Neural Comput & Applic (1995)3:101–112

Neural Computing & Applications

Discrete Optimisation Based on the Combined Use of Reinforcement and Constraint Satisfaction Schemes

A. Likas, D. Kontoravdis and A. Stafylopatis

Department of Electrical and Computer Engineering, National Technical University of Athens, Computer Science Division, Athens, Greece

A new approach is presented for finding near-optimal solutions to discrete optimisation problems that is based on the cooperation of two modules: an optimisation module and a constraint satisfaction module. The optimisation module must be able to search the problem state space through an iterative process of sampling and evaluating the generated samples. To evaluate a generated point, first a constraint satisfaction module is employed to map that point to another one satisfying the problem constraints, and then the cost of the new point is used as the evaluation of the original one. The scheme that we have adopted for testing the effectiveness of the method uses a reinforcement learning algorithm in the optimisation module and a general deterministic constraint satisfaction algorithm in the constraint satisfaction module. Experiments using this scheme for the solution of two optimisation problems indicate that the proposed approach is very effective in providing feasible solutions of acceptable quality.

Keywords: Constraint satisfaction; Discrete optimisation; Graph partitioning; Higher-order Hopfield; Reinforcement learning; Set partitioning

1. Introduction and Motivation

Discrete optimisation problems in their general formulation can be defined in terms of a tuple (S,S', f_c), where S denotes the state space of the problem, $S' \subseteq S$ denotes the set of *feasible* states, i.e. those satisfying the problem constraints, and $f_c : S \rightarrow \Re$ denotes the function that determines the cost of each state. The problem is to find a feasible state for which the cost function is optimal, i.e. to find a state $s' \in S'$ such that $f_c(s')$ is optimal in S' [1].

The conventional approach to tackle these problems is to regard the given optimisation problem as a tuple (S,S,f'), where the function f' has the form $f' = f_p + f_c$, with the function f_p encoding the constraints of the problem and taking its optimum value in the case of feasible states. In this way, the problem is reduced to an unconstrained optimisation one consisting of finding a state $s \in S$ for which the function f' is optimal in S.

Most search techniques applied to the solution of discrete optimisation problems are based on the above formulation which expresses the constraints through the penalty term f_p. In that case, the optimisation procedure can be described as an iteration of a simple *generate-test* loop, as shown in Fig. 1. This approach is unavoidable when no *a priori* information about the function f' is available. Thus, points are generated which belong to the domain of f' and the evaluation of these points

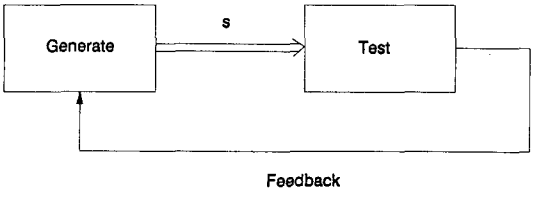

Fig. 1. The generate-test loop.

Received for publication 9 August 1994.

Correspondence and offprint requests to: A. Likas, Department of Electrical and Computer Engineering, National Technical University of Athens, Computer Science Division, 157 73 Zographou, Athens, Greece.

consists of computing the corresponding function value. The aim is to adapt the generator so that 'better' points are generated as the iteration evolves. This adaptation is based on information that is fed back to the generator after the evaluation of each point.

A factor affecting the performance of this kind of approach in finding near-optimal feasible solutions is the increased complexity of the cost function due to the addition of the penalty term. Another important factor concerns the existence of higher-order correlations between the values of variables that are used to determine the problem state. This is a consequence of the existence of constraints which specify values that cannot be assigned simultaneously to groups of variables. In general, such higher order dependencies introduce difficulty in the exploration of the state space, and constitute the principal cause of convergence to local optima. The ability to escape from such local optimum traps requires that a set of correlated variables simultaneously change their value, i.e. jumps between states that are relatively far apart (according to some distance measure) must be possible.

In this paper, we propose a novel approach to the solution of discrete optimisation problems, which is based on the incorporation of a mapping operation within the generate-test loop described above. Through the mapping operation, the output $s \in S$ of the generator is mapped to a feasible state $s' \in S'$. The cost of the point s' provides the evaluation of the generated point s. Thus, instead of performing the search directly in the space of feasible states, the search scheme generates states in S, which are evaluated based on the cost of the transformed states. According to this approach, an optimisation scheme is responsible for optimising only the cost part of the function. Thus, the complexity of the cost function is not increased, and fewer higher-order dependencies have to be discovered. However, a constraint satisfaction scheme is required which should be capable of mapping arbitrary states in S to feasible states in S'.

A basic characteristic of the proposed approach is that the constraint satisfaction scheme operates in an iterative way, thus its input-output behaviour cannot be described by a mathematical formula. Consequently, it is not possible to perform optimisation in a direct manner by using one of the well known function optimisation techniques (e.g. Lagrange or other). Instead, our optimisation scheme must rather be based on an exploration technique.

Depending on the problem, it is not always possible to map an arbitrary state $s \in S$ to a feasible one using a polynomial-time constraint satisfaction scheme. To overcome this difficulty, we can consider an alternative formulation which allows an effective treatment of the constraint satisfaction part of the problem. In this formulation, the original problem (S,S',f_c) is transformed into a new one specified as (S,S'',f''), where $S'' \subseteq S$ denotes a set of states that satisfy part of the problem constraints. We shall refer to those states as *partially feasible* states of S. Moreover, the function f'' has the form $f'' = f'_p + f_c$, with f'_p encoding the remaining constraints of the problem. Our goal is to find a state $s'' \in S''$ for which the function f'' is optimal in S''. In general, for a given problem, there may exist many alternative ways to define the set S''. Obviously, in favourable cases we can have $S'' = S'$, and this method simply corresponds to the original formulation of the problem. The proposed approach involves a constraint satisfaction technique capable of mapping arbitrary states to feasible or partially feasible ones.

In the above context, we have developed an original constraint satisfaction scheme, which is established in terms of interesting theoretical results and constitutes a second contribution of the paper. The principle of the method is equivalent to using higher-order Hopfield networks for the solution of constraint satisfaction problems.

In the next section we provide a general description of the proposed method and prescribe the characteristics of the individual schemes involved. In Sect. 3 an original constraint satisfaction technique is presented which is used as part of the general approach. Section 4 concerns the optimisation part, which is responsible for generating sample states. An original scheme is considered here which falls in the area of reinforcement learning. Implementation and performance issues are discussed in Sect. 5 through the application of the proposed method to well known optimisation problems which yielded very good results. Finally, the main conclusions are discussed in Sect. 5.

2. The Integrated Approach

Consider a discrete optimisation problem (S,S',f_c), where, as defined above, S denotes the state space of the problem (which must be finite), S' denotes the subset of S whose states satisfy a specific set of constraints and f_c is the cost function to be optimised in the domain S'. Also consider the formulation (S,S'',f'') introduced before, which is based on the notion of partially feasible states. In the following we shall use the term 'feasible' to refer to either

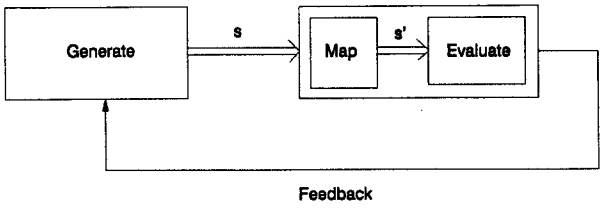

Fig. 2. The sample-map-evaluate loop.

feasible or partially feasible states, as both are treated in the same manner. Accordingly, the notation s' (S') may actually be used in place of s'' (S'') following the case.

The search is not performed directly in space S'. Instead, the optimisation scheme at each step suggests a state $s \in S$, which in turn is evaluated based on the cost of a transformed state $s' \in S'$ provided by the constraint satisfaction scheme (Fig. 2). The whole method, which integrates the constraint satisfaction scheme within the optimisation scheme, is based on the iterative application of the three-step procedure *sample-map-evaluate*, which generates sample points in the state space S, maps these points to points in S', and finally, evaluates them based on the cost of the corresponding feasible points.

The particular way in which the evaluation of a state s is performed, has as a direct consequence that the state space S is partitioned into groups of states. The number of groups is equal to the cardinality $|S'|$ of the feasible state space S'. All states that belong to the same group are mapped to the same feasible state, thus receiving the same evaluation. In general, the constraint satisfaction scheme is characterised by a kind of attraction behaviour, similar to the one exhibited by Hopfield-type neural networks [2]. Therefore, the landscape to be searched consists of 'plateaus', i.e. large flat areas where the function has the same value. To sample a near-optimal feasible state, it is not necessary for the optimisation scheme to generate the point itself. Instead, it suffices to generate an arbitrary state belonging to the same group, thus the exploration task is greatly facilitated. This argument, in conjunction with the already mentioned reduction of the number of higher-order dependencies, provides strong evidence for the success of the proposed method. The constraint satisfaction scheme adopted in our integrated approach is described in the next section. Its operation is based on theoretical results which are of interest by themselves.

In what concerns the optimisation scheme, it must be based on an exploration technique that appropriately samples the state space and evaluates the function at the sample points in an attempt to find an optimal solution. Three optimisation techniques that operate in this way are: simulated annealing [1,3], genetic algorithms [4,5] and reinforcement learning algorithms [6–9].

As far as the use of simulated annealing is concerned, its effectiveness should be affected in our case by the plateau-like shape of the function to be searched. Its application requires the definition of a neighbourhood around each state, from which the next state to be evaluated is drawn. Such a neighbourhood must be relatively small in general [1], thus to escape from large plateaus several steps may be required. This fact on the one hand decreases the speed of the method, and on the other hand, increases the probability of getting stuck on 'plateaus' corresponding to local maxima.

Both reinforcement and genetic algorithms generally possess the required property of suggesting next states that may be far in distance from the current one. Genetic algorithms are based on the evolution of a population, whose members represent sample points of the function to be optimised (fitness function) [5]. If the genetic approach were adopted as our optimisation scheme, then at each generation the constraint satisfaction module should have been called many times for the fitness of each individual population member to be computed. Since all evaluations are independent, they could be efficiently performed on a parallel machine. This implementation constitutes one of the objectives of our future research.

In the approach presented in this paper we have selected to employ a reinforcement learning algorithm as the optimisation scheme. The suitability of reinforcement learning for function optimisation stems from the trial and error nature of the paradigm, which assumes that the only information provided to the learning system arises through sampling the function values at various points of the search space. This characteristic distinguishes reinforcement learning techniques from other neural network approaches to optimisation, such as the analogue Hopfield network or the Boltzmann optimiser, in which prior problem knowledge is incorporated in the network parameters. In Sect. 4 an original reinforcement learning scheme is presented, which has been used in implementing and testing the integrated optimisation approach. For comparison reasons, the pure simulated annealing technique has also been implemented and evaluated experimentally, as will be discussed in Sect. 5.

3. The Constraint Satisfaction Scheme

The operation of the constraint satisfaction scheme is based on a general 0-1 integer programming formulation of the constraints (either the whole set or part of it) that must be satisfied. This formulation must be appropriately defined to express a given problem in terms of a set of binary variables. We say that such a variable is 'on' ('off') if it is equal to 1 (0). Let $v = (v_1, \ldots, v_N)$ denote the vector of binary variables. The value of v at any given time instant represents the state of the system.

A basic assumption concerning our scheme is that only variables that are 'on' may be responsible for the violation of some constraints. In this sense, the set C of problem constraints, which have to be satisfied by the scheme, contains tuples of binary variables that are incompatible with each other if they are 'on' simultaneously. This assumption constitutes no restriction to the generality of the approach. Indeed, if the actual problem variables have finite discrete domains, then a binary variable of the type defined above can be used to represent the assignment of each domain value to each problem variable. This formulation allows the encoding of any type of constraint.

We denote by C_k the set of tuples in which variable v_k is involved. Assuming that we are in a given state v, we associate with each variable v_k an indicator $I_k(v)$. This indicator determines whether a constraint violation involving variable v_k would occur if we set $v_k = 1$. Specifically, $I_k(v) = 1$ when at least one constraint from the set C_k would be violated, otherwise it is zero. Obviously, if variable v_k is already 'on', the indicator I_k characterises constraint violation in the current state.

Using the above definitions the constraint satisfaction problem can be defined as follows:

Find a binary vector $v = (v_1, \ldots, v_N)$ such that

$$\sum_{k=1}^{N} v_k I_k(v) + \sum_{k=1}^{N} (1 - v_k)(1 - I_k(v)) = 0 \,. \tag{1}$$

It is apparent that the above condition is satisfied iff both sums are zero. In what concerns the first sum, this guarantees that no constraint is violated, while making the second sum equal to zero ensures the maximality of the proposed solution, i.e. every variable that is zero would violate at least one constraint if it were set equal to one.

The purpose of the constraint satisfaction scheme is to map (in polynomial time) the state represented by the output of the reinforcement scheme to another one satisfying Eq. (1). This new state will then be evaluated to judge the appropriateness of the generated sample. The mapping is performed using the following iterative algorithm. At each time step, we select a variable v_k and examine whether any constraint of the corresponding set C_k is violated, assuming v_k is set to the value one. If no constraint violation occurs we set $v_k = 1$, otherwise we set $v_k = 0$. By proceeding in this fashion, a state is finally attained from which no other transition is possible (equilibrium state). This is established by the following proposition:

Proposition 1. *If the selection scheme guarantees that every variable is examined infinitely often, then the iterative selection process eventually terminates and an equilibrium state is finally attained which is characterised by the following two properties:* (i) *no constraint is violated; and* (ii) *every variable that is zero would violate at least one constraint in case it were set equal to one (maximality property).*

Proof. We define the following function, which will be called the *energy function*, corresponding to each state vector v

$$E(v) = \sum_{k=1}^{N} v_k (I_k(v) - \alpha) \,, \tag{2}$$

where α is an arbitrary positive parameter with value less than 1.

The above energy function constitutes a *Liapunov* function for our system, in the sense that at each step either it decreases or remains the same. This can be verified as follows.

Let the system be in state v, and let ΔE denote the difference in energy that would result after examination of a variable v_k. The following cases can be distinguished:

v_k changes from 0 to 1. This implies that $I_k(v) = 0$ and, consequently, the I-value of no other variable that is in the 'on' state is affected. Therefore, $\Delta E = -\alpha < 0$.

v_k changes from 1 to 0. This implies that $I_k(v) = 1$ and, in addition, the I-values of some (say $\kappa \geq 0$) 'on' variables may become zero. Thus, $\Delta E = -(1 - \alpha) - \kappa < 0$.

v_k does not change. Obviously $\Delta E = 0$.

Since the energy function is bounded from below and does not increase in any step of the algorithm, it is obvious that an equilibrium state will finally be attained with the characteristic that no change in the value of any variable will be possible. □

The operation of the proposed optimisation method requires that the evaluation of the output

generated by the exploration unit be consistent. This means that identical states should receive the same evaluation. Therefore, the mapping performed by the constraint satisfaction scheme should be *deterministic*, i.e. for the same input state the scheme should always yield the same output. Thus, in the iterative procedure described above, the binary variables are not selected in a random way, but they are sequentially examined following an order, which is specified in advance. An *operation cycle* of the constraint satisfaction scheme consists of the sequential examination of all variables following the predetermined order. To ensure deterministic operation this order remains fixed during all operation cycles.

An interesting result concerning the speed of convergence of the above deterministic operation scheme is given in terms of the following proposition:

Proposition 2. *The number of operation cycles required to converge to an equilibrium state is at most two.*

Proof. According to the operation principles of the scheme, when a variable v_k is considered and takes the value one, no constraint violation occurs as far as the constraints in which this variable is involved are concerned. In addition, all variables that will be considered afterwards and are linked by constraints with the specific variable v_k will be prevented from taking the value one. Therefore, it is not possible that variable v_k will become zero in a subsequent cycle. Consequently, during the second cycle, only variables that are zero at the end of the first cycle may change their state. After the end of the second cycle the variables that have taken the value one cannot change their state in a subsequent cycle for the same reason described above. This also holds for the variables that remained zero after the end of the second cycle. The reason is that any subsequent state change from zero to one would assume that a variable that had value one at the end of the first cycle became zero during the second cycle, which is impossible as explained previously. From the above it is obvious that after the second cycle the following properties are valid: (i) there is no constraint violation between variables that have value one; and (ii) no variable that is zero can assume the value one, because there will be violation of at least one constraint. □

It should be pointed out that the above result does not depend on the existence of a predefined examination order for variables. This order is exclusively due to the necessity of deterministic operation of the constraint satisfaction scheme within the context of the overall optimisation approach, and does not affect the speed of convergence. Considering the constraint satisfaction scheme in isolation, the proposition could be formulated in a more general manner as follows: convergence is achieved in at most $2N$ steps, where N is the number of variables, provided that each variable is examined exactly once during the first N steps, and exactly once during the next N steps.

The operation of the constraint satisfaction scheme can be summarised by means of the following algorithmic steps:

1. {Cycle 1} Following the predetermined order examine each variable v_k as follows:
   (a) Check $I_k(v)$.
   (b) If it is 1 then set $v_k = 0$, otherwise set $v_k = 1$.
2. {Cycle 2} Following the predetermined order examine each variable v_k as follows:
   (a) If $v_k = 1$ continue.
   (b) If $v_k = 0$
      (i) Check $I_k(v)$.
      (ii) If it is 1 then set $v_k = 0$, otherwise set $v_k = 1$.

The constraints may be expressed either in the form of general rules or they may be described explicitly as tuples of the constraint set C. In any case, during the computation of each $I_k(v)$ the checking can stop if a constraint violation is encountered, without needing to examine the whole set of constraints C_k. This fact has significant impact on the execution speed of our algorithm, since it greatly reduces the required computations.

Our constraint satisfaction scheme, which has been described previously in an abstract form, can be implemented in a connectionist manner by means of a discrete higher-order Hopfield network with N binary units. In general, if $v = (v_1, \ldots, v_N)$ denotes the state of a higher-order Hopfield network, its energy function is given by

$$\begin{aligned} E = & -\sum_{i_1=1}^{N}\sum_{i_2=1}^{N} v_{i_1}v_{i_2}w_{i_1i_2} \\ & -\sum_{i_1=1}^{N}\sum_{i_2=1}^{N}\sum_{i_3=1}^{N} v_{i_1}v_{i_2}v_{i_3}w_{i_1i_2i_3} \\ & -\ldots-\sum_{i_1=1}^{N}\sum_{i_2=1}^{N}\cdots\sum_{i_N=1}^{N} v_{i_1}v_{i_2}\cdots v_{i_N}w_{i_1i_2\ldots i_N} \\ & -\sum_{i=1}^{N} v_i\theta_i \,. \end{aligned} \tag{3}$$

A higher-order binary Hopfield network for constraint satisfaction problems can be constructed based on the set C of constraints which contains

tuples of units that cannot be 'on' simultaneously. Specifically, if $(i_1,i_2, \ldots i_k) \in C$ the weights $w_{i_1,\ldots,i_k}$ corresponding to any permutation of $(i_1,i_2, \ldots, i_k)$ are equal to $-1/k!$, otherwise they are equal to zero. Moreover, the threshold value of each unit is set equal to $\alpha < 1$. In this case the energy is given by the following formula

$$E = \sum_{(i_1,i_2,\ldots,i_k)\in C} v_{i_1} v_{i_2} \ldots v_{i_k} - \sum_{i=1}^{N} v_i \alpha \,. \qquad (4)$$

Following the operation of the discrete Hopfield network, the update of each unit i is performed in the following manner: $v_i = 0$ if $(-\Sigma_{(i,j,\ldots,k)\in C_i} v_j \ldots v_k + \alpha) < 0$, otherwise $v_i = 1$. Since $\alpha < 1$, if at least one constraint from the set C_i is violated, the state of the corresponding unit i becomes zero, otherwise it takes the value one.

It is obvious that the operation of the network is similar to the one described for our constraint satisfaction module. Based on this similarity, the quantity $\Sigma_{(i,j,\ldots,k)\in C_i} v_j \ldots v_k$ can be considered as a quantitative description of the indicator $I_k(v)$: $I_k(v) = 1$ if $\Sigma_{(i,j,\ldots,k)\in C} v_j \ldots v_k > 0$ and $I_k(v) = 0$ if $\Sigma_{(i,j,\ldots,k)\in C} v_j \ldots v_k = 0$.

It should be noted that, to our knowledge, the application of the Hopfield approach to constraint satisfaction problems has been restricted so far to the case of binary constraints, thus the usual second-order network has been employed. Moreover, a result similar to that of Proposition 2 has been proved [10] for the class of nonpositive binary Hopfield networks which represent binary constraint satisfaction problems. Since our proposed general constraint satisfaction scheme can be implemented using a higher-order Hopfield network, it is apparent that the result of Proposition 2 regarding the maximum number of cycles for convergence also carries over to the class of higher-order Hopfield networks having the characteristics described previously.

4. The Optimisation Scheme

As already mentioned, reinforcement learning has been previously applied to the solution of discrete optimisation problems in cases where the conventional penalty method was used. Moreover, the 0-1 integer programming formulation was adopted, thus the network consisted of binary stochastic units [7,8]. However, as is the case with any hillclimbing method, reinforcement learning schemes in their pure form have been shown in many cases to converge at local maxima [9]. This is due to the fact that their operation is solely based on correlations between the values of *individual* units and that of the reinforcement signal. Consequently, higher order dependencies are difficult to discover. Although some schemes have been proposed which succeed in escaping from local maxima [8], learning time is still a critical factor limiting the applicability of those schemes for solving optimisation problems. This is of great importance especially when the size of the problem becomes large or complex function landscapes must be searched. The proposed integrated approach aims at overcoming the above drawbacks of reinforcement algorithms.

In the problems considered here, the application of reinforcement learning actually involves the search of high-dimensional binary spaces. To achieve that, we have used a team of binary stochastic units with no interconnections among them. The size of the team was determined by the dimensionality M of the binary space to be searched. The value of M is assumed to be the only available information about the function before the computation begins. The operation of the units is stochastic, to attain the required exploratory behaviour. In particular, at each trial the team of units generates a binary sample point $y = (y_1, \ldots, y_M)$. Based on the evaluation of that output, the sampling distribution is biased towards the selection of points at which it is likely to obtain high function values. This is achieved by properly adjusting the parameters of the units.

At each time step, the reinforcement module samples a point in the state space S. The evaluation of each sample s is performed in two steps. First s is mapped to a feasible point s' or a partially feasible point s'' and then the corresponding value $f_c(s')$ or $f''(s'')$ is computed to provide the reinforcement signal r that is fed back to the reinforcement module to guide the update of its parameters. This process is repeated until a stopping criterion is met (e.g. if after a certain number of iterations no improvement has occurred).

The reinforcement scheme that has been considered in our optimisation module applies to processing elements of the type shown in Fig. 3 [11,12]. Such a unit employs two levels of stochasticity. At the first level the output n_i is drawn from a normal distribution with parameters μ_i and σ_i, while at the second level the output y_i is a Bernoulli random variable with parameter p_i. The latter is computed as a deterministic function f_i (the logistic) of the output n_i of the first level, i.e.

$$p_i = \frac{1}{1 + e^{-n_i}} \,. \qquad (5)$$

The above Normal/Bernoulli unit when employed

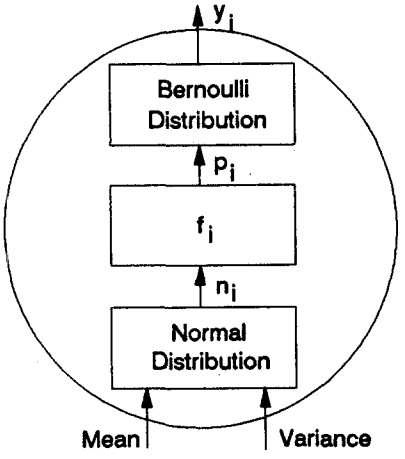

Fig. 3. The Normal/Bernoulli unit.

in conjunction with a suitable reinforcement scheme, allows the exploration of discrete domains via the modification of the mean and the standard deviation of the normal distribution. As shown in [11,12], the exploitation of the two parameters of the normal distribution leads to an improved exploratory behaviour in discrete spaces in terms of the elapsed time to find the optimal output. Moreover, experiments indicated superiority of the reinforcement schemes involving two-level Normal/Bernoulli units, over those using single-level Bernoulli units.

Considering a team of binary stochastic units such as the one shown in Fig. 3, at each time step t the parameter μ_i of each unit i is adjusted according to the following rule

$$\Delta\mu_i = \alpha_\mu(r - \bar{r})(y_i - \bar{y}_i) - \delta\mu_i \ , \tag{6}$$

where α_μ is a constant learning rate, while $-\delta\mu_i$ ($\delta < 1$) is a decay term whose role will be explained below. The quantity $\bar{r}$ is computed as an exponentially weighted average of prior reinforcement values

$$\bar{r}(t) = \gamma\bar{r}(t-1) + (1-\gamma)r(t) \ , \tag{7}$$

with γ being a decay rate ($0 < \gamma < 1$). Moreover, $\bar{y}_i$ is an average of past values of y_i and is updated as

$$\bar{y}_i(t) = \gamma\bar{y}_i(t-1) + (1-\gamma)y_i(t) \ . \tag{8}$$

The standard deviation σ, considered to be the same for all units in the team, should be small when the learning system is exploring the region of a promising local maximum, and it should be large in the opposite case [13]. One possible way to obtain the above behaviour is to update σ based on the notion of *entropy*.

The entropy of the output y of the learning system is defined as follows [8]:

$$H(y,\mu,\sigma) = -\sum_{\xi} \Pr(y = \xi|\mu,\sigma) \ln \Pr(y = \xi|\mu,\sigma) \ , \tag{9}$$

where μ is the vector of parameters μ_i and ξ represents a binary M-tuple. For a given ξ, an unbiased estimate of the entropy is given by the following quantity:

$$h(\xi,\mu,\sigma) = -\ln Pr(y = \xi|\mu,\sigma) \ . \tag{10}$$

On a particular trial at time step t, let $n_i(t)$ be the output of the first level of unit i and ξ the generated output. Then, the estimate $h(\xi,\mu,\sigma)$ of the entropy can be approximated by

$$h(t) = -\sum_{i=1}^{M} \ln \Pr(y_i = \xi_i|p_i(t)) \tag{11}$$

where $p_i(t)$ is obtained from (5) using $n_i(t)$.

When the search is confined in the neighbourhood of a local maximum, the probabilities p_i have approached one of the values 0 or 1, thus the entropy of the system tends to become zero. On the contrary, in the case of broad search the value of the entropy is high. Therefore, the update of the standard deviation σ can be performed according to the rule

$$\sigma(t) = \alpha_\sigma\hbar(t) \tag{12}$$

where the parameter α_σ takes positive values, and

$$\hbar(t) = \gamma\hbar(t-1) + (1-\gamma)h(t) \ . \tag{13}$$

The quantity $\hbar(t)$ is a trace of past values of $h(t)$ (γ being a decay factor as before) and is used in place of $h(t)$ to ensure smoother updates of the value of σ.

The reinforcement strategy described above possesses the ability of considering all the points in a large neighbourhood surrounding the current search point, instead of only considering immediately adjacent points. Such neighbourhood search procedures have the effect of avoiding local maxima of the landscape. This type of exploratory behaviour is necessary in our optimisation scheme, since the form (plateaus) of the function to be searched by the reinforcement algorithm requires that the latter be able to suggest at successive time steps sampling points which are far from each other in Hamming distance.

Another interesting feature of the above learning scheme is that of *sustained exploration* [4]. This property relates to the ability of the learning system to enter a divergence phase after it has converged to a specific state. This ensures the avoidance of local maxima since exploration continues even after

the global maximum has been generated. Sustained exploration is achieved through the incorporation of the decay term $-\delta\mu_i$ in Eq. (6). It should be noted that the alteration between a convergence and a divergence phase is triggered internally, and does not constitute an external event for the learning system.

5. Experiments

We have tested the effectiveness of our approach through experiments concerning two discrete optimisation problems, namely the set partitioning problem and the graph partitioning problem. In all the experiments, we used the scheme described in the previous section. Due to the sustained exploration property of the reinforcement algorithm, the system did not settle in a specific state. For this reason, we adopted the following conventions regarding the termination of the iterative procedure. In the set partitioning problems, for which the position of the global maximum is not known *a priori*, we keep track of the best solution found so far, and we terminate the search if for a sufficient number of consecutive iterations, the method does not manage to reach a state of greater cost than the best solution already found. In the graph partitioning problems, for which the global maximum is known, the procedure is terminated when this point is attained for the first time.

For comparison purposes we have also performed experiments using the method of simulated annealing, for both problems considered. The experiments concerned the use of simulated annealing as an independent optimisation technique and not as part of the integrated approach proposed in this paper.

5.1. Application to Set Partitioning

The formulation of the Set Partitioning Problem (SPP) has as follows [14]:

Given a finite set S containing L elements and a collection G of subsets $S_i (i = 1, \ldots, M)$ of S, find the minimum subset F of G such that all the subsets belonging to F are disjoint and they constitute a partition of S, i.e. their union is equal to S.

Consider a set of binary variables v_k $(k = 1, \ldots, M)$ whose values represent the state of the system, with $v_k = 1$ denoting that subset S_k is included in the solution set F and $v_k = 0$ denoting the opposite. The problem can be stated as follows:

$$\text{minimise} \sum_{k=1}^{M} v_k$$

subject to

$$\sum_{k=1}^{M} \sum_{\substack{i=1 \\ i \neq k}}^{M} v_k v_i a_{ki} = 0 \tag{14}$$

and

$$\sum_{k=1}^{M} |S_k| v_k = L \,, \tag{15}$$

where $|S_k|$ denotes the cardinality of subset S_k and $a_{ki} = 1$ if subsets S_k and S_i are not disjoint, otherwise it is zero.

From the above it is clear that the set partitioning problem involves two kinds of constraints: the disjointness constraint (14), and the covering constraint (15). It is not possible to construct a constraint satisfaction module that will always yield feasible solutions starting from an arbitrary state. In fact, the problem of finding just one feasible solution of a given SPP instance (not necessarily the optimal one) is NP-complete.

Therefore, our constraint satisfaction module can only provide partially feasible solutions satisfying one of the above constraints. We have chosen the satisfaction of the disjointness constraint. At each step of the optimisation procedure, the reinforcement learning module provides a binary vector of size M, which represents a selection of sub-sets S_i. The constraint satisfaction module accepts as input this arbitrary state and yields (in at most $2M$ iterations) a state corresponding to a solution set F, where all the selected subsets are disjoint. Moreover, the set F is maximal in the sense that no other subset can be added to this set without violation of the disjointness constraint. The operation of the module is based on the following definition of the indicator I_k for every $k = 1, \ldots, M$:

$$I_k(v) = 0 \quad \text{if} \sum_{i=1, i \neq k}^{M} v_i a_{ki} = 0 \,. \tag{16}$$

The cost of the solution set corresponding to the output v of the constraint satisfaction module constitutes the reinforcement signal r that is fed back to the reinforcement module to update its parameters. This signal contains information regarding the covering constraint and the cost of the problem (cardinality of F). It is computed as follows:

$$\left(\sum_{k=1}^{M} |S_k| v_k - L \right) - \frac{1}{M} \sum_{k=1}^{M} v_k \,. \tag{17}$$

Since the state v satisfies the disjointness constraints, it always holds that $\Sigma_{k=1}^{M} |S_i| v_i \leq L$, thus the first

term is maximised when the covering constraint is satisfied. The factor $1/M$ that multiplies the cost term was added to ensure that every feasible solution (satisfying the covering property) accepts higher reinforcement signal than any partially feasible solution. This fact enforces the maximisation process to search for feasible solutions first and then for solutions of optimal cost.

As already mentioned, a stopping criterion has been adopted since the optimal solution is not known *a priori*. More specifically, the optimisation procedure stops if, after $30M$ consecutive iterations, the current best solution cannot be further improved.

Experiments have been conducted on randomly generated graphs with $M = 75$ and $L = 50$. To generate the tested instances we first constructed a number of subsets of S by allocating each of the elements of S to one and only one of these subsets so as to create a disjoint solution. The remaining subsets were randomly constructed by deciding with probability q whether an element of S should belong to a particular subset. By varying the value of q, the density of the resulting problem instance could be adjusted. The value of q was chosen in the range from 0.03 to 0.1.

To further examine the effectiveness of the proposed method, we have performed comparisons with the simulated annealing approach using the penalty-based formulation [1]. The cost function to be maximised evaluates each state v as follows

$$f(v) = -\frac{\lambda}{2}\sum_{i=1}^{M}\sum_{j=1}^{M} v_i v_j a_{ij} + \left(\sum_{i=1}^{M} |S_i| v_i - L\right) - \frac{1}{M}\sum_{i=1}^{M} v_i \,, \tag{18}$$

where λ is set greater than $s_{\max} = \max\{|S_1|, \ldots, |S_M|\}$. The above choice of the parameters ensures on the one hand that every equilibrium corresponds to a solution set with disjoint subsets, and on the other hand, that every feasible solution is of higher cost than any partially feasible solution.

The temperature T is reduced according to the following logarithmic annealing schedule:

$$T_n = \frac{T_{n-1}}{1 + \log f(n)} \,, \tag{19}$$

where T_n denotes the temperature at the nth step of the schedule, and $f(n) = f(n-1)(1+\varphi)$ with $f(0) = 1$. The initial temperature T_0 was set equal to 2.0 and a very slow reduction rate $\varphi = 10^{-6}$ was adopted. At each temperature, $2M$ trials are performed, and the algorithm terminates if for 10 consecutive temperature values the system remains in the same state. In that case, the output of the procedure is the best found solution.

Table 1. Comparative results.

Algorithm	Feasible solution	Uncovered elements		
		1	2	>2
CA	67.7%	31.1%	1.2%	0%
SA	55.5%	26.6%	12.2%	5.7%

Comparative results for the proposed combined approach (denoted by *CA*) and the simulated annealing (denoted by *SA*) are displayed in Table 1. The results concern the percentage of cases (over 100 experiments) in which each method found a feasible solution (first column), or a partially feasible solution that leaves 1, 2 or more than two elements of S uncovered (second, third and fourth columns, respectively). The superiority of the proposed approach is apparent, in spite of the fact that a very slow annealing procedure was used.

5.2. Application to Graph Partitioning

The second experimental study of the effectiveness of the proposed method deals with the following problem:

Given a graph $G = (V,E)$ with $|V| = M$ and adjacency matrix A, find if there exists a partitioning of this 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 $G_B = (V_B,E_B)$. The partitioning can be represented by a binary vector as follows. With each node $k \in V$ we associate 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 $k \in V_B (k \notin V_B)$. Let $v = (v_{10}, v_{11}, \ldots, v_{M0}, v_{M1})$ denote the state vector of the system. The number of vertices of G_A is $n_A(v) = \sum_{k=1}^{M} v_{k0}$ and the number of vertices of G_B is $n_B(v) = \sum_{k=1}^{M} v_{k1}$.

Using the above representation, the problem can be stated as a constraint satisfaction one:

Find a vector v such that

$$\sum_{l=0}^{1} v_{kl} = 1, \quad k = 1, \ldots, M \tag{20}$$

$$n_A(v) = n_B(v) \tag{21}$$

$$v_{k0} \sum_{\substack{i=1 \\ i \neq k}}^{M} a_{ik} v_{i1} + v_{k1} \sum_{\substack{i=1 \\ i \neq k}}^{M} a_{ik} v_{i0} = 0, \quad k = 1, \ldots, M . \tag{22}$$

The reinforcement module employed in our scheme contains M binary units and provides a binary output vector $y = (y_1, \ldots, y_M)$. The output y_i of each unit i indicates whether node i should be included in subgraph G_A (in case $y_i = 0$) or in subgraph G_B (in case $y_i = 1$).

The operation of the constraint satisfaction module is based on the $N = 2M$ binary variables. The vector y provided by the reinforcement module specifies the initial value of vector v in the following manner. For each $k = 1, \ldots, M$, if $y_k = 0$ we set $v_{k0} = 1$ and $v_{k1} = 0$, otherwise we set $v_{k0} = 0$ and $v_{k1} = 1$. In our implementation, the constraint satisfaction module is responsible for satisfaction of the disjointness constraint (Eq. (22)), as well as of the constraint $\Sigma_{l=0}^{1} v_{kl} \leq 1$, which constitutes a relaxation of Eq. (20). Thus, the operation of the constraint satisfaction scheme is based on the following definitions of the indicators I_{k0} and I_{k1} for each $k = 1, \ldots, M$:

$$I_{k0}(v) = 0, \quad \text{if } v_{k1} = 0 \quad \text{and} \quad \sum_{\substack{i=1 \\ i \neq k}}^{M} a_{ik} v_{i1} = 0 \tag{23}$$

$$I_{k1}(v) = 0, \quad \text{if } v_{k0} = 0 \quad \text{and} \quad \sum_{\substack{i=1 \\ i \neq k}}^{M} a_{ik} v_{i0} = 0 . \tag{24}$$

The constraint satisfaction module converges in at most $4M$ steps to an equilibrium state 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. (22)) 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 second constraint (Eq. (21)) may not be satisfied, and the same holds for the first constraint (Eq. (20)), since for some k we may have that $v_{k0} = v_{k1} = 0$ (i.e. node $k \notin G'$).

Let $K(v) = \Sigma_{k=1}^{M} (v_{k0} + v_{k1})$. The evaluation of the output y of the reinforcement learning module is performed by sending a reinforcement signal r given by

$$(K(v) - M) - \kappa(n_A(v) - n_B(v))^2 , \tag{25}$$

where the parameter κ determines the relative importance of the two terms to be maximised (in all experiments the value of κ was taken equal to 0.005). It is obvious that the cost corresponding to a feasible problem solution is equal to zero and constitutes the maximum value of the reinforcement signal.

We have considered *multilevel graphs* [4] having a particular hierarchical structure. A multilevel graph is characterised by the fact that it contains clumps of nodes. A clump is a set of nodes fully connected among themselves but having only few connections to the remaining nodes of the graph. Experiments have been conducted with specific multilevel graphs containing 8, 16 and 32 clumps. Each clump may contain 4 or 6 nodes, thus the graphs considered were given the names 8×4, 16×4, 16×6, 32×4 and 32×6. The graphs 8×4 and 16×4 are referred to in [4] as *MLC-32* and *MLC-64*, respectively. The graph *MLC-32* is depicted in Fig. 4. 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.

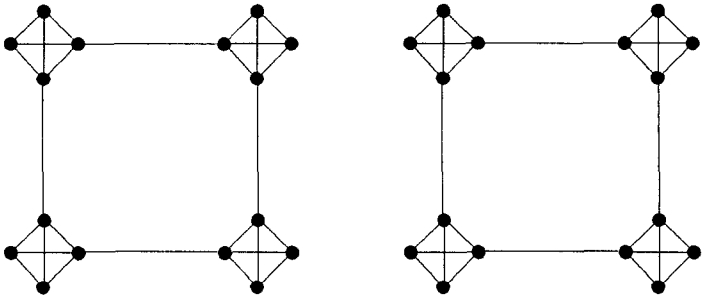

Fig. 4. The MLC-32 graph.

Table 2. Mean time (sec) for finding the optimal solution.

Algorithm	Graph type				
	8 × 4	16 × 4	16 × 6	32 × 4	32 × 6
CA	4	23	64	162	800
SA	13	140	238	480	998

As with the SPP case, we have compared the performance of our combined approach with that of simulated annealing applied to the same graph instances. In the simulated annealing approach, we consider the binary state vector $v = (v_1, \ldots, v_M)$, where $v_i = 0(1)$ indicates that node i should belong to subgraph $G_A(G_B)$. The number of nodes of subgraph G_A is $n_A(v) = \Sigma_{i=1}^{M} (1 - v_i)$, while the number of nodes of G_B is $n_B(v) = \Sigma_{i=1}^{M} v_i$.

The aim is to maximise a cost function which evaluates each state v as follows:

$$f(v) = -\sum_{i=1}^{M}\sum_{j=1}^{M} v_i(1 - v_j)a_{ij} - \kappa(n_A(v) - n_B(v))^2 . \quad (26)$$

The first term denotes the fact that subgraphs G_A, G_B must be disjoint, while the second term denotes that they should be of equal size. The value of the parameter κ was set equal to 0.005.

At each step of the algorithm, a variable v_i is selected at random and the cost of the current state is compared to the cost of the state that would result if that variable changed its value (which essentially means that node i moves from one subgraph to the other). Based on the above cost difference, a decision is made as to whether variable v_i should change its state. The annealing schedule used in the experiments was the same as in the case of the previous problem. The initial temperature was set equal to 2.0 while the rate φ was taken equal to 5×10^{-7}. At each temperature, $2M$ trials are performed and the algorithm terminates when the solution with maximum cost is attained.

Table 2 displays the mean time (over 50 experiments) required to find the optimal partitioning for the graph problems considered. As before, the displayed results concern the proposed combined approach (denoted by *CA*) and the simulated annealing (denoted by *SA*). It should be noted that in the case of simulated annealing, in spite of the slow temperature reduction rate, convergence was attained in some experiments without having found the optimal solution. The above results clearly illustrate the increased efficiency of the proposed method as compared to simulated annealing.

6. Conclusions

We have proposed a method for the solution of discrete optimisation problems. The method is based on the synergy of an exploration technique and a constraint satisfaction scheme. The exploration module searches the state space of the problem by means of an iterative procedure that generates and evaluates points. For the evaluation of each point, first the constraint satisfaction module is used to map the given point onto some other point that satisfies the problem constraints (or part of them). The cost of the new point constitutes the evaluation of the original point generated by the exploration module.

The application of the method to two optimisation problems shows that the proposed technique is very efficient in providing near-optimal feasible solutions. Due to the particular shape of the function that must be explored, reinforcement learning algorithms are naturally appropriate for implementing an exploration strategy. The use of a reinforcement learning module as a point generator leads to a better exploration of the discrete space, especially when advantage is taken of the parameters of the normal distribution.

As a general remark, it can be stated that the effectiveness of the method is more apparent in the case of optimisation problems that contain a constraint part involving a number of different constraint sets that are difficult to satisfy. In such a case, the constraint part can be more important than the cost part of the problem formulation, and it is very difficult to find feasible solutions.

It should be noted that the method can be applied as well to pure constraint satisfaction problems that involve no cost part to be optimised. In such cases, the constraint satisfaction scheme is responsible for the satisfaction of a part of the problem constraints, whereas the remaining part can be formulated as a cost function that must be optimised.

Another point that deserves attention concerns the selection of the set S''. In general, for a given problem there may exist several alternatives for the definition of S''. A general rule is to define S'' so that the contained states satisfy as many constraints as possible. This argument implies that the most favourable situation is when we can set $S'' = S'$. The success of the above rule, however, depends upon the shape of the cost function that must be optimised. It is possible, in some cases, that there exist paths leading to the optimal solution which make their way through regions of the state space containing partially feasible solutions, and that these paths are shorter than other paths running

exclusively through feasible solutions. Thus, it can be advantageous to perform the exploration in the space S'' of partially feasible solutions rather than in the space S' of feasible solutions.

References

1. Aarts E, Korst J. Simulated Annealing and Boltzmann Machines, A Stochastic Approach to Combinatorial Optimisation and Neural Computing. John Wiley, Chichester, 1989
2. Hopfield J. Neural networks and physical systems with emergent collective computational abilities. In: Proc. National Academy of Sciences USA 1982; 79: 2554–2558
3. Kirkpatrick S, Gellat Jr. CD, Vecchi MP. Optimization by simulated annealing. Science 1983; 220: 671–689
4. Ackley DH. A Connectionist Machine for Genetic Hillclimbing. Kluwer, Norwell, MA, 1987
5. Goldberg DE. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading, MA, 1989
6. Williams RJ. Toward a theory of reinforcement learning connectionist systems. Technical Report NU-CCS-88-3, Boston, MA, 1988
7. Williams RJ, Peng J. Reinforcement learning algorithms as function optimizers. In: Proc. Int Joint Conf Neural Networks, vol II. Washington, DC, 1989, pp 89–95
8. Williams RJ, Peng J. Function optimization using connectionist reinforcement learning networks. Connection Science 1991; 3: 241–268
9. Williams RJ. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning 1992; 8: 229–256
10. Shrivastava Y, Dasgupta S, Reddy SM, Guaranteed convergence in a class of Hopfield Networks. IEEE Trans Neural Networks 1992; 3(6): 951–961
11. Kontoravdis D, Likas A, Stafylopatis A. A reinforcement learning algorithm for networks of units with two stochastic levels. In: Proc ICANN-92, vol I. Brighton, UK, 1992, pp 143–146
12. Kontoravdis D, Likas A, Stafylopatis A. Enhancing stochasticity in reinforcement learning schemes: Application to the exploration of binary domains. Journal of Intelligent Systems (to appear)
13. Gullapalli V. A stochastic reinforcement learning algorithm for learning real-valued functions. Neural Networks 1990; 3: 671–692
14. Garey M, Johnson D. Computers and Intractability: a Guide to the Theory of NP-completeness. W.H. Freeman and Company, San Francisco, 1979