Emerging Traits in the Application of an Evolutionary Algorithm to a Scalable Bioinformatics Problem

Jorge Cervantes, Máximo Sánchez, Pedro Pablo González

***Abstract*—In this work we present an empirical study on the application of a genetic algorithm featuring rank based variation operators (named Rank GA) to one scalable problem in the area of bioinformatics (Protein Folding in a 2D square lattice) trying to discover the main traits that emerge as the scale of the problem grows. This study has a double intention: 1) to develop a robust easy-to-understand evolutionary algorithm for solving the protein folding problem and 2) to grasp some more general and theoretical intuition from this kind of difficult problems that often are being approached through evolutionary algorithms. We show how the use of a squared 2D lattice in the model can influence the outcome of the Rank GA and also how this algorithm compares in performance with previously published results. The Rank GA seems to perform better than other EAs as the problem size scales up.**

I. Introduction

THE understanding of protein folding is considered one of the cornerstones in structural and therefore functional biology [1]-[6]. Comprehension of this process is, on the one hand, a huge challenge due in part to an extreme combinatorial diversity and also to the subtlety in the type and number of interactions which can sum millions of them, including both stabilizing and unstabilizing interactions. On the other hand, this knowledge represents the promise of really rational protein engineering, to modify stability, selectivity and optimal conditions of biological activity to adapt these macromolecules to technological purposes. Furthermore, the detailed knowledge of the rules that govern the folding of a polypeptide chain might lead to protein design, a de-novo prediction of the structure that a particular sequence might adopt, or (in the inverse folding problem) the sequences compatible with a proposed particular 3D structure. Protein folding is indeed a largely waited scientific breakthrough capable to solve an enormous amount of problems in biomedicine, therapeutics, materials science, genetic therapy, food processing, bioremediation, alternative energy, among others, since protein engineering and design would take the efficiency and functionality of biological activity to any productive process in current and future life.

The use of computational techniques in Biology (tools, simulations, algorithms) has emerged in recent years as a complementary approach to experiments. Due to the enormous amount of possible 3D configurations for each flexible linear-chain, any protein structure prediction method needs a way to explore the space of possible structures efficiently (a search strategy), and a way to quantitatively identify the most plausible structures (an evaluation or fitness function). Search algorithms can be either exhaustive or approximate. In the first case, a proven optimal solution is returned that satisfies the problem constraints with a predefined evaluation function, whilst in the second case, there is no proof of optimality and the solution returned could be sub-optimal.

In many practical scenarios, such as the one we are considering in this work, an approximation algorithm is not only preferable but compulsory because it is typically far more efficient than the exhaustive one for exploring huge conformational spaces, and when the proof of optimality is not required. One of the most employed for the protein folding problem is the family of EAs (Evolutionary Algorithms). These techniques implement mechanisms of solution searching named genetic operators; some of them are: selection, crossover, mutation and some others that have been designed for particular applications. EAs also involve optimization techniques such as genetic programming, evolutionary programming, evolutionary strategies, estimation of distribution, genetic algorithms (GAs) and some others.

In this paper we describe the design of an EA in the category of Genetic Algorithms. We explain the reasons behind each choice of genetic operators, as thoroughly as we can, using concepts learned from the study of GA theory.

We then perform an analysis of basic forms of the protein folding problem scaling step by step towards more complex problems. This is done to understand what are the traits that emerge while the problem scales up and then to extract some general knowledge that could be used later to understand larger problems that are quite intractable with the hope that this can help. Then we use some other well-known problems in the literature to compare performance of our design against other more common algorithms. Finally, we study larger problems that are also well known, in terms of what we learned and then we throw our conclusions.

II. The Protein Folding Problem

Proteins are sequences of amino-acid molecules of different types. Each of these molecule types has different properties that have an influence on the way the whole sequence folds into its functional configuration. For this

Manuscript received February 7, 2010.

J. Cervantes (jorgecervanteso@aim.com), M. Sánchez (maxsanchez@prodigy.net.mx) and P.P.González (ppgp@servidor.unam.mx) are with the Universidad Autónoma Metropolitana, Artificios 40, Col. Hidalgo, Del. Álvaro Obregón, D.F. 01120 MEXICO.

study, (considering some previously published ones [15][16] have done the same) only two different properties of the protein sequence are modeled: Hydrophobic and Hydrophilic amino-acids (the HP model [21]). This property itself is interesting because it determines, among other important properties, to some extent, how the protein folds since, when the protein is immerse in water, hydrophobic molecules tend to settle in the middle of the folded protein whereas hydrophilic molecules tend to stay in the periphery of it. As an evolutionary computation problem, the problem is to find the folding configuration that is more likely to be found in nature.

In the model we used here, each bead in the sequence can only occupy positions in a 2D square grid. So, at each bead the sequence can either turn right 90 degrees or turn left or go straight. A fitness function has been used in which every contact between non-consecutive H beads (H-H contact) increases the fitness value of the configuration by 1. No other considerations are taken into account as has been done also in [15]. This is done to favour the accumulation of H beads together while the position of P beads is indifferent.

The problem, as defined, is very difficult since there are many interactions (epistatic effects) between each folding in the same configuration. Sometimes the H beads in the beginning of the sequence form contacts with H beads at the end producing epistatic dependencies between beads that are placed far from each other, thus, producing higher chances of being broken by, for instance, a crossover operator. Many local optima are then formed that the search algorithm must overcome.

III. The Bioinformatics Tool

In a previous publication [11] we described the first version of a bioinformatics framework called EVOLUTION, designed to perform virtual experiments. This framework is designed to deal with optimization problems of biological systems permitting its adaptation to include new tools and functionalities through the feedback from the obtained results. The current version of the bioinformatics framework is devoted to studying and exploring the protein folding problem, by integrating different lattice bead models [7][8][14], evolutionary algorithms and other computational techniques into a Multi-Agent System architecture.

The generic bioinformatics framework architecture can be seen in Figure 1, where its main components are the agents and the blackboard. As can be seen in this figure, the architecture provides three types of agents: model agents, algorithm agents and interface agents. According to this Multi-Agent System architecture, the interaction among agents is realized by indirect communication, through a communication-coordination abstraction, represented in this figure by the blackboard. Figure 2 shows the semantics assigned to the different blackboard levels and types of agents when protein folding is modeled.

Evolution, the bioinformatics framework, allows inducing folding of molecular-chain representations with the aim to explore both the behavior and efficiency of EA by following the most relevant variables of chain folding. Regarding the software development, our work will progress by using the bioinformatics framework as a scaffold or mainframe devoted to the development and testing of gradually more complex and robust EA and their associated parameters, with more realistic representations of the problem and with improved fitness functions. This framework would also allow adding functionalities for the user to tract results and algorithm efficiency, to interactively bias the conformational search with biological sense, or to perform graphical and numerical analysis.

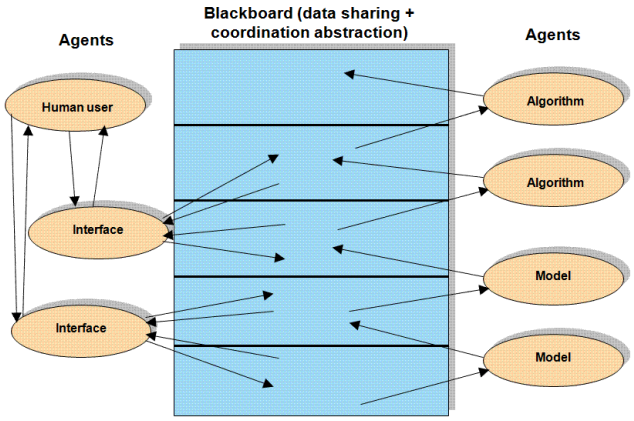

Figure 1. General overview of Evolution architecture. Interaction among agents is realized indirectly through a blackboard, which represents the communication-coordination abstraction.

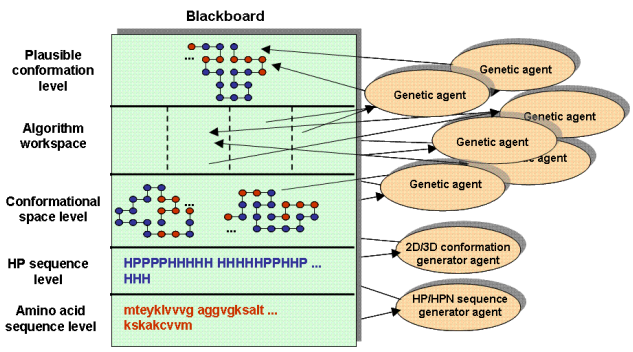

Figure 2. Semantics assigned to the different blackboard levels and types of agents in Evolution when modeling protein folding.

The EA described next was implemented in this framework as an addition to it allowing its use in full or partially. Internally, only a few new extra agents had to be added to the system.

IV. Algorithm Design

During the design of the algorithm we have considered the ideas in [15][18]. We explain in the following the reasons that led us to adopt some of the ideas there and to include the

parts that are new.

A. Flow of the Rank GA

The general flow of the Rank GA used in this work is the following:

Random Initialization with **correction operator**
Evaluation of Population and sort by fitness
While not exit criteria met do
  Fitness Proportional Selection then **sorting** by fitness
  Rank-based Recombination with **correction operator**
  Evaluation of Population then **sorting** by fitness
  Rank-based Mutation with **correction operator**
  Evaluation of Population then **sorting** by fitness

B. Constraint Handling by a correction operator

We use a correction operator as in [15] that, sequentially from the first to the last, checks each link *i* in a chain if it leads to an overlap of the next node with any previously checked node and, if so, it changes that link *i* to point to another position randomly. If there is no position that leads to a free cell in the grid then the position of the previous link at *i*-1 is marked as unsuccessful and the whole procedure is repeated at that link and so on until a successful position is found for every link.

Recursively speaking, the procedure can be explained with a C-like boolean function:

```
bool check( TNode * node ) {
  if( node.next == null )    return true;
  if( node->next->isBusy() ) return false;

  for( int i=0; i<3; i++ ) { // 4 directions
    if( check( node->next ) )
      return true;
    node->next = …randomly choose any of
                  the remaining directions;
  }
  return false;
}
```

By using this operator before each fitness evaluation, individuals are transformed into valid conformations without any overlaps at all. We assume that this is good for search since there are just too many invalid conformations in the search space. The alternative to this operator would be some kind of penalty function that considers the number of overlaps as its parameter, as in [16]. This penalty would be too difficult to fine tune for each problem instance and there is no theory nowadays that can tell us how to do this with a reasonable level of confidence. Therefore we opted for the correction operator described above which guarantees that every configuration will result in a valid one.

C. Rank Based Mutation

Following the ideas in [17] and [18] we have created a mutation operator that is applied with different probability for each individual in the population according to the rank that each one has when the population is sorted by fitness value. This probability is assigned to individual *i* as follows:

$$p_i = p_{\min} + (p_{\max} - p_{\min})\frac{Rank(i)-1}{s-1}$$

where $p_{\min}$ is the minimum mutation rate, $p_{\max}$ is the maximum mutation rate, s is the size of the population and $Rank(i)$ is the rank of individual i (ranging from 1 to s).

This operator is intended to provide robustness to search [17] since it implies the use of some level of elitism while at the same time it performs local and global search. Elitism is achieved when $p_{\min}$=0 because the individual with rank 1 is assigned this rate and remains unchanged for the next generation. Local search is performed by individuals with low mutation rates that are those closely ranked to the best one since many of them are clones of it. Global search is performed by the worst ranked individuals who use a high mutation rate. As in [17] and [18] we used for our experiments $p_{\min}$=0 and $p_{\max}$=1. The size of the population was set equal to the length of the genome length that in our case is the length of the amino acid sequence.

We rely on this kind of operator because we want to see how it compares with a normal fixed mutation rate and also because the fitness landscape in the protein folding model problem presents many local optima and high epistasis, which means the search algorithm could easily get stuck. This high epistasis level is evident by thinking that the first H bead in the sequence can form an H-H contact with the last one, and so, one change in the folding of any bead in between can alter the optimal folding in any of these.

D. Rank Based Recombination

Recombination is usually applied to any two individuals in the population. It is intended to unite the best genes of each individual into one hoping this process results in a better one. When a highly ranked individual is recombined with a low ranked one it is very likely to destroy both rather than construct a better solution because the worse one can contain genes that are quite incompatible with those of the better one [17]. We allowed then only nearly ranked individuals to recombine by using a parameter that indicates the maximum difference in rank that two individuals can have to be recombined. We allowed for our experiments mates to be 1 or 2 positions apart only.

E. Absolute vs Relative Encoding

The encoding of the folding at each bead of the amino acid sequence can be represented with an absolute or a relative encoding. Absolute encoding means that each code represents always the same direction in the physical space where the folding takes place. For instance, to the positive side of the X axis. In the relative encoding one code represents a turn, be it to the right, left, or no turn.

We chose the former because we want to reduce as much as we can the epistatic effects of mutation. It should be clear

that, using a relative encoding, a single mutation can produce a lot of overlaps that will consequently have to be corrected by the correction operator and so a single mutation would potentially be very destructive of some building blocks that have already been found in other parts of the genome. Under absolute encoding, one mutation in one gene produces fewer changes in the positions of the rest of the sequence and thus epistasis is reduced.

F. Selection

We used in all cases a fitness proportional selection scheme in order to keep this operator as simple as possible and close to the canonical Simple Genetic Algorithm (SGA)[13]. Thus we should be able to see the effects of using our other operators versus not using them.

V. Experiments and Results

We performed a number of experiments classified in two different groups: A) Configuring the problem from the simplest case scaling up towards more complicated ones, B) Problems already reported in related literature [15] where the optimum solution is known and other algorithms have been applied.

With A) we look for traits that could emerge as the problem scales up. By studying results in tractable problem instances we expect to find some phenomena that could clarify us the behavior of the algorithms at instances where it is more difficult to understand its dynamics.

With B) we want to compare the performance of our algorithm with similar ones and others previously published. We also want to see if our algorithm can shed some light in the study of how proteins tend to fold and if there is some phenomena that could be identified and explained.

A. Scaling from the Base Case

We performed a series of experiments in which we used only H beads in the sequences and made them grow from H9 to H29 one by one. Here we present the results that were more revealing of the properties that these problems exhibit.

The H9 sequence has several optimal configurations with 4 H-H contacts, one of which is depicted in Figure 3. The shape of all optimal conformations for sequence H9 happens to be a square of 3x3.

Turning our attention now to the sequence H10 we find that the optimal conformations include square shapes with an extra bead floating around and producing no H-H connections. But there are also some other conformations that present a different shape in which the last bead does produce at least one H-H connection, thus contributing to the fitness of the sequence. Figure 4 shows examples of these two cases. Quite surprisingly, our algorithm needs less effort to find one of these optimal configurations than what it takes to do so for the H9 sequence. This might be because with this extra bead, although the search space is bigger, there are more resources to construct either a square or the other shapes while the number of possible H-H connections remains the same at 4.

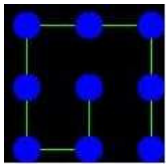

Figure 3. One of the optimal configurations for the H9 sequence with a square shape.

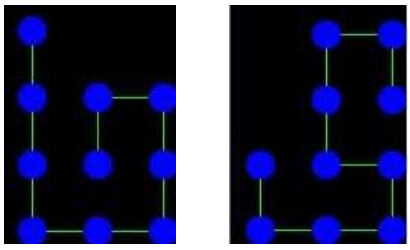

Figure 4. Two differently shaped optimal configurations for the H10 sequence.

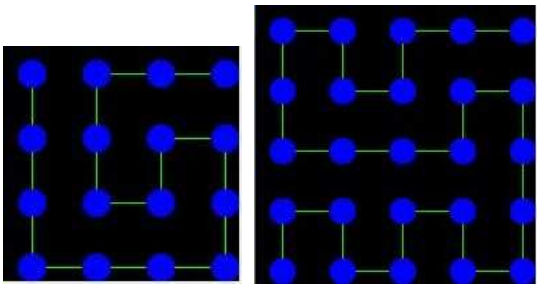

Figure 5. Examples of optimal configurations for the H16 (left) and H25 (right) sequences with square shapes.

Something similar happens with sequences H16 and H25 for which the optimal configurations are always squares of 4x4 and 5x5 respectively. Examples of these are shown in Figure 5. We assume now that any sequence of only H beads with length n^2 will have this property of having only square shaped optimal configurations.

In Figure 6 we show the average and median of the effort (in number of generations) that our algorithm needs to find an optimal sequence in this kind of problems as a function of the sequence length. One can see that there are peaks of higher (and lower) effort that arise with some regularity. These peaks correspond to the cases described above where the only possible shape of the optimal sequences is a square.

This effect can be explained by analyzing the possible optimal sequences for cases with longer sequences. Take for instance H22 for which we show optimal configurations in Figure 7. In this case it is not possible to form a square. There are many optimal configurations that resemble the one in Figure 7 (left) where the shape seems like a square of 5x5 with 3 missing positions that are always non internal in the folding of the sequence. Other optimal configurations present shapes similar to that in Figure **7** (right) where the

shape is a rectangle of 6x4 with 2 empty positions also at the peripheria.

These empty spaces provide some "space", inside the square, for other optimal configurations. This means that the algorithm can find any of these differently shaped optima and it does not need to find a particular one as is the case for H25 where only a square would be optimal.

In the case of H26, the shape of an optimal sequence does not have to be square either. Note in how the extra node in the sequence with respect to H25 provides no extra H-H contacts but it certainly provides additional options to create the 16 contacts that are possible as in Figure 8 (right).

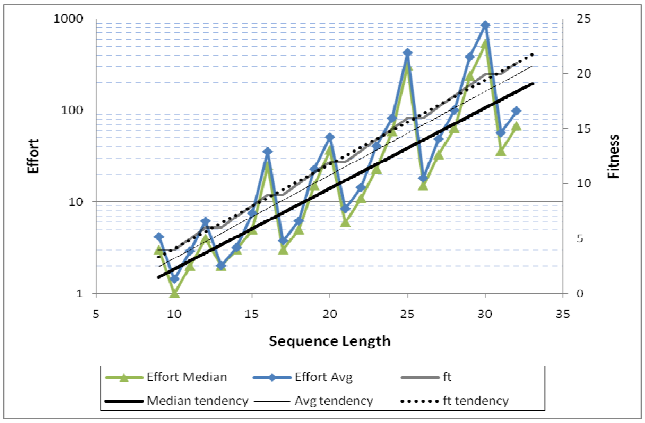

Figure 6. Average and median of the effort (generations) by our algorithm to find one optimal sequence for problems with only Hs, fitness of the optimal sequence and their tendency function as a function of sequence length.

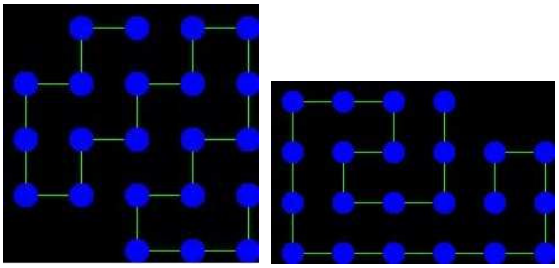

Figure 7. Optimal configurations for H22 in a 5x5 square (left) and in a 6x4 rectangle (right).

Another observation that can be done is that in Figure 6 it seems that the problem difficulty grows generally in an exponential way with the problem size. This is in concordance with Oliveto et al [20] who have formally proved this on a simpler algorithm using only mutation.

It is notable also that the empty spaces in the 6x5 rectangle always occupy one of the corners or they are next to another empty space. This is logical since more internal positions have the chance to create more H-H contacts than corners where no contacts are possible. The consequence is that optimal sequences tend to be more rounded than rectangular or square. The examples shown in Figure 8 and Figure 9 illustrate this.

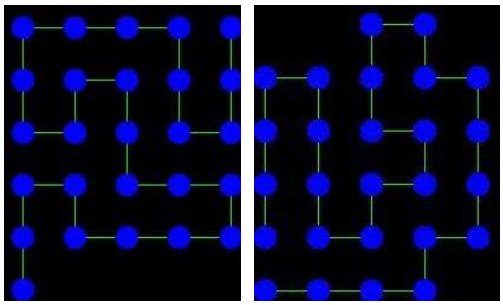

Figure 8. Optimal configurations for H26 in a 6x5 rectangle.

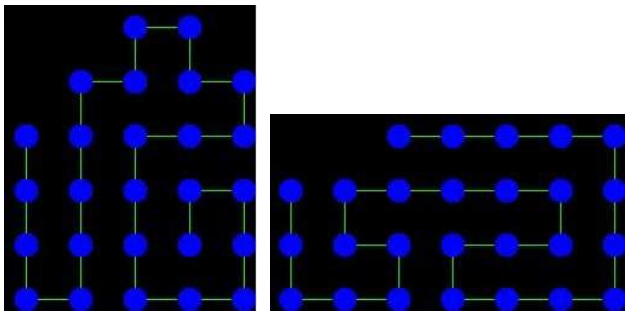

Figure 9. Examples of optimal configurations for H26 in a 6x5 rectangle (left) and in a 7x4 rectangle (right).

It is possible that some optimal sequences occupy a longer rectangle as is shown in Figure 9 (right) where the shape is a 7x4 rectangle, but these shapes are not found often because they are rare. One can think that this comes from the more limited degree of freedom given by a reduced number of empty spaces. In this case there are only 2 empty spaces compared to 4 in the previous examples.

The previous examples have shown that optimal configurations of sequences with n H beads only, can be difficult to be found if $\sqrt{n}$ is integer and a lot easier if $\sqrt{n-1}$ is integer. But that is not all. As it was shown in Figure 6, there are more peaks of high difficulty for other sequence lengths than those just mentioned. More generally, the difficulty seems to depend on the possible factorizations of n. If n has a factorization a^2 or one with two integers a and a+1 then the problem is more difficult than those where this factorization cannot be done with integers that are as close. For instance, see in Figure 6 the cases for string lengths of 12=4x3, 20=5x4 and 30=6x5. The next peaks beyond the plotted ones are then expected to be at 36, 42, 49, 56, 64, and so on. We don't know if there are more peaks present for higher values of n. It is possible that some other effects emerge then.

B. Problems in Literature

We have tested our algorithm design under some existing problems. We chose four problems taken from [15].

1) The HP-20 problem

The first problem we used is the folding of a sequence of 20 H-P beads represented by

HP-20 = H-P-H-P_2-H_2-P-H-P_2-H-P-H_2-P_2-H-P-H =
H-P-H-PP-HH-P-H-PP-H-P-HH-PP-H-P-H

In Figure 10 we show one of the optimal configurations for this case.

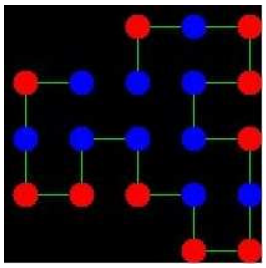

Figure 10. An optimal configuration for HP-20.

We have compared our algorithm which we called Rank GA as in [17][18] with an algorithm that uses fixed mutation rates with various values including the one that produced the best result and a normal recombination operator with probability 1 which we named SGA or Simple GA. Both using the same parameters for population size which was set to be 20 in this case.

Figure 11 shows plots of Run Length Distributions (RLD) [19] for these cases. RLDs were made by taking 100 runs for each algorithm and then sorting this runs by the effort (generations) taken to find the first optimal configuration. Clearly the RGA does not have the best performance but it should also be clear that only a few of the settings of the mutation rate for the SGA were able to outpace the RGA. This range of good mutation rates for an SGA is difficult to find, especially in large problem instances. The RGA has a reasonably good performance and the great advantage that this performance is robust against parameter changes while a SGA is very sensitive. This sensitivity of the SGA is shown in Figure 12 where we plot the estimated probability that the RGA outpaces the SGA in this problem as a function of the mutation probability used in the SGA. As can be seen, the RGA is expected to be outpaced by the SGA that uses a mutation rate between 0.032 and 0.085. This range is quite narrow and a small change in the mutation rate can make the SGA much worse. The plot does not show those cases because all of them are too bad for the SGA.

Comparing our results with those of Cox et. al. [15] we see that they have been able to produce an algorithm that shows similar behavior of the best of our SGAs. This is good for us since that algorithm is quite complex. It includes a special crossover, several different mutation operators (5), a "duplicate predator" and a so called "Brood Selection". They have reported that the duplicate predator is the one that makes that algorithm efficient. Without it, the reported efficacy when capped at 60000 fitness evaluations is of 65%. This is comparable to our RGA as can be seen in Figure 11 at an effort of 3000 generations (with 20 individuals) giving an estimate of 80% (20% probability of failure). We think that some of the operators used by Cox et. al. are making harm to the efficiency of the algorithm. Perhaps it is too complex. Besides that, it is difficult to implement and possibly sensitive to parameter tuning too. Our solution, though it does not provide the same efficiency, it is simple and robust against parameter tuning. After all, algorithms are intended to be used in problem instances where the optimal solution is unknown. This is where robustness is a true advantage.

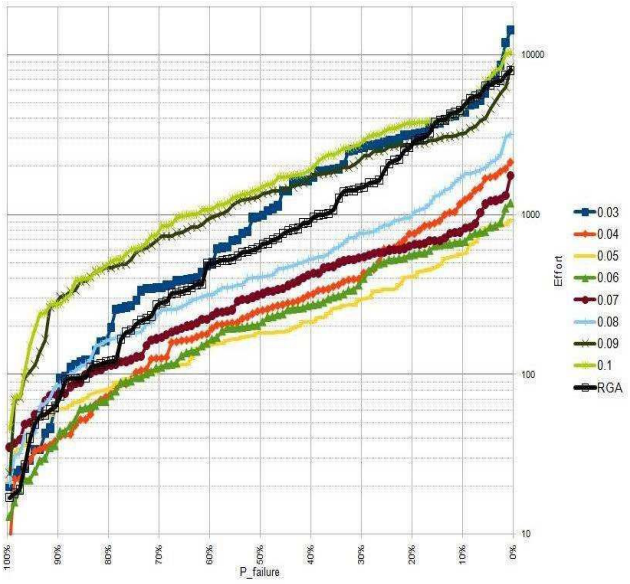

Figure 11. Plots of Effort to find an optimum (generations) vs. probability of failure of our RGA and several (the best ones) settings of the mutation rate of an SGA in the H20 problem.

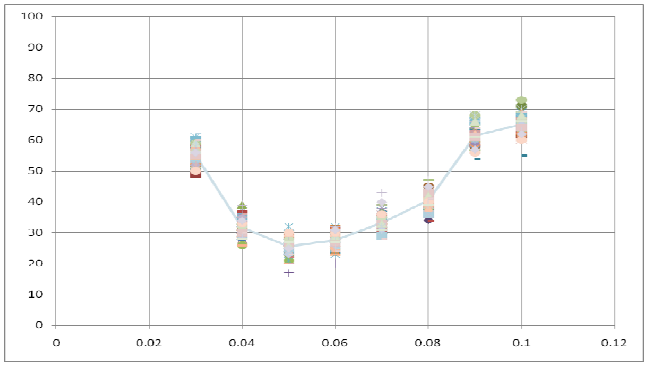

Figure 12. Probability that our algorithm outpaces an SGA in the H20 problem as a function of the probability of mutation of the SGA.

2) The HP-24 problem

The HP-24 problem is defined by the following sequence:

HP-24 = $H_2P_2(HP_2)_6H_2$ =
HH-PP-H-PP-H-PP-H-PP-H-PP-H-PP-H-PP-HH

In this case we present performance results in Figure 13. As can be seen, the results of Cox are apparently favorable against our algorithm and an SGA with mutation rate of

1/N≈0.04. It would have been good if those results were done using an RLD in order to be able to compare the whole distribution.

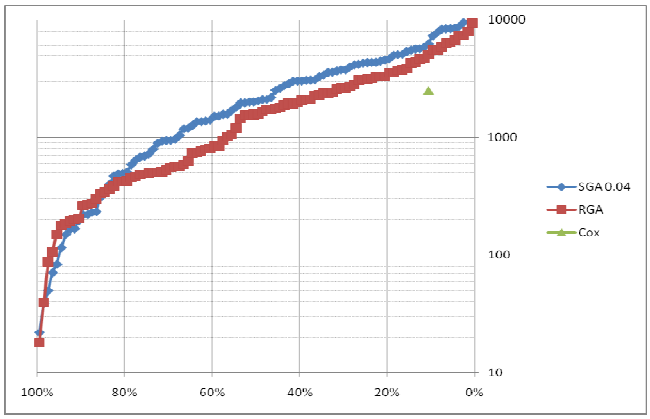

Figure 13. Plots of Effort to find an optimum (generations in *log* scale) vs Probability of failure (*%*) for our RGA and a typical SGA that uses a fixed probability of mutation of 0.04≈1/N.

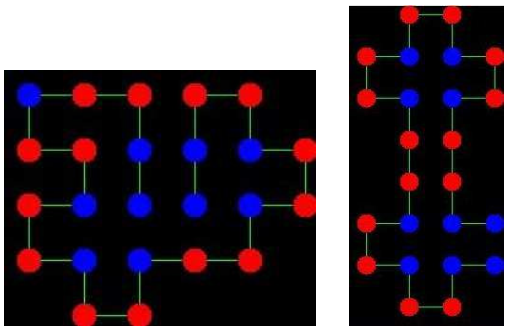

Figure 14. Two of the possible optimal configurations for the HP-24 problem.

Typical optimal configurations are shown in Figure 14. These configurations illustrate how different they can be implying that they must be far from each other in the search space. This in turn means that a good search algorithm should be able to explore the search space to regions that are far away from the current optimum if one wants to get all possible optimal configurations. For instance, if the RGA is run for a long time one could see that it would explore these far regions even when one of the optima has been found. Of course the probability to find them could be low (since any suboptimal configuration would tend to be discarded) but certainly higher than with other algorithms that concentrate in a small region near the current optimum.

3) The HP-36 problem

The HP-36 problem is represented by:

HP-36 = $P_3H_2P_2H_2P_5H_7P_2H_2P_4H_2P_2HP_2$ =
PPP-HH-PP-HH-PPPPP-HHHHHHH-PP-HH-PPPP-
HH-PP-H-PP

This is a bigger problem than the previous ones and now our algorithm has better performance than the one in [15] as can be observed in Figure 15. Our algorithm in this case was capped at 3000 generations.

Cox's algorithm has an efficacy of 5% using 60,000 evaluations (corresponding to 1666 generations with 36 individuals) while ours yields about 14%. It seems that as the problem becomes larger our algorithm tends to improve over the performance of Cox's algorithm.

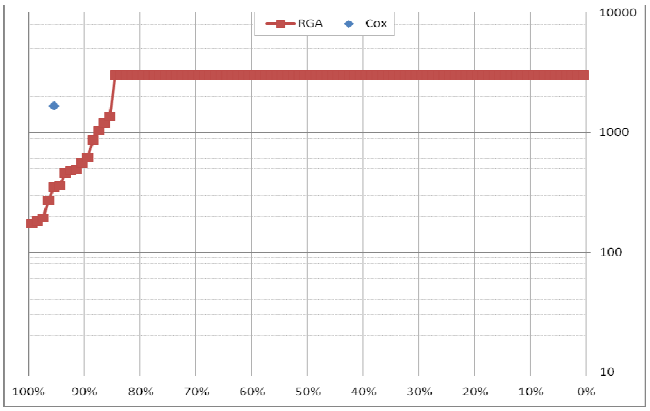

Figure 15. Comparison of our RGA (RLD) with the algorithm of Cox et al. [15] (dot) in the HP-36 problem.

4) The HP-48 problem

This problem is defined by the sequence

HP-48 = $P_2H(P_2H_2)_2P_5H_{10}P_6(H_2P_2)_2HP_2H_5$ =
PPHPPHHPPHHPPPPPHHHHHHHHHH

In this problem the optimal sequences have all a square shape in the H beads with several options for the P beads. This problem was reported in [15] as very difficult. According to the results in previous sections, this problem is likely to be difficult simply because of this square shape. Any algorithm should have problems here. Our algorithm does too.

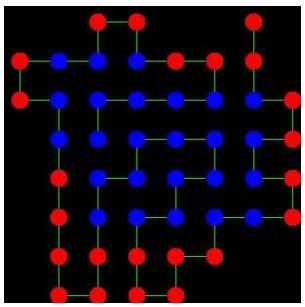

Figure 16. A near optimal configuration for HP-48.

Figure 16 shows one nearly optimal configuration found by our algorithm. One can see that the shape of the H beads is more or less close to a rounded square. This is the kind of shapes that are more easily found by our algorithm and we think that this is also the one that is more likely to be present in nature. The 2D square model induces a bias that makes it difficult to find square shaped optima. So we think that it is not good to try and solve this kind of problems. It should

suffice with the use of more rounded shapes to be able to study the traits present in real life protein folding.

VI. Conclusion

We have tested a Rank GA in larger multimodal problems than those reported in [17] and [18] showing similar results. As the problem is scaled up to larger instances, this algorithm seems to behave better. This kind of algorithms, due to their robustness, seems promising for applications with fitness landscapes that are known to be multimodal but where little more knowledge is available due to their size. This robustness is also good when trying to study simpler cases because the results are a good indicative of landscape features because they perform local and global search at once.

An in-depth systematic analysis using a Rank GA can guide us to understand the problem at hand. In protein folding on a 2D squared lattice, we found that problems where the optimal configuration has a square shape are harder to solve by the algorithm than others and in some cases it can be much harder. There is no need, for the study of Protein Folding, to use such problem instances since it seems unlikely that this is the case in nature. It would be more productive if longer sequences were used knowing that they have easier to find optima. Also, even in case the global optimum was difficult to find, the study of suboptimal configurations found by the algorithm is of interest because for nature it is just as difficult and real life proteins could have similar shapes.

We are currently working on further improvements to the mutation operator in order to obtain a less epistatic operator (more local) and thus we hope to be able to make better use of the building blocks that are present in the population.

Acknowledgment

The authors wish to thank Arturo Rojo and Hiram Beltrán for their support and the anonymous referees for their useful comments.

References

[1] E. Callaway, The shape of protein structures to come, Nature. 449 (2007) 765.

[2] E.J. Dodson, Computational biology: Protein predictions, Nature. 450 (2007) 176-177.

[3] R.J. Ellis, Protein folding: Inside the cage, Nature. 442 (2006) 360-362.

[4] K. Sugase, H.J. Dyson, P.E. Wright, Mechanism of coupled folding and binding of an intrinsically disordered protein, Nature. 447 (2007) 1021-1025.

[5] Z. Zhou, Y. Bai, Structural Biology: Analysis of protein-folding cooperativity, Nature. 445 (2007) E16-E17.

[6] M. Socolich, S.W. Lockless, W.P. Russ, H. Lee, K.H. Gardner, R. Ranganathan, Evolutionary information for specifying a protein fold, Nature. 437 (2005) 512-518.

[7] C. Thachuk, A. Shmygelska, H.H. Hoos, A replica exchange Monte Carlo algorithm for protein folding in the HP model, BMC Bioinformatics. 8 (2007) 342.

[8] A. Shmygelska, H.H. Hoos, An ant colony optimisation algorithm for the 2D and 3D hydrophobic polar protein folding problem, BMC Bioinformatics. 6 (2005) 30.

[9] M. Sadqi, D. Fushman, V. Muñoz, Atom-by-atom analysis of global downhill protein folding, Nature. 442 (2006) 317-321.

[10] C. Clementi, Coarse-grained models of protein folding: toy models or predictive tools?, Current Opinion in Structural Biology. 18 (2008) 10-15.

[11] H.I. Beltrán, A. Rojo-Dominguez, M.E. Sanchez-Gutierrez, P.P. González Perez, Exploring Dimensionality, Systemic Mutations and Number of Contacts in Simple HP ab-initio Protein Folding Using a Blackboard-based Agent Platform, World Academy of Science, Engineering and Technology. 52 (2009) 280-289.

[12] M. Mitchell, An Introduction to Genetic Algorithms, The MIT Press, 1998.

[13] D.E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, 1º ed., Addison-Wesley Professional, 1989.

[14] M. Mann, S. Will, R. Backofen, CPSP-tools – Exact and complete algorithms for high-throughput 3D lattice protein studies, 9 (2008) 230.

[15] Cox, G.A., Mortimer-Jones, T.V., Taylor, R.P. Johnston R.L., 2004. Development and Optimization of a novel Genetic Algorithm for Studying Model Protein Folding. Theor Chem Acc (2004) 112: 163–178 DOI 10.1007/s00214-004-0601-4.

[16] Krasnogor N., Hart W.E., Smith J., Pelta D.A., 1999. Proceedings of the 1999 international Genetic and Evolutionary Computation Conference (GECCO99) pp. 1596-1601.

[17] Cervantes, J., Stephens, C.R., 2008. Rank Based Variation Operators for Genetic Algorithms. Proceedings of the 2008 international Genetic and Evolutionary Computation Conference (GECCO08) pp. 905-912.

[18] Cervantes, J., Stephens, C.R., 2009. Limitations of Existing Mutation Rate Heuristics and How a Rank GA Overcomes Them. IEEE Transactions on Evolutionary Computation Vol. 13, No. 2 April 2009 pp. 369-397. DOI=http://doi.acm.org/10.1109/TEVC.2008.927707.

[19] H. Hoos, T. Stützle. Evaluating Las Vegas Algorithms: Pitfalls and Remedies. Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, UAI-98 pages 238-245, Morgan Kaufmann Publishers, San Francisco CA, 1998.

[20] Oliveto, P.S., Lehre P.K., Neumann, F. Theoretical Analysis of Rank Based Mutation – Combining Exploration and Exploitation. Proceedings of the Eleventh Congress on Evolutionary Computation 2009.

[21] Dill, K. A., Polymer Principles and Protein Folding. Protein Science, 1999. 8(6): pp. 1166-1180.