-Π 1-

Parallel Recombinative Reinforcement Learning: A Genetic Approach

A. Likas, K. Blekas and A. Stafylopatis

Department of Electrical and Computer Engineering
National Technical University of Athens
157 73 Zographou, Athens, Greece

Abstract

A technique is presented that 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 method has been tested on the graph partitioning problem in comparison with other techniques under two different types of fitness evaluation yielding very promising results.

Keywords

Genetic algorithms, reinforcement learning, hybrid technique, optimization, graph partitioning, parallel implementation.

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 neighbourhood 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 because they are suitable for parallel implementation.

A degenerate example of population-based search technique is parallel point-based hill-climbing, where there is a population of point-based hill-climbers 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 their implementation on parallel machines attractive.

The most widely studied recombinative population-based optimization procedures are various types of genetic algorithms (Goldberg, 1989; Holland, 1975; 1992). An early algorithm that employed a Boltzmann-machine-like structure of both genetic and hillclimbing principles was the SIGH (Stochastic Iterated Genetic Hillclimbing) (Ackley, 1987). Most other techniques have been developed as hybrids that extend the traditional simple genetic approach by incorporating more sophisticated hillclimbing procedures. Many attempts aim at combining features of genetic algorithms and simulated annealing (Boseniuk *et al.*, 1991; Brown *et al.*, 1989; Goldberg, 1990). One promising such technique is Parallel Recombinative Simulated Annealing (PRSA) (Mahfoud and Goldberg, 1995) which seems

to improve many of the weaknesses of both genetic algorithms and simulated annealing. Another approach incorporating the idea of simulated annealing into genetic algorithms is described in (Yip and Pao, 1994; 1995).

An interesting type of point-based optimizers has been developed from results in the context of reinforcement learning theory applied to connectionist networks (Barto and Anandan, 1985; Gullapalli, 1990; Williams and Peng, 1989; 1991; Williams, 1992). A special class of such algorithms, called REINFORCE algorithms, has been proved to follow the stochastic hillclimbing property (Williams, 1988; 1992). Reinforcement learning algorithms exhibit two desirable properties which are crucial for the effectiveness of any search technique (Ackley, 1987): *learning while searching*, i.e., guiding the search towards the more promising regions of the state space, and *sustained exploration*, i.e., gradually switching 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 resulting 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 scheme. 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. Finally 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 (Holland, 1975). There are many variations of the basic approach (Davis, 1991; Goldberg, 1989; Michalewicz, 1994). In their traditional formulation, a population of binary strings is assumed 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., 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. 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.

The main problem with simple genetic algorithms is that they exhibit a fast convergence behaviour, 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 (Mahfoud and Goldberg, 1995). 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. This problem of premature convergence has engaged the attention of many researchers (Whitley, 1989; Mahfoud, 1994; Oei *et al.*, 1991), and some techniques have been developed which allow the formation and maintenance of stable sub-populations. Indeed, the 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 for solving difficult problems.

2.2. Optimization Using Reinforcement Learning

In the reinforcement learning approach to function optimization (Kontoravdis *et al.*, 1992; 1995; Williams and Peng, 1989; 1991), 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 (Gullapalli, 1990) and discrete (Kontoravdis *et al.*, 1992; 1995; Williams and Peng, 1989; 1991) function optimization problems. In what concerns the application to problems defined on binary domains, the simplest scheme considers that the point $y = (y_1,...,y_n)$ $(y_j \in \{0,1\})$ to be evaluated at each step is generated by a team of *Bernoulli units* according to a vector $P = (p_1,...,p_n)$ of probabilities. 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 $W = (w_1,...,w_n)$ 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 (Williams, 1988; Williams and Peng, 1991) constitute an important class of reinforcement learning schemes. When applied to a team of Bernoulli units, a REINFORCE algorithm prescribes that at each step weights are updated according to the formula:

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

where α_j is the learning rate factor, r is the reinforcement signal delivered by the environment at each iteration step t and $\bar{r}$ is the reinforcement comparison:

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

The parameter 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 (Williams, 1988) and (Williams, 1992) 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 (Williams and Peng, 1989) that incorporates a decay term $-\delta w_j$ in the update Eq. (2) in order to achieve the sustained exploration objective:

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

where $0 < \delta < 1$.

As already stated, our scheme is based on manipulating the vectors of probabilities $P_i = (p_{i1},...,p_{in})$, $i = 1,...,p$, 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 assignment operation which is derived from (1) and (4) (inserting the subscript i):

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

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 entire population, but each component of every string has its individual probability of being 0 or 1.

3.1. The Basic Approach

Consider p population members, each member i being a vector P_i of probabilities p_{ij} ($i = 1,...,p$, $j = 1,...,n$) that constitute the state of a reinforce-

ment learning optimizer. We can say that each reinforcement optimizer i is assigned one *population slot* where its probability state vector P_i is kept.

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. At each step, first a reproduction procedure takes place during which the probability vectors are recombined and a new generation of vectors is created. 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 Eq. (5).

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 value of the crossover probability p_c must generally be high (close to 1.0) so as to reinforce the genetic hypostasis of PRLL, rather than to switch the entire approach to a situation of independent reinforcement optimizers (p_c close to zero).

The recombination operator that we have adopted is a variant of single-point crossover. The new probability vector P is created in the following manner:

- We randomly select the crossover point t ($1 \leq t \leq n-1$).
- If $t \leq \lfloor n/2 \rfloor$, then we set $p_j = p_{kj}$ for $j = 1,...,t$ and $p_j = p_{ij}$ for $j = t+1,...,n$.
- If $t > \lfloor n/2 \rfloor$, then we set $p_j = p_{ij}$ for $j = 1,...,t$ and $p_j = p_{kj}$ for $j = t+1,...,n$.

According to the above approach, the new vector P remains as close as possible to the vector P_i. This 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.

In implementing the above scheme we may use either uniform selection or selection based on fitness value. In the latter case, members with high fitness value have more chances to be selected and recombined in the next generation, thus favouring convergence. Due to the synchronous scheme of

reproduction, however, even the 'bad' members of the population will take part in the selection operation and, thus, population diversity is maintained. Therefore, we adopted selection based on fitness value, which has proved more efficient through experimentation.

Then a sampling procedure is performed and p points $Y_i = (y_{i1},...,y_{in})$ 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 Eq. (5) so that the search is guided towards promising regions of the space. There are two main alternative ways for computing the reinforcement comparison:

- Consider a global reinforcement comparison value $\bar{r}$ for the entire population, which is computed as a weighted average of prior *average* reinforcement values taken over the p population members, i.e., $r(t)$ in Eq. (3) is taken equal to $r(t) = 1/p \sum_{i=1}^{p} r_i(t)$.
- Consider a separate reinforcement comparison value $\bar{r}_i$ for each population slot, i.e., compute a weighted average of prior reinforcement value r_i concerning optimizer i.

After experimenting with both approaches, we have adopted the second one. The main reason is that the employment of more detailed values for reinforcement comparison leads to better local exploration and better exploitation of the reinforcement updates. Another reason is that the first alternative implies a tight coupling among population members, which is unfavourable to parallel implementation. Moreover, the second choice makes our algorithm compatible with the non-recombinative approach, since, 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. Therefore, in addition to the probability vector P_i, the reinforcement comparison value $\bar{r}_i$ 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 described next.)

If no decay term is used in the reinforcement update rule, as in Eq. (2), the search process converges, i.e., 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 $r_i - \bar{r}_i$ become zero not allowing any further updates of the probabilities p_{ij}. This convergence behaviour is due to 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 to the reinforcement update rule, as in Eq. (4), 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.

The PRRL algorithm as formulated so far is given in Figure 1. It is clear that the algorithm is characterized by a high degree of parallelism since all

- *Initialize strings with all probabilities equal to 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. *With probability* p_c *decide whether crossover will be performed or not. If the decision is negative proceed to Step 3.*
  2. *Randomly choose a member from the rest of the population. Combine the two parents to produce a new probabiliiy vector as described in the text. The new vector replaces the current one in location* i.
  3. *Based on the probability vector* P_i *generate a point of the state space and evaluate its fitness* r_i.
  4. *Update the probabilities of location* i *according to the reinforcement learning rule.*
  5. *If* $r_i > r_i^{\max}$ *set* $r_i^{\max} = r_i$.
  6. *Update the value of reinforcement comparison* $\bar{r}_i$.

Fig. 1: The basic PRRL algorithm

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* (Goldberg and Segrest, 1987) we have chosen, as already mentioned, to replace each current population member by a child that is closer to it than to its mate, so as to avoid a serious disruption of the states of local optimizers. A second technique that has been proved very effective in maintaining population diversity is based on the notion of *apathy* (Ackley, 1987).

According to the latter approach, some population members remain apathetic for some generations, which means 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 apply this principle effectively, a criterion is needed for a member to become apathetic as 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.

Another possibility is to apply a mutation operator, which can be considered as an extension of the classical genetic operator. After creation of a new generation, this operator can be applied to each individual probability p_{ij}, which means that with a given probability p_i^m the value p_{ij} will be replaced by $1-p_{ij}$. As is generally the case with genetic algorithms, to sustain diversity the mutation probability should vary dynamically following the convergent or divergent behaviour of each population member. To achieve this we varied the mutation probability p_i^m depending on the quantity $s_i = |r_i - \bar{r}_i|$. Indeed, a small value of s_i implies that the probabilities p_{ij} do not change enough at each step so that the optimizer i converges to a narrow region of the state space. To obtain a smooth variation of the mutation probability we have considered that $p_i^m = 1/(1+\exp(\bar{s}_i))$, where $\bar{s}_i$ is a weighted average of previous values of s_i. This approach, however, has not

proved very effective, as it was difficult to tune the effect of mutation so as not to disrupt the crossover operation. Therefore, we have not included this feature in the improved version of the PRRL algorithm.

The operator of apathy described previously proposes a good exploration of a region of fitness function. Another operator that has proved very efficient in maintaining population diversity is the *inversion* operator. Inversion can be seen as a special case of mutation applied to the entire probability vector, and not to each component independently, so that every p_{ij} is replaced by $1-p_{ij}$.

The mechanism of inversion should not be enabled in the case where the optimizer explores locally a region of high fitness value, since this could change its components dramatically and could lead the genetic search to another region of lower fitness. For this reason inversion cannot be applied in apathetic situations, but only when an optimizer has already explored a fitness region and the decay term of the reinforcement learning rule moves the search algorithm into another region of the space. An efficient criterion for applying the inversion operator to an optimizer i is based on the number ζ_i of its components j that have converged and are entering a divergence phase, i.e., that satisfy the following condition:

$$|\delta \ln \frac{p_{ij}}{1-p_{ij}}| > |\alpha_{ij}(r_i - \bar{r}_i)(y_{ij} - p_{ij})| \qquad (6)$$

The above inequality means that the decay term is becoming dominant in Eq. (5). Thus, when a non-apathetic population member has a large value of ζ, e.g., such that $\zeta_i > 0.75n$, where n is the length of the string, the components of the optimizer i are inverted. After that, the inversion operator is not allowed to be applied again to the same member for a specific number of generations, and so a new counter (in addition to the apathy counter) must be kept for each population member.

Based on these additional features the PRRL algorithm now takes the form given in Figure 2.

4. Application to Graph Partitioning

The test problem that was selected to evaluate the effectiveness of our approach is the graph bipartitioning problem: Given a graph $G = (V,E)$ with $|V| = M$ and adjacency matrix A, find if there exists a partitioning of this

- *Initialize strings with all probabilities equal to 0.5 and the two counters 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. Combine the two parents to produce a new probability vector. The new vector replaces the current one in location* i*.*
  4. *Based on the probability vector* P_i *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 apathy 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. *If inversion is disabled for member* i *increase the value of its inversion counter by 1. If the value of the counter is the maximum allowed then enable inversion and set the counter to zero.*
  9. *If member* i *has value* $\zeta_i > 0.75n$ *and it is not in apathy and inversion is enabled, then apply the inversion operator and then disable inversion.*
  10. *Update the value of reinforcement comparison* $\bar{r}_i$*.*

Fig. 2: The improved PRRL algorithm

graph into two disjoint subgraphs of equal size (M even). We shall say that two subgraphs are disjoint if their corresponding sets of vertices are disjoint and there are no edges of the original graph G that connect vertices belonging to the two subgraphs.

Consider an arbitrary partitioning of the original graph G into two subgraphs $G_A = (V_A, E_A)$ and $G_B = (V_B, E_B)$. We shall consider two different

representations of the partitioning that correspond to two different ways of defining the fitness function.

4.1. The Direct Approach

In a first approach, that we shall call the *direct approach*, the partitioning can be represented by a binary vector $\vec{y} = (y_1,...,y_M)$. This vector describes the state of the system where the value of y_i describes the state of the system where the value of y_i indicates whether node i belongs to subgraph G_A or G_B, i.e., if $y_i = 0$ node i belongs to G_A, otherwise it belongs to G_B. The number of nodes of subgraph G_A is $n_A(\vec{y}) = \sum_{i=1}^{M}(1-y_i)$, while the number of nodes of G_B is $n_B(\vec{y}) = \sum_{i=1}^{M} y_i$. The solution of the problem can be expressed as finding the maximum of a function having two components. The first component is responsible for the minimization of the number of edges between the two subgraphs, while the second one balances the sizes of the two subgraphs. Hence, the fitness function to be maximized takes the following form:

$$f(\vec{y}) = -\sum_{i=1}^{M}\sum_{\substack{j=1 \\ j\neq i}}^{M} a_{ij}(1-y_j)y_i - \kappa(n_A(\vec{y}) - n_B(\vec{y}))^2 \qquad (7)$$

where κ is a coefficient which determines the relative importance of the two components. In our experiments an effective value of κ was 0.001. It can be noted that the maximum value of the above function is zero.

The above direct approach is based on the principle of encoding the constraints of the problem into the fitness function in the form of a penalty term that takes its optimum value in the case of feasible states. The performance of this type of approach is usually affected by the increased complexity of the fitness function and by the existence of higher-order correlation between variables due to constraints. In general, such dependencies introduce difficulty in the exploration of the state space and constitute the principal cause of convergence to local optima. For the above reasons, the direct approach represents a hard optimization problem that constitutes a good test for the effectiveness of our method. Indeed, our aim is to evaluate the proposed technique in comparison with other methods rather than to find the best solution to the specific graph partitioning problem. Therefore, we experimented with the direct approach, as well as with the next described approach that provides a more effective formulation.

4.2. The Post-Processing Approach

An approach that can remedy the difficulties discussed above is based on the idea of applying a post-processing operation to the solution vectors produced at each step of the algorithm. In this case, only part (possibly none) of the constraints are incorporated into the fitness function, so that the search scheme is responsible for optimizing mainly the cost part of the problem. Post-processing of generated solutions aims at 'amending' violations of constraints. Amendment is performed by mapping each generated state to a feasible state satisfying the entire set of constraints. The evaluation of this new state is used as evaluation of the original state (Figure 3). This post-processing operation requires a constraint satisfaction scheme that can map arbitrary states to feasible ones. Such a deterministic constraint satisfaction algorithm having polynomial complexity is presented by Likas *et al.* (1995), where it is combined with a reinforcement learning algorithm acting as a general search procedure. An analogous idea has been considered by Liepins and Potter (1991) in the context of genetic algorithms and is referred to as 'repairing' of chromosomes in (Orvosh and Davis, 1994).

The second approach considered for the solution of the graph bipartitioning problem relies on a representation based on the idea of post-processing and incorporates the constraint satisfaction scheme developed in (Likas *et al.*, 1995).

The operation of the constraint satisfaction scheme is based on a general 0-1 integer programming formulation of the constraints (either the entire set or part of it) that must be satisfied. This formulation must be appropriately defined to express a given problem in terms of a state vector $\vec{v}$ of binary variables. A basic assumption concerning the scheme is that only variables that are 'on' may be responsible for the violation of some constraints. In this sense, the set of problem constraints that must be satisfied contains tuples of

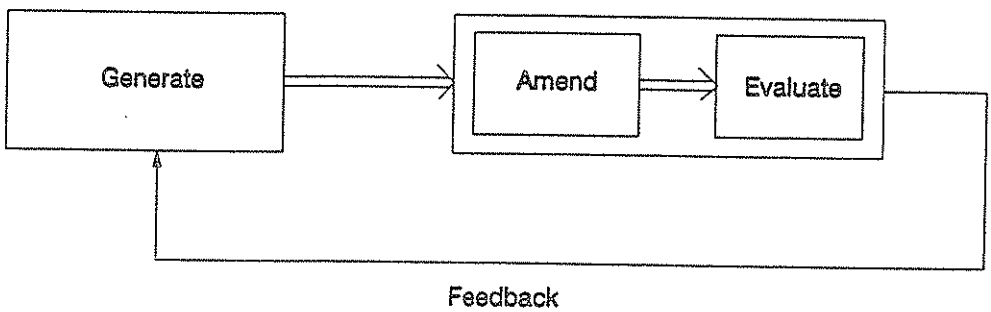

Fig. 3: The post-processing loop

binary variables that are incompatible with each other if they are 'on' simultaneously. This assumption constitutes no restriction to the generality of the approach, as any type of constraint is amenable to this encoding. The algorithm consists of examining the variables v_k in a deterministic order and appropriately adjusting their values depending on constraint violation. It is shown in (Likas *et al.*, 1995) that the algorithm converges to an equilibrium state in at most two cycles, a cycle denoting the sequential examination of all variables. Equilibrium states are characterized by the property that no constraint is violated and that the solution is maximal, which means that every variable that is 0 would violate at least one constraint if it were set to 1.

To apply this approach to the graph bipartitioning problem, we consider that each optimizer i provides an output vector $\vec{y}$ as defined in the direct approach. At each step, the output vector is fed to the constraint satisfaction that maps $\vec{y}$ to an 'amended' representation whose evaluation is taken as the evaluation of $\vec{y}$.

The representation used by the constraint satisfaction scheme is slightly different. The partitioning is expressed by a binary vector $\vec{v}$, such that each node k of graph G is associated with two binary variables v_{k0} and v_{k1}, where $v_{k0} = 1(0)$ means that $k \in V_A (k \notin V_A)$ and $v_{k1} = 1(0)$ means that $k \in V_B (k \notin V_B)$. The number of vertices of G_A is $n_A(\vec{v}) = \sum_{k=1}^{M} v_{k0}$ and the number of vertices of G_B is $n_B(\vec{v}) = \sum_{k=1}^{M} v_{k1}$. The above formulation requiring $2M$ variables is necessary for application of the constraint satisfaction scheme. Indeed, since constraints are violated only by variables with value 1, the representation using M variables (as for the vector $\vec{y}$) is not adequate in this case.

Using the new representation, the problem can be stated as that of finding a vector $\vec{v}$ satisfying the constraints:

$$\sum_{l=0}^{1} v_{kl} = 1, \quad k = 1, \ldots, M \tag{8}$$

$$n_A(\vec{v}) = n_B(\vec{v}) \tag{9}$$

$$v_{k0} \sum_{\substack{i=1 \\ i \neq k}}^{M} a_{ik} v_{i1} + v_{k1} \sum_{\substack{i=1 \\ i \neq k}}^{M} a_{ik} v_{i0} = 0, \quad k = 1, \ldots, M. \tag{10}$$

The vector $\vec{y}$ obtained as the output of the reinforcement learning operation of each optimizer i specifies the initial value of vector $\vec{v}$ in the

following manner. For each $k = 1,...,M$, if $y_k = 0$ we set $v_{k0} = 1$ and $v_{k1} = 0$, otherwise we set $v_{k0} = 0$ and $v_{k1} = 1$. The constraint satisfaction scheme will be responsible for satisfaction of the disjointness constraint (Eq. (10)), as well as of the constraint $\sum_{l=0}^{1} v_{kl} \leq 1$, which constitutes a relaxation of Eq. (8). The remaining requirements are incorporated in the fitness function.

The constraint satisfaction algorithm converges to an equilibrium state $\vec{v}$ that corresponds to a partially feasible problem solution, in the sense that it provides a maximal subgraph G' of G that can be divided into two disjoint subgraphs G'_A and G'_B. This means that the third constraint (Eq. (10)) is satisfied. We say that G' is maximal in the sense that no other node can be added to either G'_A or G'_B because the two subgraphs will not remain disjoint.

The constraint satisfaction scheme does not guarantee satisfaction of the second constraint (Eq. (9)), or of the first constraint (Eq. (8)) (since for some k we may have that $v_{k0} = v_{k1} = 0$, i.e., node $k \notin G'$). Satisfaction of these constraints is achieved by the reinforcement learning part through evaluation of the fitness function. If we let $K(\vec{v}) = \sum_{k=1}^{M} (v_{k0} + v_{k1})$, then the evaluation of the output $\vec{y}$ of the reinforcement learning part is performed by sending a reinforcement signal r given by

$$(K(\vec{v}) - M) - \kappa(n_A(\vec{v}) - n_B(\vec{v}))^2 \qquad (11)$$

where the parameter κ in this case was taken equal to 0.005. It is obvious that the value corresponding to a feasible problem solution is equal to zero and constitutes the maximum value of the fitness function.

4.3. Experimental Results

In our experiments we have considered *multilevel graphs* (Ackley, 1987) having a particular hierarchical structure. Multilevel graphs contain sets of nodes (clumps), such that the nodes of each set are fully connected among themselves but have only a few connections to the remaining nodes of the graph. Experiments have been conducted with specific multilevel graphs containing 8, 16 or 32 clumps, each clump consisting of 4 or 6 nodes. In particular, we have used five types of graphs which are denoted 8 x 4, 16 x 4, 16 x 6, 32 x 4 and 32 x 6. Figure 4 represents the 8 x 4 multilevel graph.

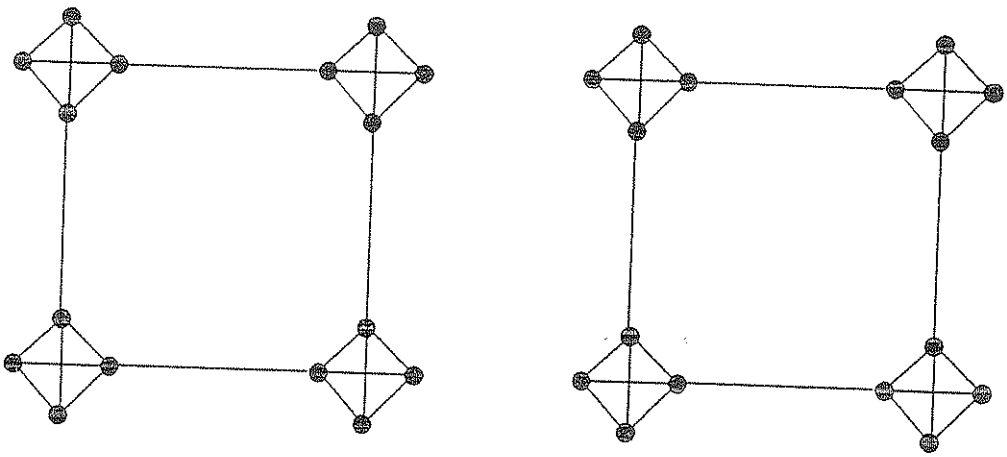

Fig. 4: The 8 × 4 multilevel graph

In all cases, it is possible to produce a partition of the graph into two disjoint subgraphs of equal size. The main difficulty with multilevel graphs arises from the existence of clumps, since it is difficult to move a clump across the partition one node at a time. The selected multilevel graphs are useful in order to gain insight into the behaviour of the algorithm when dealing with difficult problem instances.

Experiments were conducted for the two formulations of the graph partitioning problem presented previously. 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.

All the experiments for the proposed PRRL method were carried out on a Silicon Graphics Power Challenge machine with 14 R8000 processors considering the five multilevel graph instances mentioned above. We have used the random number generator provided by the C library of Silicon Graphics (a multiplicative congruential random number generator with period 2^{32}). For each graph 30 experiments were performed using different seed values for the random number generator. The population size was nearly double the size of each graph. 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 values for the apathy and the inversion counter were both set equal to 150.

Our aim has been to study experimentally the performance of PRRL in comparison with other significant methods, in order to assess its advantage over genetic techniques with enhanced features and over reinforcement learning schemes. For this reason, we have considered three methods for our

experiments: the Parallel Recombinative Simulated Annealing (PRSA), a reinforcement learning scheme and a simple genetic algorithm (GA). Comparison with the last method, which has less enhanced features than the other two, has been included for reasons of completeness.

Combining the benefits of both genetic algorithms and simulated annealing, the PRSA approach (Mahfoud and Goldberg, 1995) generates a population of simulated annealing optimizers using crossover and mutation, as well as a cooling schedule. Starting with an initial value T_s = 14.0, the temperature of each optimizer was decreased every 20 generations with a rate of 4% during a first cooling stage, and with a rate of 1.5% after the temperature had reached a certain value T_x = 4.0 and down to the final temperature T_f = 0.4. The variation of the mutation probability was analogous to the behaviour of cooling. Starting with a large value (0.4), it was decremented every 40 generations until it reached a value of 0.001. The probability of crossover was equal to 1.0.

The reinforcement learning scheme employed was the REINFORCE algorithm described in Section 2.2 (Williams and Peng, 1991; Kontoravdis *et al.*, 1995), expressed by Eqs. (1), (3) and (4). The values selected for the parameters of learning rate and decay term were $\alpha = 0.2$ and $\delta = 0.01$.

The GA algorithm follows the traditional genetic approach (Goldberg, 1989) applying the selection, crossover, and mutation operators to the current population. The selection scheme used is based on the principle of 'survival of the fittest', while the crossover is the single-point crossover: two selected parents are combined producing two new strings with probabilities of crossover and mutation being 0.6 and 0.01 respectively.

The same experiments as in the case of PRRL were carried out using the other three optimization methods. The results are summarized in Tables 1 and 2 respectively for the two formulations considered. The displayed results represent the percentage of cases in which the global maximum was found, as well as the average number of iteration (generation) steps required to find it. The superiority of PRRL is apparent as far as both the success rate and the required number of generations are concerned. As was expected following the earlier discussion, PRRL performs better under the post-processing approach than under the direct approach.

As can be observed in the two tables, in may cases no results are reported for the three methods being compared with PRRL, especially under the

Table 1
Comparative results for the direct approach

	PRRL		PRSA		REINFORCE		GA	
Multilayer Graph	*Success (%)*	*Average steps*	*Success (%)*	*Average steps*	*Success (%)*	*Average steps*	*Success (%)*	*Average steps*
8 × 4	100.0	173	100.0	893	100.0	4925	57.5	1048
16 × 4	100.0	695	83.5	3435	100.0	11650		
16 × 6	100.0	864						
32 × 4	100.0	1109						
32 × 6	52.5	2416						

Table 2
Comparative results for the post-processing approach

	PRRL		PRSA		REINFORCE		GA	
Multilayer Graph	*Success (%)*	*Average steps*	*Success (%)*	*Average steps*	*Success (%)*	*Average steps*	*Success (%)*	*Average steps*
8 × 4	100.0	2	100.0	2	100.0	87	100.0	2
16 × 4	100.0	36	100.0	83	100.0	212	100.0	354
16 × 6	100.0	39	100.0	154	100.0	355	100.0	1017
32 × 4	100.0	71	100.0	292	100.0	453	75.5	2512
32 × 6	100.0	132	100.0	580	100.0	890		

direct approach, because they were not successful in finding the optimum within the 5000 generation limit. This is not surprising since the solution of graph partitioning is a very hard task for simple genetic algorithms and, to some extent, for reinforcement learning. In the case of PRSA it can be seen that, while its results under the post-processing approach are nearly as successful as those of PRRL, its performance under the direct approach is not analogous and PRSA fails to retrieve efficiently the optimal solution. As a matter of fact, the high dimensional graphs are too hard to be partitioned using the direct form of fitness function. On the other hand, it seems that post-processing provides a fitness evaluation mechanism that greatly facilitates the searching task, hence all methods yield results of comparable good quality in a reduced number of steps.

Our results show that the proposed PRRL method offers clear advantages over the techniques that have been used for comparison. 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.

4.4. Parallel Implementation

The PRRL algorithm as described in the previous section features fully parallelizable characteristics, which are well adapted to shared memory architectures. The vectors of each generation can be accessed by a number of processors which are able to apply all the genetic operators (steps 1 to 10) of the algorithm to population members independently and in parallel, thus ensuring a high degree of parallelism. In our parallel code development on the Silicon Graphics Power Challenge machine we have used the Power C language which provides very rich programming facilities.

Two series of experiments were performed for the measurement of speedup using PRRL for the solution of the graph partitioning problem. First we considered the behaviour of speedup for the five graph instances as a function of the number of processors, using up to eight R8000 processors of the Silicon Graphics Power Challenge. The measured speedup values for the 32×6 graph are represented in Figure 5, while the performance was analogous in the case of the remaining graphs. The speedup achieved for the 32×6 graph using 8 processors was 6.2 and 7.1 for the two methods used for the solution of the graph partitioning problem (direct and post-processing approach respectively). As can be observed, the post-processing

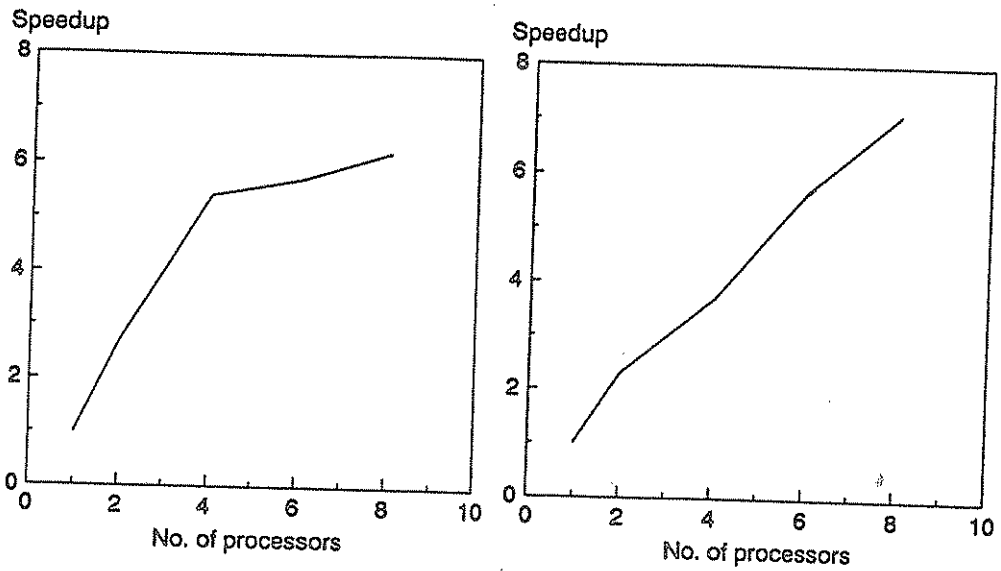

Fig. 5: Speedup for the 32 × 6 graph using (a) the direct and (b) the post-processing approach

approach yields almost linear speedup and attains higher speedup values than the direct approach. This is due to the fact that the computational complexity of the function evaluations performed by each processor is higher in the case of post-processing.

The second set of results concerns the variation of speedup as a function of the size of the graph. In Figure 6 the speedup is displayed for the five multilevel graphs using eight processors for the direct and the post-processing approach. It can be observed that, as the size of the graph grows, speedup also increases obtaining its largest value for the 32 × 6 multilevel graph. Analogous is the behaviour of speedup if less processors are used. Since the processing time grows in analogy to the problem size, parallelism is beneficial for large scale problems. Again, the speedup in the case of the post-processing method is higher than in the direct approach. It can be said, in conclusion, that PRRL exhibits ideal behaviour under parallelism for the post-processing approach and quite satisfactory behaviour for the direct approach.

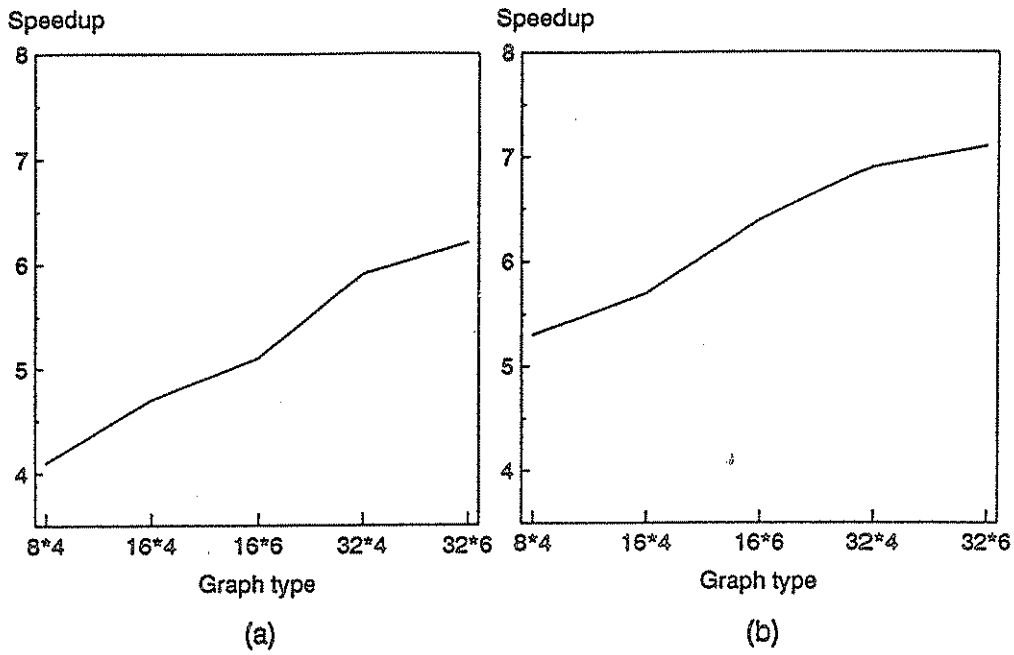

Fig. 6: Speedup using 8 processors for (a) the direct and (b) the post-processing approach

5. Conclusions

We have introduced Parallel Recombinative Reinforcement Learning (PRRL), a genetic 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, while the method is able to avoid the weaknesses of its constituents. Comparison with other methods such as Parallel Recombinative Simulated Annealing, reinforcement learning, as well as a traditional genetic algorithm, has illustrated the superiority of the proposed technique in terms of solution quality. The methods have been tested on the graph bipartitioning problem, which is a hard optimization problem, under two different formulations of the fitness function, a direct formulation containing penalty terms for problem constraints and an approach involving post-processing of generated solutions for constraint satisfaction.

As the PRRL method is directly amenable to parallel implementation, we have investigated its performance on a shared-memory parallel architecture (Silicon Graphics Power Challenge) obtaining very good results in terms of

speedup, which attains linear behaviour. Further work on PRRL includes application to other hard optimization problems of large size and evaluation of the performance obtained through parallel implementation.

As the proposed approach is quite general, another line of research concerns the use of other reinforcement learning algorithms, as well as the incorporation of other techniques for enhancing the exploration capabilities of the method. Also, a topic of theoretical interest concerns the investigation of the effects of recombination on the convergence properties of reinforcement learning algorithms.

References

1. Ackley, D.H. *A Connectionist Machine for Genetic Hillclimbing,* Boston, Kluwer Academic Publishers, 1987.
2. Barto, A.G., Anandan, P. Pattern recognizing stochastic learning automata, *IEEE Transactions on Systems, Man and Cybernetics,* 15, 360-375, 1985.
3. Boseniuk, T., Ebeling, W. Boltzmann-, Darwin- and Haeckel-strategies in optimization problems, *Lecture Notes in Computer Science: Parallel Problem Solving from Nature,* 496, 430-444, 1991.
4. Brown, D.E., Huntley, C.L., Spillane, A.R. A parallel genetic heuristic for the quadratic assignment problem, *Proc. Third Inter. Conf. On Genetic Algorithms,* 1989; pp. 406-415.
5. Davis, L. *Handbook of Genetic Algorithms,* New York, Van Nostrand Reinhold, 1991.
6. Goldberg, D.E., Segrest, P. Finite Markov chain analysis of genetic algorithms, in: *Genetic Algorithms and their Applications: Proc. of the Second Inter. Conf. on genetic Algorithms,* 1987; pp. 1-8.
7. Goldberg, D.E. *Genetic Algorithms in Search, Optimization and Machine Learning,* Reading, MA, Addison-Wesley, 1989.
8. Goldberg, D.E. A note on Boltzmann tournament selection for genetic algorithms and population-oriented simulated annealing, *Complex Systems,* 4, 445-460, 1990.
9. Gullapalli, V. A stochastic reinforcement learning algorithm for learning real-valued functions, *Neural Networks,* 3, 671-692, 1990.
10. Holland, J.H. *Adaptation in Natural and Artificial Systems,* Ann Arbor, University of Michigan Press, 1975.

11. Holland, J.H. *Adaptation in Natural and Artificial Systems,* Cambridge, MIT Press, 1992.
12. Kontoravdis, D., Likas, A., Stafylopatis, A. A reinforcement learning algorithm for networks of units with two stochastic levels, in: *Proc. ICANN-92,* Brighton, United Kingdom, I, 143-146, 1992.
13. Kontoravdis, D., Likas, A., Stafylopatis, A. Enhancing stochasticity in reinforcement learning schemes: Application to the exploration of binary domains, *Journal of Intelligent Systems,* 5 (1), 49-77, 1995.
14. Liepins, G.E., Potter, W.D. A genetic algorithm approach to multiple-fault diagnosis, in: *Handbook of Genetic Algorithms,* Davis, L. (Ed.), New York, Van Nostrand Reinhold, 1991; pp. 227-250.
15. Likas, A., Kontoravdis, D., Stafylopatis, A. Discrete optimization based on the combined use of reinforcement and constraint satisfaction schemes, *Neural Computing and Applications,* 3, 101-112, 1995.
16. Mahfoud, S.W. Genetic drift in sharing methods, in: *Proc. First IEEE Conference on Evolutionary Computation (ICEC'94),* Orlando, Florida, 1, 67-72, 1994.
17. Mahfoud, S.W., Goldberg, D.E. Parallel recombinative simulated annealing: A genetic algorithm, *Parallel Computing,* 21, 1-28, 1995.
18. Michalewicz, Z. *Genetic Algorithms + Data Structures = Evolution Programs,* Springer-Verlag, 1994.
19. Oei, C.K., Goldberg, D.E., Chang, S.J. Tournament selection, niching and the preservation of diversity, *IlliGAL Rep. 91011,* University of Illinois, Urbana, 1991.
20. Orvosh, D., Davis, L. Using a genetic algorithm to optimize problems with feasibility constraints, in: *Proc. First IEEE Conference on Evolutionary Computation,* Orlando, FL, 548-553, 1994.
21. Whitley, D. The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best, in: *Proc. Third Inter. Conf. on Genetic Algorithms,* Los Altos, CA, Morgan Kaufmann, 116-121, 1989.
22. Williams, R.J. Toward a theory of reinforcement learning connectionist systems, *Technical Report NU-CCS-88-3,* Boston, MA, 1988.
23. Williams, R.J., Peng, J. Reinforcement learning algorithms as function optimizers, in: *Proc. Inter. Joint Conf. on Neural Networks,* Washington, DC, II, 89-95, 1989.

24. Williams, R.J., Peng, J. Function optimization using connectionist reinforcement learning networks, *Connection Science*, 3, 241-268, 1991.
25. Williams, R.J. Simple statistical gradient-following algorithms for connectionist reinforcement learning, *Machine Learning*, 8, 229-256.
26. Yip, P.P.C., Pao, Y.-H. A guided evolutionary simulated annealing approach to the quadratic assignment problem, *IEEE Transactions on Systems, Man and Cybernetics*, 24 (9), 1383-1387, 1994.
27. Yip, P.P.C., Pao, Y.-H. Combinatorial optimization with use of guided evolutionary simulated annealing, *IEEE Transactions on Neural Networks*, 6 (2), 290-295, 1995.