega}@unex.es

Abstract— **Searching for the best possible alignment for a set of sequences is not an easy task, mainly because of the size and complexity of the search space involved. Genetic algorithms are predisposed for optimizing general combinatorial problems in large and complex search spaces. We have designed a Genetic Algorithm for this purpose, AlineaGA, which introduced new mutation operators with local search optimization. Now we present the contribution that these new operators bring to this field, comparing them with similar versions present in the literature that do not use local search mechanisms. For this purpose, we have tested different configurations of mutation operators in eight BAliBASE alignments, taking conclusions regarding population evolution and quality of the final results. We conclude that the new operators represent an improvement in this area, and that their combined use with mutation operators that do not use optimization strategies, can help the algorithm to reach quality solutions.**

Keywords - Multiple sequence alignments; genetic algorithms; local search; optimization; bioinformatics

I. INTRODUCTION

DNA, RNA and protein alignment is one of the most common tasks in bioinformatics.

An alignment is a mutual placement of two or more sequences which exhibits where the sequences are similar and where they differ [1]. Multiple sequence alignment is an optimization problem searching for the best alignment from large, complex search spaces [2].

Genetic Algorithms (GAs) are a biological inspired technology which conducts randomized search and optimization techniques guided by the principles of natural evolution and genetics. They are efficient, adaptive and robust search processes which produce near optimal solutions, having a large degree of implicit parallelism [1], which makes them suitable for solving the problem of multiple sequence alignment.

In our prior investigation, we have developed AlineaGA [3, 4], a GA which performs multiple sequence alignment of protein sequences. In its implementation, we use mutation operators previously introduced by other authors and we extend their initial concept by presenting new mutation operators with local search mechanisms. This optimization allows the improvement of a solution by searching its vicinity for fittest individuals and replacing it if a better one is found.

Our purpose is to compare the original mutation operators with the new improved ones, in terms of the quality of the obtained solutions and of population's evolution. For this matter, we tested AlineaGA with eight BAliBASE [5] Reference 1 alignments, using different combinations and probabilities of mutation.

This paper is organized as follows. In the next Section we introduce concepts underlying our research. In Section III, we present a brief explanation regarding AlineaGA methods. Section IV presents the mutation operators used by AlineaGA. The experiments performed in order to validate and observe the impact of these new operators are discussed in Section V. Finally, the concluding Section presents final considerations and topics for future work.

II. BACKGROUND

Multiple alignments are present in most of the computational methods used in molecular biology. They are used to study molecular evolution, to help predict the secondary or tertiary structure of new sequences, RNA folding, gene regulation and polymerase chain reaction primer design [6]. They are also applied to different areas such as functional genomics, evolutionary studies, structure modelling, mutagenesis experiments and drug design.

Two major methods exist for performing multiple sequence alignment. The progressive method gradually builds up the alignment by aligning the two most similar sequences first, and one after another, adding the more distant ones. This is a very fast and simple method; however it has a crucial problem: mistakes made at an intermediate step cannot be corrected later with the addition of more sequences to the alignment. In addition, it does not provide a metric which can be used to say that one alignment is preferable to another or to say that the best possible alignment, given a set of parameters, has been found [7]. On the other hand, iterative methods try to optimize a scoring function which reflects biological events such that

978-0-7695-3872-3/09 $26.00 © 2009 IEEE
DOI 10.1109/ISDA.2009.106

optimization of the score leads to a correct alignment [8]. One representative example of iterative methods are GAs, other examples may be found in our prior review [9].

A. Alignment

An alignment is a mutual placement of two or more sequences which reveal where the sequences are similar, and where they differ. An optimal alignment is the one that exhibits the most correspondences and the fewest differences, but which may or may not be biologically meaningful [1]. An example of an alignment of four hypothetical protein sequences is presented in Fig. 1.

```
-TISCTGNIGAG-NHVKWYQQLPG
-RLSCSSIFSS--YAMYWVRQAPG
L-LTCTVSFDD--YYSTWVRQPPG
PEVTCVVSHEDPQVKFNWYVQ-PG
```

Figure 1. Example of a multiple sequence alignment.

Each input sequence is represented in a different line and it can have different lengths. Columns with different symbols in the alignment show that several mutation events have taken place. Columns with identical symbols, which are represented as bold characters, indicate that no mutation occurs. The symbol "–" is used to represent a space introduced in the sequence in order to improve the alignment result. This space is usually referred to as a gap. The introduction of gaps in the sequences allows the alignment to be extended into regions where one sequence may have lost or gained sequence characters not found in the other.

B. Genetic Algorithms

GAs, introduced in 1975 by Holland [10], are stochastic algorithms whose search methods model some natural phenomena: genetic inheritance and Darwinian strife for survival [11].

In GAs, adaptation proceeds by maintaining a population of structures from which new structures are produced by means of genetic operators such as crossover and mutation [12]. Crossover combines the features of randomly chosen individuals (parents) to form two similar offspring by swapping corresponding segments of the parents. Mutation arbitrarily alters some values within the individual, by a random change with a probability defined by a mutation rate [6]. Each structure in the population has an associated fitness score, and these scores are used in a competition that establishes which structures are used to form new ones [12].

The capacity to take advantage of gathering information about an initially unknown search space, in order to bias subsequent search into useful subspaces, is a distinctive feature of GAs, making them suitable for problems with large, complex, and poorly understood search spaces [12].

III. ALINEAGA METHODS

This section presents a brief explanation regarding representation, selection, evaluation crossover and mutation in AlineaGA.

Our algorithm uses a noncodified representation of the individuals. This means that real multiple sequence alignments, as the one presented in Fig. 1, are used as data structures for each individual, resulting in chromosomes represented by arrays of characters on which each line corresponds to a sequence in the alignment, and each column represents a residue at a specific position.

The individuals are initially randomly generated, and then selected, combined and mutated in order to produce new solutions through the course of a defined number of generations.

The sum-of-pairs function, presented in (1), is used as a fitness measure to evaluate the population.

$$Sum-of-Pairs=\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} ScoringMatrix(l_i,l_j) \qquad (1)$$

This score is assessed by scoring all of the pairwise comparisons between each residue in each column of an alignment and adding the scores together [13]. To do this calculus, it is necessary to use a scoring matrix which determines the cost of substituting a residue for another, and also a gap penalty value to determine the cost of aligning a residue with a gap. We use the PAM 350 [14] scoring matrix with a gap penalty of -10 [3].

Parents' selection is determined by their fitness. We use Tournament selection for this purpose. This means that the fittest parents have more probability of reproducing themselves by means of a crossover operation.

Two crossover operators are used in AlineaGA: One Point crossover, which derives from Goldberg's standard one point crossover operator [15] with an extension that treats the existing gaps in each sequence; and RecombineMatchedCol [16], on which the fully identical columns of the first parent which don't appear in the second one are identified, and then one of these fully aligned columns is randomly selected and is generated in the second alignment, originating the offspring. These crossover operators are selected randomly within each generation.

Mutations can occur, introducing new characteristics in the population and thus, increasing diversity. Each mutation operator is randomly selected from a pool of operators and it is applied to an individual according to the defined mutation probability. If the mutated solution is worst than the original one, a new mutation can be applied to the mutated individual. This can be repeated until the fitness improves or during a certain number of attempts defined by the user. In our tests, we perform a maximum of 2 tries because it represents a good trade off between speed and robustness, without transforming completely the individuals in only one generation.

IV. MUTATION OPERATORS

We use three popular and effective mutation operators present in the literature: the Gap Insertion, Gap Shifting and Merge Space operators. We have improved these methods by hybridizing a simple local search optimization, resulting in three new operators: the "Smart" ones.

A. Gap Insertion

The Gap Insertion operator extends the alignments by inserting gaps into the sequences. It produces a mutation

identical to the GenAlignRefine [13] gap insertion operator which simply inserts a gap into every sequence of the alignment in a random fashion. In our implementation there is a small variation which allows the user to decide the number of gaps that he wishes to introduce on each mutation.

B. Gap Shifting

Shifting gaps is another way to introduce new alignment configurations. In RAGA's gap shifting operator, a gap is randomly chosen in an alignment and it is moved to another position [17]. We use the same concept.

C. Merge Space

This operator merges together two or three spaces of a sequence [18]. This is accomplished by randomly selecting two or three consecutive gaps of a sequence, which may or may not be adjacent, and then merging these gaps together. After that, they are shifted to a randomly chosen position in the same sequence, as Fig. 2 illustrates.

Figure 2. Merge Space mutation operator.

D. Smart Gap Insertion

This variation of the Gap Insertion operator only produces the mutation if the fitness of the mutated alignment is greater than the fitness of the original one. To do so, a random position in the alignment matrix (line and column indexes) is chosen. Then, a desired number of gaps is inserted on that position and all the other lines are filled with gaps until they all have the same size, as Fig. 3 exemplifies.

Figure 3. Smart Gap Insertion mutation operator.

A direction probability determines the insertion of additional gaps at the beginning or at the end of the sequence lines. This probability is initially set to 50% at the beginning of each generation, and according to the results it will be increased (increasing the possibility of the gap insertion at the beginning of the remaining sequences), or decreased (increasing the possibility of the gap insertion at the end of the remaining sequences). However, the mutation and the change on the direction probability will only occur if the fitness of the generated alignment is greater than the original one. If the operator is unable to improve the alignment at the first attempt, it tries to choose another random position and repeats the whole process. The defined number of maximum attempts is set to 3 but it can be customized according to user's needs.

E. Smart Gap Shifting

The Smart Gap Shifting exemplified in Fig. 4, tries to move the gaps of an alignment until its fitness improves. As in the Smart Gap Insertion operator, the shift direction is determined by a direction probability which initially is set to 50% and it is updated when better alignments are found. Also, the mutation occurs only if the fitness of the generated alignment is better than the original one.

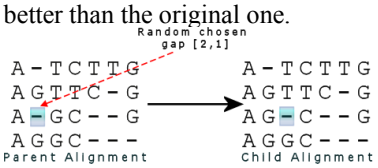

Figure 4. Smart Gap Shifting mutation operator.

F. Smart Merge Space

The Smart Merge Space is similar to the Merge Space operator but, it only applies the mutation if the fitness of the mutated solution is greater than the fitness of the original one.

G. Gap Column Remover

The Gap Column Remover (Fig. 5) removes gap columns that can be generated by crossover and mutation operations [3]. It is not influenced by the mutation probability and it is applied at the end of each generation.

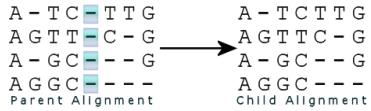

Figure 5. Gap Column Remover.

V. Testing and Results

Our objective is to evaluate the impact of the new "Smart" operators in the quality of the obtained solutions and to understand their influence in population's evolution. For this purpose we test different combinations of mutation operators as well as different mutation probabilities, maintaining all the remaining parameters unchanged.

In our tests, we use eight datasets from the Reference 1 alignments of BAliBASE [5]. Four of these datasets (1aho, 1fmb, 1plc, 2mhr) have more than 35% of identity among its sequences and the other half (1fjlA, 1hpi, 1pfc, 1ycc) have 20% to 40% of identity. We have measured the sum-of-pairs score of each one of these datasets. Later we use these reference results to evaluate the different test configurations.

A. Test Configurations

Although we vary the combinations of mutation operators and its probabilities, we maintain the remaining parameters stable. In all our tests the population size is 100, the number of generations is 2000, and the crossover probability is 0.8. The number of inserted gaps by the Gap Insertion and Smart Gap Insertion operators is 10 and the tournament size is 2. The remaining parameters for each one of the configurations is presented in Table I.

TABLE I. TEST PARAMETERS CONFIGURATION

Configuration	Type of Mutation Operators (Probability)
MxNormal20	Normal (0.2)
MxSmart20	Smart (0.2)
MxN&S20	Normal (0.2) and Smart (0.2)
MxNormal40	Normal (0.4)
MxSmart40	Smart (0.4)
MxN&S40	Normal (0.4) and Smart (0.4)
MxN20&S40	Normal (0.2) and Smart (0.4)

To simplify the description we separate the mutation operators into two categories: "Normal", which refers to Gap Insertion, Gap Shifting and Merge Space operators; and "Smart", which include the Smart Gap Insertion, Smart Gap Shifting and Smart Merge Space operators. In both cases, the mutation operators are randomly chosen with equal probability of selection.

B. *Results*

Next we present the results of our tests. All the results were obtained by averaging the sum-of-pairs score from 30 runs of AlineaGA for each configuration/dataset.

1) Population's Evolution: Figures 6 to 13 compare the population evolution on the test datasets for the different configurations. The "Normal" operators have worse performances than the "Smart" ones on all datasets. It is notorious that the configuration with higher probability of "Normal" mutation, "MxNormal40" has the worst performance. This may be because of the random nature of the operators.

Configuration "MxNormal20", with a lower mutation probability, presents a slow consistent evolution; however, it does not achieve the higher scoring values that the remaining configurations do in the same number of generations.

"MxSmart20" presents a great improvement in comparison with the "Normal" configurations; however, in the majority of the datasets, "MxSmart40" behaves better than the remaining configurations in the first 200 generations. Nevertheless, after population's stabilization, we can observe that there are some difficulties for this configuration to explore the search space for better solutions.

To increase the capacity of exploring different search spaces, we tested three combinations of "Normal" and "Smart" operators. Comparing "MxN&S20" and "MxN&S40", we can observe that although the first configuration evolves faster than the second, the slower convergence of "MxN&S40" allows it to find better solutions. In fact, this configuration presents the best performance for the half of the datasets: 1fmb, 2mhr, 1fjlA and 1pfc.

By combining low values of "Normal" mutations with high values of the "Smart" ones, "MxN20&S40" presents a fast convergence, only beaten by "MxSmart40". Yet, in this case, the faster convergence does not disable the capacity for exploring interesting spaces, allowing this configuration to achieve the best results for the remaining datasets: 1aho, 1plc, 1hpi and 1ycc.

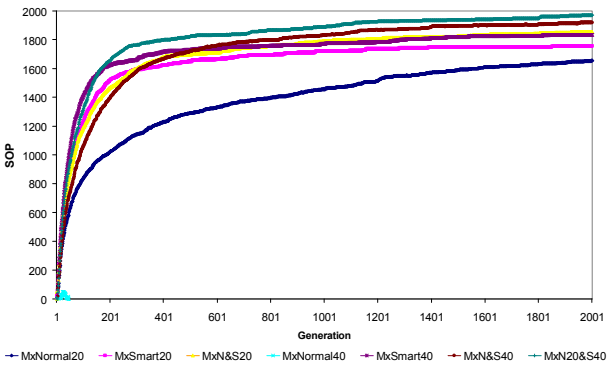

Figure 6. Population evolution for dataset 1aho.

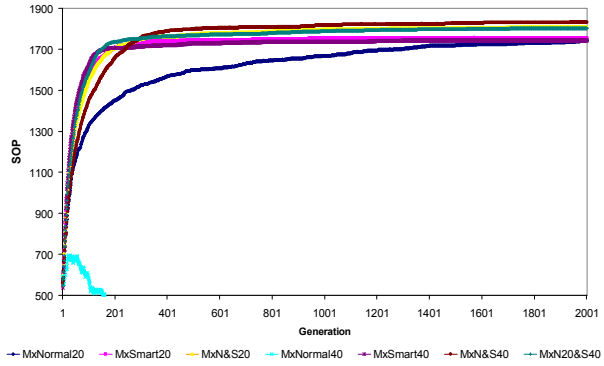

Figure 7. Population evolution for dataset 1fmb.

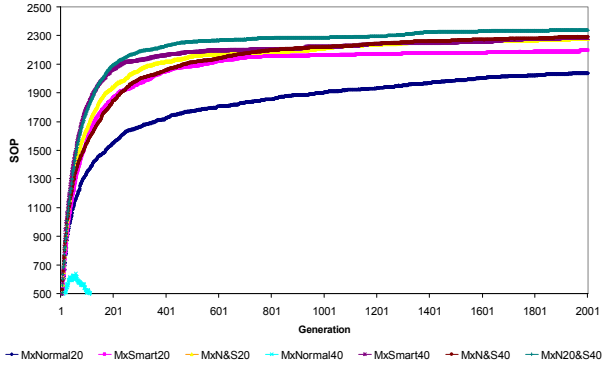

Figure 8. Population evolution for dataset 1plc.

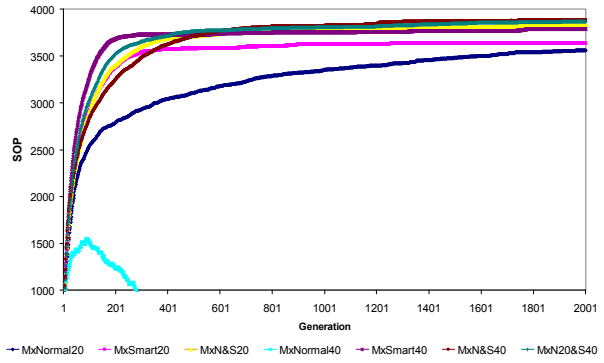

Figure 9. Population evolution for dataset 2mhr.

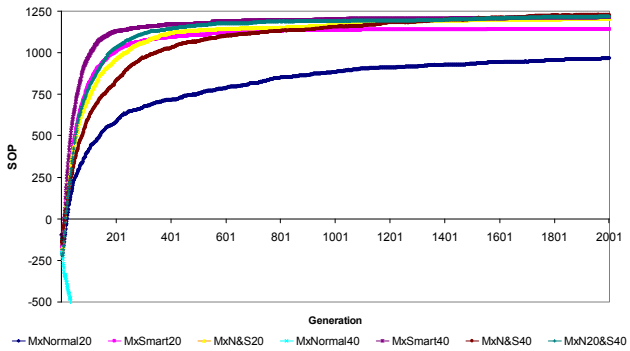

Figure 10. Population evolution for dataset 1fjlA.

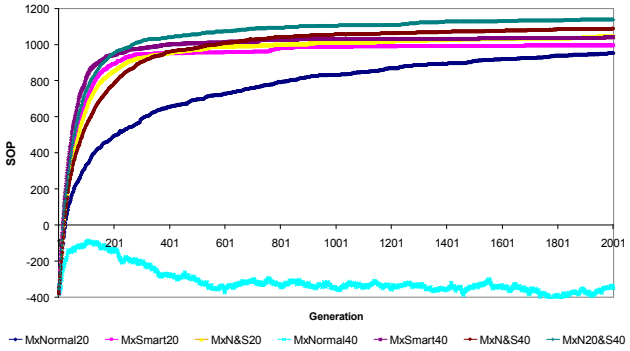

Figure 11. Population evolution for dataset 1hpi.

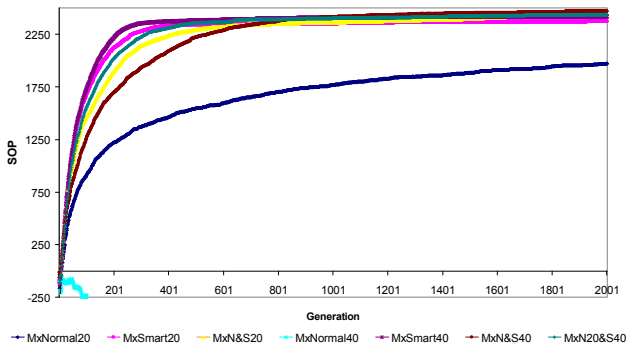

Figure 12. Population evolution for dataset 1pfc.

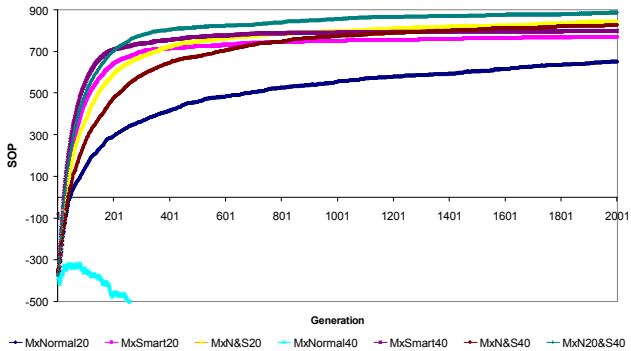

Figure 13. Population evolution for dataset 1ycc.

2) Performance: Table II summarizes the performance of all the configurations on the test datasets.

TABLE II. RESULTS FOR THE TEST CONFIGURATIONS

Dataset	SOP of BAliBASE	Configuration	Average Best SOP	Best SOP
1aho	2015	MxNormal20	1659,43	1850
		MxSmart20	1757,70	2063
		MxN&S20	1857,27	2115
		MxNormal40	259,60	581
		MxSmart40	1837,13	2097
		MxN&S40	1923,47	2143
		MxN20&S40	1974,93	2134
1fmb	1706	MxNormal20	1742,97	1816
		MxSmart20	1756,80	1850
		MxN&S20	1810,47	1848
		MxNormal40	888,70	1196
		MxSmart40	1743,07	1836
		MxN&S40	1834,20	1861
		MxN20&S40	1803,47	1855
1plc	2403	MxNormal20	2041,57	2328
		MxSmart20	2199,43	2516
		MxN&S20	2283,40	2577
		MxNormal40	842,67	1505
		MxSmart40	2285,80	2594
		MxN&S40	2293,17	2574
		MxN20&S40	2338,50	2536
2mhr	4002	MxNormal20	3571,73	3849
		MxSmart20	3636,50	4021
		MxN&S20	3827,87	4038
		MxNormal40	1821,93	2695
		MxSmart40	3786,37	4004
		MxN&S40	3889,60	4051
		MxN20&S40	3866,07	4038
1fjlA	1740	MxNormal20	976,40	1247
		MxSmart20	1141,77	1598
		MxN&S20	1206,83	1441
		MxNormal40	-73,40	431
		MxSmart40	1215,73	1451
		MxN&S40	1238,50	1574
		MxN20&S40	1219,63	1768
1hpi	1208	MxNormal20	954,33	1102
		MxSmart20	996,03	1183
		MxN&S20	1047,10	1194
		MxNormal40	106,10	298
		MxSmart40	1038,73	1193
		MxN&S40	1090,40	1212
		MxN20&S40	1140,73	1207
1pfc	2216	MxNormal20	1973,43	2328
		MxSmart20	2371,50	2508
		MxN&S20	2424,13	2521
		MxNormal40	194,77	843
		MxSmart40	2417,53	2515
		MxN&S40	2471,13	2559
		MxN20&S40	2438,27	2540
1ycc	963	MxNormal20	652,73	941
		MxSmart20	770,10	1006
		MxN&S20	844,63	1105
		MxNormal40	-163,13	200
		MxSmart40	799,07	1113
		MxN&S40	826,77	1101
		MxN20&S40	887,70	1114

SOP, sum-of-pairs; Average Best SOP was obtained by averaging the results of 30 runs of each test.

The "SOP of BAliBASE Alignment" column presents the sum-of-pairs score for the different datasets and it was computed using PAM 350 scoring matrix and a gap penalty

of -10. Column “Average Best SOP” shows the average sum-of-pairs score obtained in 30 runs of AlineaGA, and column “Best SOP” presents the best solution found in those 30 runs.

The best average results for all datasets are obtained by “MxN&S40” and “MxN20&S40” configurations. However, comparing with the BAliBASE score, the average sum-of-pairs of the test configurations are better in only two datasets: 1fmb and 1pfc. In the remaining cases the average sum-of-pairs value does not reach BAliBASE’s score.

The majority of the configurations allow the algorithm to find better fitness values for the best solution. The exceptions are the “MxNormal20” and “MxNormal40”, which use only the “Normal” operators.

“MxSmart40” obtains the best score in the dataset 1plc, but in general, it has a worst performance than the configurations that combine both “Normal” and “Smart” mutations.

Despite the fact that the “Smart” operators are superior to the “Normal” ones, their combined use in “MxN&S20”, “MxN&S40” and “MxN20&S40” configurations, proved to be the best solution for finding the best multiple sequence alignment on the test datasets. In these cases, a high probability of using the “Smart” operators leads to better results, but we cannot establish a similar relation for the use of the “Normal” ones. Probably, this may be due to the random fashion that is used in operator’s selection.

VI. Conclusions

Mutation operators have an important role in introducing new patterns in the population, being decisive for exploring solution’s search space. Our “Smart” operators have proved that they can influence the search for better solutions, leading to better results than the “Normal” ones.

Configuration “MxSmart40”, which uses only “Smart” operators with a probability of 0.4, has proven that in the first 200 generations can achieve better results than any other configuration, with gains that in some cases are superior to 100%. Though, if the number of generations is not a problem, the best option is to use a combination of “Smart” and “Normal” mutations. Our best results were attained with configurations “MxN&S40” and “MxN20&S40”. It is not easy to choose the best one, but “MxN20&S40” can converge faster than “MxN&S40” towards the solution without presenting premature convergence.

We believe that the results can be improved if the choice of operators is not made in a random fashion. Also, the number of generations and other scoring matrixes can have influence in the results for different datasets, but in this study we kept these parameters unchanged in order to establish a similar environment for all test configurations.

A straightforward development of our investigation is to determine a set of rules that can evaluate the evolution of the population, and that choose the correct type of operator at a given time.

References

[1] S. K. Pal, S. Bandyopadhyay, and S. S. Ray, "Evolutionary computation in bioinformatics: A review," *IEEE Transactions on Systems Man and Cybernetics Part C-Appl and Rev*, vol. 36, pp. 601-615, 2006.

[2] J. Horng, L. Wu, C. Lin, and B. Yang, "A genetic algorithm for multiple sequence alignment," *Soft Computing*, vol. 9, pp. 407-420, 2005.

[3] F. J. M. Silva, J. M. Sánchez Pérez, J. A. Gómez Pulido, and M. Á. Vega Rodríguez, "AlineaGA: A Genetic Algorithm for Multiple Sequence Alignment," in *New Challenges in Applied Intelligence Technologies*, vol. 134, *Studies in Computational Intelligence*, N. T. Nguyen and R. Katarzyniak, Eds.: Springer-Verlag, 2008, pp. 309-318.

[4] F. J. M. Silva, J. M. Sánchez Pérez, J. A. Gómez Pulido, and M. Á. Vega Rodríguez, "Un Algoritmo Genético para Alineamiento Múltiple de Secuencias," presented at VI Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB 09), Málaga, Spain, 2009.

[5] J. D. Thompson, F. Plewniak, and O. Poch, "BAliBASE: a benchmark alignment database for the evaluation of multiple alignment programs," *Bioinformatics*, vol. 15, pp. 87-88, 1999.

[6] L. A. Anbarasu, P. Narayanasamy, and V. Sundararajan, "Multiple molecular sequence alignment by island parallel genetic algorithm," *Current Science*, vol. 78, pp. 858-863, 2000.

[7] C. Notredame and D. G. Higgins, "SAGA: sequence alignment by genetic algorithm," *Nucleic Acids Research*, vol. 24, pp. 1515-1524, 1996.

[8] T. Lassmann and E. L. L. Sonnhammer, "Quality assessment of multiple alignment programs," *FEBS Letters*, vol. 529, pp. 126-130, 2002.

[9] F. J. M. Silva, J. M. Sánchez Pérez, J. A. Gómez Pulido, and M. Á. Vega Rodríguez, "Alineamiento Múltiple de Secuencias utilizando Algoritmos Genéticos: Revisión," presented at Segundo Congreso Español de Informática, Zaragoza, Spain, 2007.

[10] J. H. Holland, "Adaptation in natural and artificial systems, Univ Mich Press," *Ann Arbor*, 1975.

[11] Z. Michalewicz, *Genetic algorithms + data structures = evolution programs - Third, Revised and Extended Edition*, 3 ed: Springer, 1996.

[12] K. De Jong, "Learning with genetic algorithms: An overview," *Mach Learning*, vol. 3, pp. 121-138, 1988.

[13] C. Wang and E. J. Lefkowitz, "Genomic multiple sequence alignments: refinement using a genetic algorithm," *BMC Bioinformatics*, vol. 6, 2005.

[14] M. O. Dayhoff, R. M. Schwartz, and B. C. Orcutt, "A Model of Evolutionary Change in Proteins," in *Atlas of Protein Sequence and Structure*, vol. 5: National Biomedical Research Foundation, 1978, pp. 345-352.

[15] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning* Reading, MA: Addison-Wesley Publishing Company, 1989.

[16] K. Chellapilla and G. B. Fogel, "Multiple sequence alignment using evolutionary programming," presented at Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on, Washington DC, USA, 1999.

[17] C. Notredame, E. A. O'Brien, and D. G. Higgins, "RAGA: RNA sequence alignment by genetic algorithm," *Nucleic Acids Research*, vol. 25, pp. 4570-4580, 1997.

[18] J.-T. Horng, C.-M. Lin, B.-J. Liu, and C.-Y. Kao, "Using Genetic Algorithms to Solve Multiple Sequence Alignments," presented at Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000), Las Vegas, Nevada, USA, 2000.