Parallel Recombinative Reinforcement Learning

Aristidis Likas, Konstantinos Blekas and Andreas Stafylopatis
Computer Science Division
Department of Electrical and Computer Engineering
National Technical University of Athens
157 73 Zographou, Athens, Greece

Abstract

A technique is presented which is suitable for function optimization in high-dimensional binary domains. The method allows an efficient parallel implementation and is based on the combination of genetic algorithms and reinforcement learning schemes. More specifically, a population of probability vectors is considered, each member corresponding to a reinforcement learning optimizer. Each probability vector represents the adaptable parameters of a team of stochastic units whose binary outputs provide a point of the function state space. At each step of the proposed technique the population members are updated according to a reinforcement learning rule and then recombined in a manner analogous to traditional genetic algorithm operation. Special care is devoted to ensuring the desirable properties of sustained exploration capability and sustained population diversity. The efficiency of the method is tested on two deceptive problems that constitute typical benchmarks yielding very promising results.

1 Introduction

This paper presents a population-based technique for the solution of function optimization problems defined on high-dimensional binary domains. The method does not assume any a priori knowledge regarding the function to be optimized. Information can be collected by generating points in the function domain and observing the corresponding function value. Therefore, the only way to proceed is through a *generate and test* search process, which at each step samples one or more points in the problem state space and based on the received values adjusts its state and/or the sampling strategy.

The majority of search techniques of this kind are *point-based*, in the sense that at each step they consider as current state the position of one point in the function space and they generally select the next point to be evaluated from the neighborhood of the current point. *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 technique is parallel point-based hillclimbing, where there is a population of point-based hillclimbers 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 is focused on *recombinative* population-based techniques which generate points for testing by combining information from the current population. They are characterized by great flexibility concerning the ways in which the current population members can be combined and the strategy under which the generated points will replace the old ones. In addition, they exhibit a high degree of parallelism which makes attractive their implementation on parallel machines.

The most widely studied recombinative population-based optimization procedures are based on genetic algorithms [12, 8, 13]. All other techniques have been developed as 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, mainly aiming at combining features of genetic algorithms and simulated annealing [5, 10, 4]. One promising such technique is Parallel Recombinative Simulated Annealing (PRSA) [15] which seems to improve many of the weaknesses of both genetic algorithms and simulated annealing.

An interesting type of point-based optimizers has been developed from results in the context of reinforcement learning theory applied to connectionist networks [2, 17, 11, 18, 19]. A special class of such algorithms called REINFORCE has been proved to follow the stochastic hillclimbing property [16, 19]. Reinforcement learning algorithms exhibit two desirable properties which are crucial for the effectiveness of any search technique [1]: *learn while searching*, i.e., guide the search towards the more promising regions of the state space, and *sustain exploration*, i.e., gradually switch from local to global search in order to attain other promising regions.

The technique presented in this paper constitutes a population-based method that 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 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 result-

ing vectors of probabilities are adjusted according to the reinforcement learning rules. The resulting scheme exhibits the properties of learning while searching and sustained exploration, since such properties are ensured by the employed reinforcement learning schemes. In addition, since we are dealing with a population-based method, care should be taken to maintain the diversity of the members so that the situation does not reduce to a population of almost identical optimizers simultaneously searching the same region of the space. We call this third desirable property *sustained diversity* and the decisions made in developing our scheme were significantly affected by this objective. We shall denote the proposed population-based approach as *Parallel Recombinative Reinforcement Learning (PRRL)* in analogy with the definition of PRSA.

The next section provides a brief description of genetic algorithms and reinforcement learning schemes. The proposed recombinative approach is described in Section 3, whereas experimental results from applying PRRL to test problems are reported in Section 4. Section 5 summarizes the main conclusions and directions of further research.

2 Genetic Algorithms and Reinforcement Learning

In this section the basic features of traditional genetic algorithms and reinforcement learning algorithms are briefly visited. This short presentation will facilitate the description of the characteristics of the proposed approach.

2.1 Genetic Algorithms

Genetic algorithms in their simple form constitute the first population-based optimization method [12]. There are many variations of the basic approach [8]. In their traditional formulation they assume a population of binary strings and at each generation step new members are created by applying genetic operators to appropriately selected strings.

The most commonly used string selection scheme follows the principle of 'survival of the fittest', i.e., considering maximization problems, strings are selected for reproduction with probability proportional to their corresponding fitness (function value). The crossover operator is responsible for the recombination of the selected strings. Usually two parents are considered for recombination, although even the whole population can participate in the generation of one string, as happens for example in the election scheme of the SIGH algorithm [1]. The two parents can be combined in a variety of ways (sometimes depending on the characteristics of the function to be optimized), the single-point crossover and

the uniform crossover being the most commonly used. In addition, the basic genetic algorithm employs a mutation operator which introduces randomness in the search process by randomly flipping some bit values in the population strings.

Viewed as a population of point-based optimizers, the traditional genetic algorithm performs a kind of *parallel recombinative random search*, in the sense that each population member can be considered as a point-based random optimizer that performs pure random search using the mutation operator. Of course the main search task is accomblished through recombination of these naive optimizers using the crossover operator, but the simplicity of the mutation operator reduces the effectiveness of the algorithm in performing local search.

The main problem with simple genetic algorithms is that they exhibit a fast convergence behavior, mainly due to the effects of the selection scheme which is biased towards strings having high function values and the crossover operator which cannot reintroduce diversity [15]. The crossover between nearly identical strings provides strings similar to their parents. Population diversity can be introduced only through mutation but its effectiveness is rather limited. Therefore, this property of gradual decrease of diversity in the population limits the sustained exploration capabilities of simple genetic algorithms, since it inhibits continuing search which is necessary in solving difficult problems.

2.2 Optimization Using Reinforcement Learning

In the reinforcement learning approach to function optimization [17, 18, 14] 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.

Reinforcement learning schemes have been applied to both continuous [11] and discrete [17, 18, 14] function optimization problems. In what concerns the application to problems defined on binary domains, the simplest scheme considers that the point $y = (y_1, \ldots, y_n)$ $(y_j \in \{0, 1\})$ to be evaluated at each step is generated by a team of *Bernoulli units*. Each Bernoulli unit j determines the component y_j of the output vector through a Bernoulli selection with probability $p_j = f(w_j)$, where $W = (w_1, \ldots, w_n)$ is the vector of adjustable

parameters (weights) and f is a sigmoid function of the form

$$p_j = f(w_j) = 1/(1 + \exp(-w_j)) \tag{1}$$

REINFORCE algorithms [16, 19] constitute an important class of reinforcement learning algorithms. When applied to a team of Bernoulli units a REINFORCE algorithm prescribes that at each step weights are updated according to the formula:

$$\Delta w_j = \alpha_j (r - \overline{r})(y_j - p_j) \tag{2}$$

where α_j is the learning rate factor, r is the reinforcement signal delivered by the environment and $\overline{r}$ the reinforcement comparison:

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

γ is a decay rate positive and less than 1, which in all our experiments was set equal to 0.9.

An important result proved in [16, 19] 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.

Since the above pure REINFORCE schemes converge, a simple modification has been suggested [17] that incorporates a decay term $-\delta w_j$ in the update equation (2) in order to achieve the sustained exploration objective:

$$\Delta w_j = \alpha_j (r - \overline{r})(y_j - p_j) - \delta w_j \tag{4}$$

where $0 < \delta < 1$.

As already stated in the introduction, our scheme is based on manipulating the vectors of probabilities $P_i = (p_{i1} \ldots, p_{in})$ which actually constitute the members of the population. For this reason the weights w_j have been discarded and the necessary updates are performed directly on the probability values, according to the following equation which is derived from (1) and (4):

$$p_{ij} := f\left((1-\delta) \ln \frac{p_{ij}}{1-p_{ij}} + \alpha_{ij}(r_i - \overline{r}_i)(y_{ij} - p_{ij})\right) \tag{5}$$

3 The Recombinative Scheme

Parallel Recombinative Reinforcement Learning can be considered as a population-based extension of the reinforcement learning approach to function optimization. From a different point of view, it can be considered as a *mutation-adaptive* genetic algorithm in the

sense that the mutation probability is adapted as the search proceeds. The mutation probability is not a global parameter having the same value for the whole population, but each component of every string has its individual probability of being 0 or 1.

3.1 The Basic Approach

Consider a PRRL approach with p population members. Each population member i is a vector P_i of probabilities p_{ij} $(i = 1, \ldots, p,\ j = 1, \ldots, n)$ that constitute the state of a reinforcement learning optimizer. In this sense, we can say that each reinforcement optimizer i is assigned one *population slot* where its probability state vector P_i is kept. At each step, first a reproduction procedure takes place during which the probability vectors are recombined and a new generation of vectors is created. Then a sampling procedure is performed and p points $Y_i = (y_{i1}, \ldots, y_{in})$ $(i = 1, \ldots, p)$ of the function space are generated with Bernoulli selection using the corresponding probabilities p_{ij}. The fitness r_i of each point Y_i is evaluated and a reinforcement update of the probabilities takes place using equation (5) 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 state 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 probability vector P_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 $p_{lj} = p_{kj}$ for $j = 1, \ldots, t$ and $p_{lj} = p_{ij}$ for $j = t+1, \ldots, n$.
- If $t > \lceil n/2 \rceil$, then we set $p_{lj} = p_{ij}$ for $j = 1, \ldots, t$ and $p_{lj} = p_{kj}$ for $j = t+1, \ldots, n$.

According to the above approach, the new vector P_l remains as close as possible to the vector P_i. That child becomes the new state vector P_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.

Reproduction is followed by the sampling (based on the new probability vectors) and evaluation phase for the p population members. Then the reinforcement update takes place according to the learning rule of equation (5). There are two main alternative ways for computing the reinforcement comparison:

- Consider a global reinforcement comparison value $\overline{r}$ for the whole population as a weighted average of prior *averaged* reinforcement values, i.e., $r(t)$ in equation (3) is taken equal to $r(t) = 1/p \ \sum_{i=1}^{p} r_i(t)$.

- Consider a separate reinforcement comparison value $\overline{r}_i$ for each population slot, i.e., compute a weighted average of prior reinforcement values r_i concerning optimizer i.

After exprimenting with both approaches, we have adopted the second one. The reason is that the employment of more detailed values for reinforcement comparison leads to better local exploration and better exploitation of the reinforcement updates. Moreover, it makes our algorithm compatible with the non-recombinative approach, in the sense that, if the probability of crossover is set equal to zero, our scheme reduces exactly to the case of a population of independently running reinforcement optimizers. Another reason is that the first alternative implies a tight coupling among population members, which is unfavourable to parallel implementation. Therefore, in addition to the probability vector P_i, the reinforcement comparison value $\overline{r}_i$ is kept for each i and is updated at each step. (We also keep the value r_i^{max} which is the maximum function value that has been sampled at slot i. This is necessary for keeping track of the best value found by the system up to any given instant, as well as for application of the apathy principle as will be shown next.)

At the beginning, all the components of the probability vectors are set equal to 0.5, i.e., no initial knowledge is provided to the learning 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. In this case, the same point is continuously sampled by all population members and the corresponding differences $r_i - \overline{r}_i$ become zero not allowing any further updates of the probabilities p_{ij}. 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 causes the optimizers that search regions of low fitness to be oriented towards regions of higher fitness already explored by other members. In this way, 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.

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 states of the member probabilities are relatively close.

It is clear that the PRRL algorithm as formulated so far is characterized by a high degree of parallelism since all operations can be performed simultaneously for all members of the population.

3.2 Sustained Diversity

In order to reduce the effects of crossover and avoid *genetic drift* [7] we have chosen not to employ a selection and replacement scheme based on the 'survival of the fittest' principle. Instead, a uniform random selection scheme concerning all population members has been applied. Another decision already mentioned was to replace each current population member with the closest of the two generated children, so as to avoid a serious disruption of the states of local optimizers. A third approach that has proved very effective in maintaining population diversity is based on the notion of *apathy* [1].

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 than the best value achieved so far. Thus, from the moment the search attains a high fitness region, the optimizer is allowed to explore that region following the reinforcement learning rule. If for a specified number of steps no better solution is obtained the member is brought back to the active state. Thus, an apathy step counter is necessary for each population member.

Based on the features described above the PRRL algorithm takes the following form.

- Initialize all probabilities to the value 0.5.
- 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 probability vector as described previously. The new vector replaces the current one in location i.
4. Based on the probability vector generate a point of the state space and evaluate its fitness r_i.
5. Update the probabilities of location i according to the reinforcement learning rule.
6. If $r_i > r_i^{\max}$ set $r_i^{\max} = r_i$. If moreover 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 $\overline{r}_i$.

4 Experimental Results

The test problems that were selected for evaluating the effectiveness of our approach are versions of the order-3 deceptive problem that have been commonly employed to test the exploration capabilities of genetic schemes [6, 9].

The order-3 deceptive subfunction is described in Table 1. The problems considered here are the same as in [15] and assume eight subfunctions of the above type, thus the dimension of the binary space is 24. The fitness of each 24-bit string results from the sum of the corresponding values of the eight subfunctions.

The first of the examined problems (called the *tight problem* [15]) is a concatenation of eight subfunctions, where the first three bits of the string are the domain of the first subfunction, the next three bits are the domain of the second subfunction and so on. In the second problem (called the *loose problem*) bits 0, 8, and 16 constitute the domain of the first subfunction, bits 1, 9 and 17 are the domain of the second subfunction and so on. Thus, the bits of each subfunction are maximally separated throughout the string.

x	$f(x)$	x	$f(x)$
000	28	100	14
001	26	101	0
010	22	110	0
011	0	111	30

Table 1: The order-3 deceptive subfunction

The selected problems are useful in order to gain insight into the behaviour of the algorithm when dealing with difficult optimization problem instances, without making the state space very large and the search process intractable. Both problems have 255 local maxima and only one global maximum with value 240. They are called deceptive because (as can be observed from Table 1) each subfunction, apart from the global maximum, has also a local maximum (called deceptive) which is more easy to approach through local hilliclimbing than the global one which is isolated. It is obvious that in the case of the tight problem the crossover operation will be very beneficial, while in the case of the loose problem its contribution will be rather insignificant with stochastic local hillclimbing being the major exploration mechanism.

In all experiments the algorithm was terminated when the global maximum was found or when 5000 generations had been performed. In the latter case the result was the best solution found during the search.

Experiments were carried out for the proposed PRRL method considering different sizes of the population p. For each population size 30 experiments were performed using different seed values for the random number generator. The values of the parameters were $\alpha = 0.05$ and $\delta = 0.02$, whereas the crossover probability was $p_c = 1$ and the maximum allowed value for the apathy counter was set equal to 150.

The same experiments were carried out using a Parallel Reinforcement Learning (PRL) scheme, as well as a simple genetic algorithm (GA). The PRL scheme considers a population of reinforcement learning optimizers that operate independently (without crossover). The GA algorithm considered follows the traditional genetic approach with crossover probability equal to 0.6 and mutation probability equal to 0.1.

The results concerning PRRL and PRL are summarized in Tables 2 and 3 for the tight and the loose problem respectively. The displayed results represent the percentage of cases in which the global maximum was found, as well as the the average number of generation

	PRRL		PRL	
p	*Success (%)*	*Avg. Nb. Steps*	*Success (%)*	*Avg. Nb. Steps*
8	46.6	1900	20.0	
16	80.0	1019	46.6	1133
32	100.0	640	93.3	1097
64	100.0	530	100.0	1143

Table 2: Comparative results for the 'tight' problem

	PRRL		PRL	
p	*Success (%)*	*Avg. Nb. Steps*	*Success (%)*	*Avg. Nb. Steps*
8	33.3	2746	28.0	3287
16	62.0	1896	49.3	3010
32	88.3	1465	75.0	2559
64	100.0	1249	93.3	1973

Table 3: Comparative results for the 'loose' problem

steps required to find it. The superiority of PRRL is apparent in both the success rate and the required number of generations. It should be observed that PRRL performs better on the tight problem than on the loose one. This is due to the effect of single-point crossover which is well adapted to the structure of the tight problem. It is possible that uniform or multi-point crossover would be more appropriate for the loose problem.

Results are not reported for the simple GA, because in very few cases was it successful in finding the optimum within the 5000 generations. This is not surprising since it is known that the solution of deceptive problems is a very hard task for simple genetic algorithms [6, 9, 15].

Our results show that the proposed technique offers clear advantage over conventional genetic algorithms and parallel reinforcement learning. This illustrates the improvement due to the combined use of features such as recombination, reinforcement learning, sustained exploration and sustained diversity, which lead to a more effective exploration of the state space.

5 Conclusions

We have introduced a recombinative technique for function optimization on high-dimensional binary domains. The method borrows from genetic algorithms and reinforcement learning and aims to retain useful characteristics of both approaches. Indeed, the benefits from recombination are enhanced by the exploration capabilities of reinforcement learning algorithms. The technique was tested on benchmark problems and exhibited very good results in comparison to both simple genetic algorithms and non-recombinative reinforcement learning. As the method is directly amenable to parallel implementation our current research concerns the investigation of its performance on parallel hardware considering other optimization problems of large size.

References

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

[2] Barto, A.G. and Anandan, P., *Pattern Recognizing Stochastic Learning Automata*, IEEE Transactions on Systems, Man and Cybernetics, vol. 15, pp. 360-375, 1985.

[3] Barto, A.G. and Jordan, M.I., *Gradient Following without Back-propagation in Layered Networks*, In M. Caudill and C. Butler (eds), Proceedings of the IEEE First Annual Conference on Neural Networks, vol. II, pp. 629-636, San Diego, 1987.

[4] Boseniuk, T. and Ebeling, W., *Boltzmann-, Darwin- and Haeckel-strategies in optimization problems*, Lecture Notes in Computer Science: Parallel Problem Solving from Nature, vol. 496, pp. 430-444, 1991.

[5] Brown, D.E., Huntley, C.L. and Spillane, A.R., *A Parallel Genetic Heuristic for the Quadratic Assignment Problem*, Proc. Third Intern. Conf. on Genetic Algorithms, pp. 406-415, 1989.

[6] Goldberg, D. E., *Simple Genetic Algorithms and the Minimal Deceptive Problem*, Genetic Algorithms and Simulated Annealing, L. Davis (ed), pp. 74-88, Pitman, London, 1987.

[7] 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.

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

[9] Goldberg, D. E., *Messy Genetic Algorithms: motivation, analysis and first results*, Complex Systems, vol. 3, pp. 493-530, 1989.

[10] Goldberg, D. E., *A note on Boltzmann Tournament Selection for Genetic Algorithms and Population-oriented Simulated Annealing* Complex Systems, vol. 4, pp. 445-460, 1990.

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

[12] Holland, J. H., *Adaptation in Natural and Artificial Systems*, Ann Arbor: University of Michigan Press, 1975.

[13] Holland, J. H., *Adaptation in Natural and Artificial Systems*, MIT Press, Cambridge, 1992.

[14] Kontoravdis, D., Likas, A. and Stafylopatis, A. *A Reinforcement Learning Algorithm for Networks of Units with Two Stochastic Levels*, Proceedings ICANN-92, vol. I, pp. 143-146, Brighton, United Kingdom, 1992.

[15] Mahfoud, S. W. and Goldberg, D. E., *Parallel Recombinative Simulated Annealing: A Genetic Algorithm*, IlliGAL Tech. Report No. 92002, Univ. of Illinois at Urbana-Champaign, July 1993.

[16] Williams, R.J., *Toward a Theory of Reinforcement Learning Connectionist Systems*, Technical Report NU-CCS-88-3, Boston, MA, 1988.

[17] Williams, R.J. and Peng, J., *Reinforcement Learning Algorithms as Function Optimizers*, Proceedings of the International Joint Conference on Neural Networks, Washington DC, vol. II, pp. 89-95, 1989.

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

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