MULTIVALUED PARALLEL RECOMBINATIVE REINFORCEMENT LEARNING: A MULTIVALUED GENETIC ALGORITHM

A. LIKAS

16-98

Preprint no. 16-98/1998

Department of Computer Science
University of Ioannina
451 10 Ioannina, Greece

Multivalued Parallel Recombinative Reinforcement Learning: A Multivalued Genetic Algorithm

Aristidis Likas
Department of Computer Science
University of Ioannina
45110, Ioannina, Greece
e-mail: arly@cs.uoi.gr

Abstract

Parallel Recombinative Reinforcement Learning (PRRL) [6, 7] constitutes a population based technique (genetic algorithm) suitable for function optimization in high dimensional binary domains. We present here an extension to this method that concerns the case of population strings whose elements may take any value from a finite discrete set. To achieve this, a new type of stochastic unit (the *multivalued discrete stochastic unit*) is defined that can be used in reinforcement learning problems, and a REINFORCE update scheme for the adaptable parameters of the unit is presented. By considering a population of optimizers of this kind that are appropriately recombined at each step (in a manner analogous to PRRL), we obtain the *Multivalued Parallel Recombinative Reinforcement Learning (MPRRL)* technique and present experimental results from the application of the technique to some test problems.

1 Introduction

Optimization techniques can be divided into two main categories. *Point based* techniques perform exploration of the problem state space by examining one point at each step and appropriately moving from one point to another. On the other hand, *population based* search techniques consider as current state the locations of *many points* in the state space and derive new points by suitably manipulating the current ones. Such techniques have gained much attention, mainly since they are suitable for parallel implementation.

A degenerate example of population-based search techniques is parallel point-based hill-climbing, where there is a population of point-based hillclimbers (tabu search, simulated annealing, etc) operating in parallel and independently, with no information exchange among them. Their exploration capability is limited by the capabilities of individual optimizers. The main interest in population based techniques is focused on *recombinative* techniques which at each step generate points in the state space by combining information from the current population. The most widely studied optimization procedures of this kind are based on *genetic algorithms* [3] and their hybrids that extend the traditional simple genetic approach by incorporating more sophisticated hillclimbing procedures. The first such hybrid that has been developed was the SIGH algorithm (Stochastic Iterated Genetic Hillclimbing) [1] and many other attempts followed, such as Parallel Recombinative Simulated Annealing (PRSA) [8].

In this spirit, the Parallel Recombinative Reinforcement Learning (PRRL) method [6, 7] has been developed that constitutes a population-based that tackles discrete optimization problems defined on *binary* ($0 or 1$) domains (i.e. on the unit hypercube). It is based on the parallel operation and combination of individual reinforcement learning optimizers. The approach can be considered as an extension of traditional genetic algorithms in the sense that it considers *vectors of probabilities* instead of *bit vectors* as population members. At each step these members are recombined and, in addition, the components of the resulting vectors of probabilities are adjusted according to reinforcement learning rules. The latter update is used as a *local search* technique used to enhance the *global search* capabilities provided by the genetic algorithms. More specifically, among the possible reinforcement updates, the REINFORCE family of algorithms is used that have been proved to follow the stochastic hillclimbing property [10].

A limitation of the PRRL method is that *it works with binary population strings* and, consequently, it requires the specification of the discrete optimization problem in a 0-1 formulation. This is natural for several problems, but there are also many cases where the problem variables assume values from finite discrete sets with more than two elements (for example, in the k-graph coloring problem, each variable may be assigned any of the k colors ($k \geq 3$)). The formulation of multivalued problems as 0-1 problems has two major drawbacks: i) for a problem with N discrete variables, where each variable i assumes k_i discrete values, $\sum_{i=1}^{N} k_i$ binary variables are needed and ii) additional constraints must be imposed so that only one from the k_i binary variables corresponding to the same discrete variable i can take the value one simultaneously.

The above limitation of the PRRL method can be attributed to the fact that there has not been developed any REINFORCE scheme for effectively treating multivalued discrete optimization problems. All previous REINFORCE approaches to discrete optimization problems are confined to the exploration of binary domains using a team of binary stochastic units called Bernoulli units ([9]). In our attempt to maintain the multivalued formulation, we use in this paper a type of unit, that will be called the *multivalued discrete stochastic (MDS) unit*, and present an update scheme for the adaptable parameters of the MDS unit based on the REINFORCE theorem. Using a team of MDS units (in a manner analogous to the case of Bernoulli units [9]), a reinforcement learning optimizer is obtained suitable for treating multivalued problems. Then, following the principles of the PRRL algorithm, we propose the Multivalued PRRL (MPRRL) algorithm that constitutes a population based technique based on the parallel operation and genetic recombination of reinforcement optimizers of the above kind.

The next section describes the MDS unit and a suitable parameter update scheme that belongs to the class of REINFORCE algorithms. The proposed recombinative approach is described in Section 3, whereas experimental results from applying MPRRL to test problems are reported in Section 4. Section 5 summarizes the main conclusions and provides directions for further research.

2 Multivalued discrete reinforcement learning

In the reinforcement learning approach to function optimization [4, 9, 5] the state of the learning system is determined through a probability distribution. At each step, a point in the function space is generated according to the above distribution, and the corresponding function value which is called *reinforcement* is provided to the system. Then, the parameters of the distribution are updated so as to direct the search towards the generated point in case of a high reinforcement value. In the opposite case, the point is made less probable to be sampled again in the upcoming trials. In order to judge whether a point is 'good' or not, a *standard of comparison* must be specified which in most cases is considered as a trace (weighted average) of past reinforcement values (eq. (3)).

Reinforcement learning schemes have been applied to both continuous [4] and binary [9, 5] function optimization problems. They are mainly based to the generate-and-test paradigm, where at each step, first a point is generated and evaluated and then the parameters of search mechanism are adapted according to the fitness of the generated point. In what concerns the exploration of binary domains, the simplest reinforcement scheme considers that the point $y = (y_1, \ldots, y_n)$ ($y_i \in \{0, 1\}$) of the problem state space selected at each step is stochastically generated by a team of n *Bernoulli units*. Each Bernoulli unit i ($i = 1, \ldots, n$) contains an adaptable parameter (weight) w_i and determines the component y_i of the output vector through a Bernoulli selection with probability $p_i = f(w_i)$, where f is a sigmoid function of the form

$$p_i = f(w_i) = 1/(1 + \exp(-w_i)) \tag{1}$$

If r is the evaluation of the generated point y, the parameters w_i of the units are updated using the following REINFORCE update scheme:

$$\Delta w_i = \alpha(r - \overline{r})(y_i - p_i) \tag{2}$$

In the above equation the parameter α is the learning rate and $\overline{r}$ is the reinforcement comparison mentioned previously that is updated at each step as follows:

$$\overline{r}(t) = \gamma\overline{r}(t-1) + (1-\gamma)r(t) \tag{3}$$

with γ being a decay rate positive and less than 1.

To treat multivalued discrete optimization problems we use here an extension of the Bernoulli unit, that will call the Multivalued Discrete Stochastic (MDS) unit. This type of unit provides as output y one-out-of M possible values $\{a_1, \ldots, a_M\}$. An MDS unit is characterized by a parameter vector $w = (w_1, \ldots, w_M)$ where each parameter w_i corresponds to the output value a_i. The relative values of the weights w_i express the favor of the MDS unit towards the selection of the corresponding output a_i. During selection, using the parameter vector w, a probability vector $p = (p_1, \ldots, p_M)$ is computed as follows:

$$p_i = \frac{\exp(w_i/T)}{\sum_{j=1}^{M}\exp(w_j/T)} \tag{4}$$

It is obvious that $\sum_{i=1}^{M} p_i = 1$. By performing selection using the probability vector p, one of the M values a_i is selected as the output y of the MDS unit.

In order to use the MDS unit for discrete optimization, the update scheme of the parameters w_i must be specified. In our approach we have selected a reinforcement update scheme based on the REINFORCE theorem [9, 10]. REINFORCE algorithms constitute an important class of reinforcement learning algorithms. When applied to a team of stochastic units (of any type) a REINFORCE algorithm prescribes that at each step the weights are updated according to the formula:

$$\Delta w_i = \alpha(r - \overline{r})\frac{\partial \ln g}{\partial w_i} \tag{5}$$

where α is the learning rate factor, r is the reinforcement signal delivered by the environment (ie. the function value) and $\overline{r}$ the reinforcement comparison defined previously (eq. (3)). In addition, $g(a, w) = Prob\{y = a|w\}$ is the probability that the output y of the MDS unit will be equal to the value a given that the parameter vector is w. The quantity $\partial \ln g/\partial w_i$ is called *eligibility* of the parameter w_i.

An important result proved in [10] is that for any REINFORCE algorithm the average update in parameter space W lies in a direction for which the expected value of r is increasing, i.e., the algorithm is characterized by the stochastic hillclimbing property. Therefore REINFORCE algorithms can be used to perform stochastic function maximization [9, 10, 5].

To specify a REINFORCE parameter update scheme for the case of MDS units, the form of eligibility must be determined. From the definition of the MDS unit we find that if the probability vector is $p = (p_1, \ldots, p_M)$, then

$$g(a,p) = Prob\{y = a|p\} = \begin{cases} p_1 & \text{if } a = a_1 \\ p_2 & \text{if } a = a_2 \\ . \\ . \\ . \\ p_M & \text{if } a = a_M \end{cases} \tag{6}$$

This naturally leads to the following equation

$$\frac{\partial \ln g(a,p)}{\partial p_i} = \begin{cases} \frac{1}{p_i} & \text{if } a = a_i \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

Moreover,

$$\frac{\partial \ln g}{\partial w_i} = \sum_{k=1}^{M} \frac{\partial \ln g}{\partial p_k} \frac{\partial p_k}{\partial w_i} \tag{8}$$

and from eq. (4)

$$\frac{\partial p_i}{\partial w_k} = \begin{cases} \frac{1}{T} p_i (1 - p_i) & \text{if } k = i \\ -\frac{1}{T} p_i p_k & \text{if } k \neq i \end{cases} \tag{9}$$

Using the above equations we finally find that the *in the case where the selected output is a_k* the parameter update equation suggested by the REINFORCE theorem is

$$\Delta w_i = \begin{cases} \frac{1}{T}\alpha(r - \overline{r}) p_i (1 - p_i) & \text{if } i = k \\ -\frac{1}{T}\alpha(r - \overline{r}) p_i p_k & \text{if } i \neq k \end{cases} \tag{10}$$

It must be noted that, due to the stochastic hillclimbing property, pure REINFORCE update schemes (as the one described above) tend to eventually confine the search to a small region of the state space, ie. they are trapped to local minima. For this reason a simple modification has been suggested [9] that incorporates a decay term $-\delta w_i$ $(0 < \delta < 1)$ in the update equation (10) in order to achieve the *sustained exploration* objective. Adding this term to equation (10) we get the following parameter update equation:

$$\Delta w_i = \begin{cases} \frac{1}{T}\alpha(r - \overline{r}) p_i (1 - p_i) - \delta w_i & \text{if } i = k \\ -\frac{1}{T}\alpha(r - \overline{r}) p_i p_k - \delta w_i & \text{if } i \neq k \end{cases} \tag{11}$$

The above equation is the reinforcement update equation used in the MPRRL algorithm.

Consider now a multivalued discrete optimization problem with n variables X_i $(i = 1, \ldots, n)$, each one assuming values from the discrete set $A_i = \{a_{i1}, \ldots, a_{ik_i}\}$ (where $k_i = |D_i|$). To tackle this problem, we can employ a team of n MDS units, with each unit i having parameter vector $w_i = (w_{i1}, \ldots, w_{ik_i})$ and set of output values $A_i = \{a_{i1}, \ldots, a_{ik_i}\}$.

The algorithm is simple: at every step, each MDS unit i selects the output $y_i \in D_i$ (using the parameter vector w_i) and thus a point $y = (y_1, \ldots, y_n)$ of the problem state space is obtained. The reinforcement signal r delivered to the learning system is the fitness of the point y and, in the sequent, all MDS units update their parameter vectors w_i using eq. (11). The above procedure can be repeated until a termination criterion is satisfied (for example, the maximum number of steps is reached).

3 The MPRRL algorithm

Multivalued Parallel Recombinative Reinforcement Learning (MPRRL) can be considered as a population-based recombinative extension of the MDS reinforcement learning approach to multivalued discrete optimization. It is based on the employment of p population members, where each population member i is an MDS reinforcement learning optimizer and follows the operation principles of the PRRL algorithm [6, 7].

Consider a discrete optimization problem with n variables X_i, each one assuming values from a finite discrete set $A_i = \{A_{i1}, \ldots, A_{ik_i}\}$ (where $k_i = |A_i|$). A point $Y = (y_1, \ldots, y_n)$ of the problem state space results from the assignment of a value $y_i \in A_i$ to each variable X_i $(i = 1, \ldots, n)$.

The MPRRL algorithm considers a population of p MDS reinforcement learning optimizers, where each optimizer i is a team of n MDS units. Let (i, j) denote the $j - th$ MDS unit of population member i. The weight vector of this unit is $w_j^i = (w_{j1}^i, \ldots, w_{jk_j}^i)$. Let also $W_i = (w_1^i, \ldots, w_n^i)$ denote the whole parameter vector of optimizer i. In this sense, we can say that each reinforcement optimizer i is assigned one *population slot* where its parameter vector W_i is kept. At each step, first a reproduction procedure takes place, during which the parameter vectors are recombined and a new generation of vectors is created. Then a sampling procedure follows, where each optimizer selects a point of the state space and, thus, p points $Y_i = (y_{i1}, \ldots, y_{in})$ $(i = 1, \ldots, p)$ (with $y_{ij} \in A_j$) are generated using the MDS selection scheme described in the previous section. The fitness r_i (corresponding function value) of each point Y_i is evaluated and then, a *reinforcement update* of the parameters W_i of each optimizer is performed (using equation (11)), so that the search is guided towards promising regions of the space.

At each generation step, the p new population members are created as follows: for each current member i we decide with probability p_c whether crossover will be applied or not. If the decision is negative, the parameter vector of slot i does not change, otherwise, another member k is randomly selected and crossover is performed between the probability vectors corresponding to the two parents. The recombination operator that we have adopted is a variant of single-point crossover. The new parameter vector W_l is created in the following manner:

- We randomly select the crossover point t $(1 \leq t \leq n-1)$.
- If $t \leq \lceil n/2 \rceil$, then we set $w^l_{jm} = w^k_{jm}$ $(m = 1, \ldots, k_j)$ for $j = 1, \ldots, t$ and $w^l_{jm} = w^i_{jm}$ $(m = 1, \ldots, k_j)$ for $j = t+1, \ldots, n$.
- If $t > \lceil n/2 \rceil$, then we set $w^l_{jm} = w^i_{jm}$ $(m = 1, \ldots, k_j)$ for $j = 1, \ldots, t$ and $w^l_{jm} = w^k_{jm}$ $(m = 1, \ldots, k_j)$ for $j = t+1, \ldots, n$.

According to the above approach, the new vector W_l remains as close as possible to the vector W_i. That child becomes the new parameter vector W_i for slot i. In this way, the characteristics of individual population members are preserved to some extent, thus retarding the decrease of population diversity. It must be noted that the above reproduction scheme is *synchronous*, i.e., all p children can be created simultaneously and independently based on the precedent generation, thus achieving a *high degree of parallelism*.

At the beginning, all the components of the parameter vectors are set equal to zero, i.e., no initial knowledge is provided to the optimization system. In case no decay term is used in the reinforcement update rule, the search process converges, in the sense that all probability vectors tend to be similar to each other and, in addition, their individual components tend to be 1 or 0. This convergence behavior is justified by the use of both the crossover mechanism and the reinforcement update rule. The crossover mechanism destroys *population diversity* and eventually all optimizers search the same region of the function space. On the other hand, the local search performed through the use of reinforcement updates eventually converges to a point of high fitness value. If a decay term is added, the search algorithm does not converge, in the sense that it does not continuously produce the same points. Thus, sustained exploration is achieved, but we still have a lack of *sustained diversity* due to the effects of crossover, since the parameter vectors of population members are relatively close.

3.1 Sustained Diversity

In analogy with the PRRL case, in order to reduce the effects of crossover and avoid *genetic drift* [2], the MPRRL algorithm employs a mechanism based on the notion of *apathy* [1, 6, 7] that has been proved very effective in maintaining population diversity.

According to the latter approach, some population members remain apathetic for some generations in the sense that they cannot be selected for recombination. Apathetic members cannot change their state through crossover, but can be chosen for crossover by other members of the population. To effectively apply this principle, a criterion is needed for a member to become apathetic as a well as a criterion for becoming active again. We have chosen to put a member into apathy whenever it generates a point of the state space yielding a higher fitness value r_i than the best value $r_i^{\max}$ achieved so far by optimizer i. Thus, from the moment the search attains a high fitness region, the optimizer is allowed to explore that region following the reinforcement learning rule without being disrupted by the crossover mechanism. If no better solution is obtained for a specified number of steps, the member is brought back to the active state and crossover is allowed. Thus, an apathy step counter is necessary for each population member.

Following the discussion presented above, the MPRRL algorithm has the following final form.

- Initialize all parameter vectors to zero.
- Set the apathy counter of each population member i to zero.
- Repeatedly generate a new population from the current one until a maximum number of generations is attained. Each new population is created by performing the following steps for each location $i = 1, \ldots, p$:

  1. If member i is in apathy proceed to Step 4.
  2. With probability p_c decide whether crossover will be performed or not. If the decision is negative proceed to Step 4.
  3. Randomly choose a member from the rest of the population following a uniform distribution. Combine the two parents to produce a new parameter vector as described previously. The new vector replaces the current one in location i.
  4. Based on the parameter vector W_i generate a point Y_i of the state space using the MDS approach and evaluate its fitness r_i.
  5. Update the parameters W_i of location i according to the reinforcement learning rule.
  6. If $r_i > r_i^{\max}$ set $r_i^{\max} = r_i$. If the population member i is not in apathy then put it into apathy and proceed to Step 8.
  7. If the member is in apathy increase the value of its counter by 1. If the value of the counter is the maximum allowed put the member back into the active state and set the counter to zero.
  8. Update the value of reinforcement comparison $\overline{r}_i$.

	MPRRL		MPRL		GA	
No. of nodes	*Succ (%)*	*Avg. Steps*	*Succ (%)*	*Avg. Steps*	*Succ (%)*	*Avg. Steps*
30	100	329	100	421	40	1892
60	100	572	100	654	16.7	3512
90	100	1102	100	1467	6.7	
120	100	1543	90	1978	0	

Table 1: Comparative results for the 3-graph coloring problem.

In the case where one seeks suboptimal solutions in short execution time, both the sustained exploration mechanism and the sustained diversity mechanism can become inactive by setting the value of δ (eq. (10)) equal to zero and the maximum number of steps in apathy equal to zero respectively.

4 Experimental results and Comparisons

Experiments have been conducted on several well-known multivalued discrete optimization problems to compare the effectiveness of MPRRL against the traditional genetic algorithm (GA) and the MPRL (multivalued parallel reinforcement learning) approach, which considers a population of independent multivalued reinforcement learning optimizers. It must be noted that the MPRL method is a special case of the MPRRL method where the crossover probability p_c is zero. The MPRL method was implemented in order to study the effect of recombination.

In all experiments the size p of the population was fixed and equal to 100. At first, we conducted experiments concerning the NP-complete 3-graph coloring problem. The method was tested on difficult graph instances, concerning sparse graphs with $2n$ edges, where n is the the number of vertices (graph size). Such kind of graphs contain a very small number of solutions, usually only one which was embedded by construction. The 3-graph coloring problem can be considered as a problem with n discrete variables (each one corresponding to a vertex). Each variable assumes one of three color values. A point in the state space results from the assignment of a color value to each vertex variable, and the fitness value of this point is the negative of the number of graph edges having vertices with the same color. It is obvious that an acceptable solution has zero fitness (global maximum).

For each graph size, 30 experiments were performed using each technique. The maximum allowed number of generations was 5000. The values of the parameters of the MPRRL and MPRL methods were $\alpha = 0.1$, $\delta = 0.002$ and $T = 1.0$. Moreover, in the MPRRL case the maximum number of apathetic steps was 100 and the crossover probability $p_c = 0.6$.

The parameters of the GA were: crossover probability $p_c = 0.6$ and mutation probability $p_m = 0.01$. Average values concerning the percentage of runs that provided a problem solution and the average number of generation steps required to find the solution are displayed Table 1. It is clear that the MPRRL method is much more effective compared with the other approaches. It must also be noted that the MPRL approach also yielded encouraging results, although of lower quality than the MPRRL method.

The second problem we have examined was the graph k-partitioning problem specified as follows: Given a graph $G = (V, E)$ with $|V| = n$, find if there exists a partitioning of this graph into k disjoint subgraphs of almost equal size. This problem can be formulated as a multivalued discrete optimization problem with n variables corresponding to the n graph vertices where each variable assumes one of k values. The assignment of value i ($i = 1, \ldots, k$) to a variable j means that the vertex j is assigned to the corresponding subgraph G_i. The fitness $f(Y)$ of a point $Y = (y_1, \ldots, y_n)$ of the problem state space is computed by first counting the number $C(Y)$ of edges with vertices belonging to different subgraphs and then computing an imbalance measure quantifying the imbalance in the sizes of the k subgraphs:

$$f(Y) = -C(Y) - \kappa \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} (n_i(Y) - n_j(Y))^2 \tag{12}$$

where $n_i(Y)$ is the number of vertices of subgraph i when the candidate solution is Y. The coefficient κ determines the relative importance of the two components. In our experiments the value of κ was 0.001.

We have selected to experiment with the 4-graph partitioning problem using difficult graph instances known as *multilevel graphs* [1]. Multilevel graphs contain sets of vertices (clumps) such that vertices of each set are fully connected among themselves but have only a few connections to the remaining vertices of the graph. Experiments have been conducted with specific multilevel graphs containing 8 or 16 clumps, each clump consisting of 4 or 6 vertices. In particular we have used four types of graphs denoted as 8×4, 16×4, 8×6, 16×6. Fig. 1 presents the 16×4 graph. In all cases, it is possible to produce a partition of the graph into four disjoint subgraphs of equal size. Therefore the global maximum of the fitness function is zero. The main difficulty with multilevel graphs arises with the existence of clumps, since it is difficult to move the clump from one partition to another by transferring one vertex at a time, ie. transfers must be performed in groups.

In analogy with the graph coloring case, for each graph instance 30 experiments were performed using the MPRRL, MPRL and GA techniques. The parameter values in all methods were the same as in the graph coloring experiments. Each method was terminated either when the global maximum was encountered or when the maximum number of generations

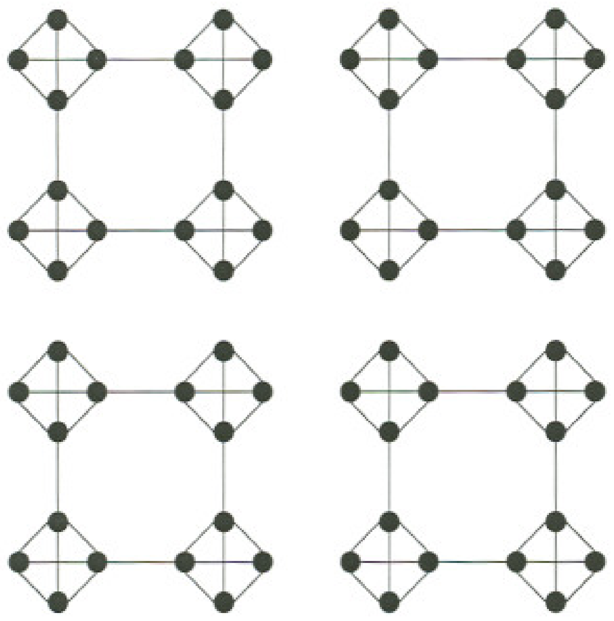

Figure 1: The 16 × 4 multilevel graph.

	MPRRL		MPRL		GA	
Graph	*Succ (%)*	*Avg. Steps*	*Succ (%)*	*Avg. Steps*	*Succ (%)*	*Avg. Steps*
8 × 4	100	640	100	570	20	1892
8 × 6	100	832	100	844	16	3512
16 × 4	100	1853	85	2732	0	
16 × 6	100	3512	70	4210	0	

Table 2: Comparative results for the 4-graph partitioning problem.

was reached. Table 2 displays average values concerning the percentage of runs that found the global maximum as well as the average number of generation steps required to find that solution. As in the graph coloring case, it is clear that the MPRRL method is superior to the other approaches, especially compared to the pure GA technique that completely fail to find acceptable solutions. The MPRL approach also provided acceptable results. This fact suggests that the proposed MDS reinforcement learning optimizer constitutes a interesting method for multivalued discrete optimization. Moreover, the superiority of the MPRRL technique indicates that, in the case where a population of optimizers is employed, it is preferable to recombine them in an appropriate way, instead of having them operate solely and independently.

5 Conclusions

We have introduced Multivalued Parallel Recombinative Reinforcement Learning (MPRRL), which constitutes a genetic technique for tackling multivalued discrete optimization problems. The MPRRL method is based on the use of MDS units described in this work, whose parameters are updated using a rule based on the REINFORCE theorem. Using a team of MDS units we have shown that it is possible to develop a reinforcement learning optimizer for multivalued discrete optimization problems. Going one step further, we have shown that the appropriate recombination of MDS optimizers (ie. the MPRRL method) constitutes a powerful discrete optimization technique.

As experimental results indicate, the MPRRL method is superior to traditional genetic algorithms, and has the ability to effectively treat multivalued problems of high dimensionality. Therefore we aim at testing the effectiveness of MPRRL in dealing with difficult high dimensional real-world problems, where the traditional genetic algorithms have been previously employed (either successfully or not), and we expect to achieve solutions of better quality.

References

[1] Ackley, D. H., *A Connectionist Machine for Genetic Hillclimbing*, Kluwer Academic Publishers, 1987.

[2] Goldberg, D. E. and Segrest, P., *Finite Markov Chain Analysis of Genetic Algorithms*, Genetic Algorithms and their Applications: Proc. of the Second Intern. Conf. on Genetic Algorithms, pp. 1-8, 1987.

[3] Goldberg, D. E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, 1989.

[4] Gullapalli, V., *A Stochastic Reinforcement Learning Algorithm for Learning Real-Valued Functions*, Neural Networks, vol. 3, pp. 671-692, 1990.

[5] Kontoravdis, D., Likas, A. and Stafylopatis, A. *A Reinforcement Learning Algorithm for Networks of Units with Two Stochastic Levels*, Journal of Intelligent Systems, vol. 5, no. 1, pp. 50-76, 1995.

[6] Likas, A., Blekas, K. and Stafylopatis, A., *Parallel Recombinative Reinforcement Learning*, Proc. Int. Conf. on Machine Learning (ICML'95), Heraklion, Greece, April 1995, (Lecture Notes in Artificial Intelligence, Vol. 912, pp. 311-314, Springer Verlag, 1995).

[7] Likas, A., Blekas, K. and Stafylopatis, A., *Parallel Recombinative Reinforcement Learning: A Genetic Algorithm*, Journal of Intelligent Systems, vol. 6, no. 2, pp. 145-169, 1996.

[8] Mahfoud, S. W. and Goldberg, D. E., *Parallel Recombinative Simulated Annealing: A Genetic Algorithm*, Parallel Computing, vol. 21, pp. 1-28, 1995.

[9] Williams, R.J. and Peng, J., *Function Optimization Using Connectionist Reinforcement Learning Networks*, Connection Science, vol. 3, pp. 241-268, 1991.

[10] Williams, R.J., *Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning*, Machine Learning, vol. 8, pp. 229-256, 1992.