Global search in combinatorial optimization using reinforcement learning algorithms

Conference Paper · February 1999

DOI: 10.1109/CEC.1999.781925 · Source: IEEE Xplore

2 authors:

Victor V Miagkikh
Institute of Electrical and Electronics Engineers
17 PUBLICATIONS **172** CITATIONS
SEE PROFILE

William F. Punch
Michigan State University
146 PUBLICATIONS **5,439** CITATIONS
SEE PROFILE

Global Search in Combinatorial Optimization using Reinforcement Learning Algorithms

Victor V. Miagkikh and William F. Punch III
Genetic Algorithms Research and Application Group (GARAGe)
Michigan State University
2325 Engineering Building
East Lansing, MI 48824
Phone: (517) 353-3541
E-mail: {miagkikh,punch}@cse.msu.edu

Abstract- This paper presents two approaches that address the problems of the local character of the search and imprecise state representation of reinforcement learning (RL) algorithms for solving combinatorial optimization problems. The first, Bayesian, approach aims to capture solution parameter interdependencies. The second approach combines local information as encoded by typical RL schemes and global information as contained in a population of search agents. The effectiveness of these approaches is demonstrated on the Quadratic Assignment Problem (QAP). Competitive results with the RL-agent approach suggest that it can be used as a basis for global optimization techniques.

1 Introduction

The success of a generate-and-test algorithm for a particular problem is determined by factors such as: its ability to use past experience to form feasible solutions, its exploitation/exploration strategy, its utilization of problem-specific information etc. In creating a feasible solution, the algorithm has to make a number of decisions, e.g. which value should be assigned to a particular free parameter. The quality of the solution generated is often the only type of feedback available after a sequence of decisions is made. Since we expect the algorithm to make decisions which result in better solutions over time, the problem of intelligent solution generation can be approached with reinforcement learning (RL).

The problems with delayed reinforcement that RL approaches face are well modeled by *Markov Decision Processes* (MDPs). MDPs are defined by: a set of Markov states, the actions available in those states, the transition probabilities and the rewards associated with each state-action pair. Model based RL-algorithms are explicitly looking for MDP solution, an optimal *policy*, which is a mapping from MDP states to actions which maximizes the expected average reward received by following a path through MDP states. An *action-value function* for a policy is defined as a mapping from each state-action pair to the expected average reward obtained by choosing an action in that state according to the given policy and following that policy thereafter. The *state-value function* for a policy specifies the desirability of a state and is defined as the expected average reward obtained by following that policy from a given state. Since probabilities are not always known, typical RL algorithms, e.g. SARSA or Q-learning, are model free. Iterative updates used by these algorithms do not use transition probabilities and are proven to converge to optimal value function. A greedy policy that chooses an action according to the maximum optimal action-value is known to be globally optimal based on the expected average reward criterion. Readers interested in a more detailed treatment of RL should read references such as Sutton and Barto (1997).

For an example of an optimization problem formulated in RL terms, consider an RL approach to the Traveling Salesman Problem (TSP). The states are the cities. The actions are the choices of the next city to visit, and the action-values indicate the desirability of the city to visit next. Global reward is the inverse of the tour length. Immediate rewards can be defined as inverse of the distance between a pair of cities.

The idea of using RL in optimization problem solving is almost as old as RL itself. It was first studied in the n-armed bandit by Bellman (1956) and later applied to more difficult optimization problems by various researchers. For example, Dorigo (1992,1996) has developed an optimization technique known as Ant Systems(AS). The key idea behind AS is a heuristic approximation to action-values, which he terms *pheromone*. Even though AS were derived by simulating the behavior of a population of ants, they have much in common with other RL algorithms. Another application of RL to optimization is that of Crites and Barto (1996) where they applied Q-learning to elevator scheduling. This paper is particularly relevant to our research since it explores the possibilities of multi-agent RL algorithms in optimization. Each agent in a team of RL algorithms controls a particular elevator car cooperatively solving the entire problem. Among other relevant publications, Gambardella and Dorigo (1995) who described the application of Q-learning to TSP and asymmetric TSP. Zhang and Dietterich (1996) use TD(λ) to solve a Job-Shop Scheduling problem. Singh and Bertsekas (1996) used RL for the channel allocation problem.

In order to discuss advantages and disadvantages of RL in optimization, let us first contrast them against another well-known optimization technique, genetic algorithms (GA). While RL supports value-function, which reflects the desirability of free parameter assignments, GA approaches explicitly view only the overall fitness of a solution. In constructing a new solution, GAs are not guided by any synthetic fitness values associated with any smaller part of solution. Rather, GAs are guided by schema theory which states that the more favorable a particular choice of values for a subset of solution parameters is, the more frequently such a schema appears as a part of solutions in the population. These *building blocks* thus represent the preferred values of solution parameters and their combinations. The ability to both explore and exploit schemata in the search space is the key to GA success as first pointed out by Holland (1975).

Thus, each schema in a GA has an implicit probability of appearing in generated solution where the better a schema is, the higher the probability of it occurring in a solution. Such a representation is similar to tossing a coin and storing all outcomes instead of the number of trials and the number of heads. This raises the question: is a population of solutions an accurate and computationally effective way of representing the preferences of free parameter choices as compared to some form of sufficient statistics? Since the number of schemata grows exponentially with the size of the problem, maintaining values associated with each possible combination of parameters becomes prohibitive. On the other hand, GAs do not directly learn from bad experience. Moreover, finite populations can drop some alleles from the population and there is only a slight chance that they may be reintroduced via mutation, and they may not survive to be used.

Use of RL techniques in optimization problems has good and bad aspects. On the positive side, they are proven to converge to optimum given the right circumstances and are applicable to problems with a large number of states. They can also be used in conjunction with function-approximation techniques to add generalization and reduce space requirements. Boyan and Moore (1998) report good results on a number of discrete and continuous optimization problems using this approach. Direct estimation of desirability of assignments by value functions has a potential to be both more precise and computationally cheaper than other approaches. This possibility is one of the major motivations for conducting research on applicability of RL algorithms to optimization.

There are also disadvantages. The first is the local rather than global character of search in the RL schemes proposed so far. The algorithm has to explore the space by choosing probabilistically from among all actions, not just the action with the highest action-value estimate. In combinatorial optimization, even one incorrect exploratory step can seriously damage the quality of the resultant solution. Therefore, to generate a good solution, the most preferable action has to be selected most of the time, which strongly shifts the balance from exploration to exploitation and leads to local rather than global search.

Another problem in RL is the coarse representation of the state. For instance, in solving a TSP by AS as described in Dorigo *et al.* (1996) or Ant-Q in Gambardella and Dorigo (1995), the state is the current city, and the action is which city to visit next. Clearly a *full representation* of the state would contain both the current city *and* the tour of cities already visited. Since this history obviously influences further assignment, their simple definition loses the Markov property, and a suboptimal sequence of cities, a building block, is not captured. Consequently, the algorithm will not be able to handle parameter interdependence sufficiently well. As mentioned earlier, the number of states in an RL approach cannot be so large as to keep an estimate for every possible sequence because the number of states grows exponentially. Nevertheless, this problem may be addressed by the use of function-approximation and other means as will be discussed further.

2 Capturing Parameter Interdependencies Using a Bayesian Approach

The coarse state representation does not take into account interaction of available assignments with those already made. To include these interactions we can use a Bayesian approach.

To construct a feasible solution for a combinatorial optimization problem, a number of free parameters should be instantiated. Let $x \leftarrow \psi$ denote the fact that some free parameter x is assigned value ψ. For example, in the Quadratic Assignment Problem described in section 4, the free parameters are locations and the values to be assigned to those parameters are ordinal numbers of facilities. Let $P(y \leftarrow \chi \mid x \leftarrow \psi)$ denote conditional probability of assigning a free parameter y the value χ given that x was already assigned ψ. This conditional probability indicates an assignment already made influences the probability of the assignment under consideration. We can find $P(y \leftarrow \chi \mid x \leftarrow \psi)$ using Bayes' rule:

$$P(y \leftarrow \chi \mid x \leftarrow \psi) = \frac{P(x \leftarrow \psi \mid y \leftarrow \chi) P(y \leftarrow \chi)}{P(x \leftarrow \psi)} \quad (2.1)$$

where $P(y \leftarrow \chi)$ is the prior probability of assigning y to χ and $P(x \leftarrow \psi \mid y \leftarrow \chi)$ is the likelihood of assignment $y \leftarrow \chi$ with respect to assignment $x \leftarrow \psi$. If we set

$P(x \leftarrow \psi)$ to one, (2.1) can be simplified to $P(y \leftarrow \chi \mid x \leftarrow \psi) = P(x \leftarrow \psi \mid y \leftarrow \chi)P(y \leftarrow \chi)$. Now suppose that a sequence of k assignments S≡($x_1 \leftarrow \psi_1, x_2 \leftarrow \psi_2, ..., x_k \leftarrow \psi_k$) was made. The posterior probability of assignment $y \leftarrow \chi$ given that assignments in S are made is $P(y \leftarrow \chi \mid S) = P(S \mid y \leftarrow \chi)P(y \leftarrow \chi)$. By computing this posterior probability for all candidate values of χ for free parameter y in consideration, one can find how particular χ fits with assignments already made. Note that the number of conditional probabilities $P(S \mid y \leftarrow \chi) \equiv P(x_1 \leftarrow \psi_1, x_2 \leftarrow \psi_2, ..., x_k \leftarrow \psi_k \mid y \leftarrow \chi)$ grows exponentially with the size of the problem. We can use the naive Bayesian approach to address this problem. Assuming independence of the assignments $x_i \leftarrow \psi_i$ we get:

$$P(y \leftarrow \chi \mid S) = P(y \leftarrow \chi)\prod_{i=1}^{k} P(x_i \leftarrow \psi_i \mid y \leftarrow \chi) \quad (2.2)$$

The probability of assignment $y \leftarrow \chi$ given that k assignments in S were already made is the product of its prior probability $P(y \leftarrow \chi)$, and conditional probabilities $P(x_i \leftarrow \psi_i \mid y \leftarrow \chi)$ for all prior assignments. If n is the total number of decisions to construct a feasible solution, then there are O(n^4) conditional probabilities $P(x_i \leftarrow \psi_i \mid y \leftarrow \chi)$. Thus, even the naive Bayesian scheme leads to a high, *O(n^4)*, space complexity, which is acceptable only for moderately sized problems.

Since prior and conditional probabilities are not known, their estimates $\hat{P}(y \leftarrow \chi)$ and $\hat{P}(x_i \leftarrow \psi_i \mid y \leftarrow \chi)$ should be determined in the course of the search. Those can be found as the frequencies of assignments' co-occurrences. However, maintaining both probability estimates *and* value-function estimate involves significant increase of space and computational requirement. To avoid this, we may keep only value function estimates using dependence of actions' probabilities on action-values and policy being used. Under any reasonable policy, the actions with higher action-values have more chance of being selected. From RL point of view, a state corresponds to a free parameter and an action corresponds to a choice of value for that parameter. For proportional policy, the probability estimate, $\hat{P}(s,a)$, of action a in state s, is proportional to its action-value, $\hat{Q}(s,a)$:

$$\hat{P}(s,a) = \frac{\hat{Q}(s,a)}{\sum_{\forall a' \in A} \hat{Q}(s,a')} \Rightarrow \hat{P}(s,a) \propto \hat{Q}(s,a) \quad (2.3)$$

where A is the set of all actions available in s. Assuming (2.3) for all possible policies, we can use Bayesian scheme not on the probability estimates, but on normalized action-values. For example, using a Monte Carlo update rule, we can find the action-values of assignments as:

$$\tilde{P}(y \leftarrow \chi) = \tilde{P}(y \leftarrow \chi) + \alpha[r - \tilde{P}(y \leftarrow \chi)] \quad (2.4)$$

$$\begin{aligned}\tilde{P}(x_i \leftarrow \psi_i \mid y \leftarrow \chi) &= \tilde{P}(x_i \leftarrow \psi_i \mid y \leftarrow \chi) + \\ &+ \beta\left[r - \tilde{P}(x_i \leftarrow \psi_i \mid y \leftarrow \chi)\right]\end{aligned} \quad (2.5)$$

where $\tilde{P}(y \leftarrow \chi) \equiv \hat{Q}(y,\chi) \propto \hat{P}(y,\chi) \equiv \hat{P}(y \leftarrow \chi)$ to conform with our notation for assignments and show the relationship with probabilities. $\tilde{P}(x_i \leftarrow \psi_i \mid y \leftarrow \chi)$ is the expected average reward for taking action ψ_i in state x_i given that action χ was taken in state y. The reward r can be based on the comparison with the average fitness, *avFit*, of the last M solutions generated:

$$r = (avFit - Fitness)/avFit + 0.5 \quad (2.6)$$

The high space complexity of this approach is a problem. This approach is therefore only possible for problem instances of moderate size unless used along with function-approximation. Also, since only n^2 of the total of O(n^4) entries $\tilde{P}(x_i \leftarrow \psi_i \mid y \leftarrow \chi)$ get updated per iteration, this method will converge slowly. However, since this approach attempts to solve the problem of parameter interdependence directly, it has significant theoretical importance for the sake of comparison with indirect approaches, one of which will be introduced further.

3 The Approach Based on a Population of RL Search Agents

Since direct capturing of parameter interdependencies by keeping additional estimates is expensive, we can think about indirect approaches. One such approach is as follows: we continue to use a coarse representation of the state but stop looking for general preference values which would be valid in any part of the search space. Since coarse representation collapses many "true" states of the system into one making them indistinguishable, the action-values associated with "coarse state"-action pairs can only be valid for a local part of the search space. We will call this the *principle of locality of action-values*. However, action-values from different parts of the search space *can* be more broadly applicable. Therefore, this approach maintains a population of not only solutions, which are the best results of the search conducted by the RL algorithm situated in some area of the search space, but also their action-values. This coupling of a locally-best solution, the action-values and an RL algorithm is defined as an agent, an expert in its local area of the search space. As soon as we have local information from different parts of the search space, we need a way to combine the results of "best yet" search in one area with another.

Since each agent in the population is addressing the same optimization problem, we expect that at least some other agent's action-values are useful in areas other than the local space in which there were formed. This assumption of homogeneity allows us to combine results from multiple agents. Consider one such approach: a new solution is formed by copying a part of the locally-best solution found by one agent, while the remaining assignments are made using action-values borrowed from another agent. How would this compare to recombining two solutions using GA crossover? In GA crossover we have two kinds of information, the two instances and perhaps some problem-specific information. For example, Grefenstette (1985) crossover for TSP has to make 40% of its assignments at random to avoid conflicts with previous assignments. With action values, we can *direct* those assignments rather than make them randomly. This increases the chances of finding a good sequence. Thus, the operation described looks like a kind of crossover, using two instances to generate one child, based on indirect transfer of information though action-values. We may also think of it as combining both partial results and preferences resulting from search conducted by other agents. Possible variation of this theme is to generate a partial solution with one agent and use another agent to generate the remainder. Approaches using both a central solution and action-values are also possible. This synthetic approach would allow combining the advantages of both RL and GA.

In addition to capturing interdependencies, a population of RL search agents provides oportunities for more global search. As was noted in the introduction, the local character of the search comes in part from constructing the entire solution from scratch. Our approach uses an RL algorithm to generate not the whole solution, but only a part of it. The other part is replicated from the best solution found so far by this or another RL algorithm. At first glance this might seem to make the approach even more locally oriented. This is the case only if the replicated part is discovered by some other agent, which followed a similar thread of search. To enforce independent threads of search as conducted by each agent in the population, we can choose the following replacement policy: the child competes with the parent which was the source of replicated material, and the better solution (parent or child) is placed into the next generation. In this case, two agents are "similar" (same preferences, etc.) only if they discovered the same solution independently based on their own action-values.

Another way to make the search more global is to allow the RL approach to "wander" more (follow less stringently its preferences). To avoid introducing poor solutions into the population, each solution can be passed through a problem-specific local optimizer to see if this exploration found a useful area of the search space. These two approaches are complementary because independent threads reduce crowding which can cause preliminary convergence to a local optimum. In its turn, local optimization allows broader search by allowing parameters controlling exploration in the RL algorithm to be set less tightly.

Since an instance in the population is not only a solution, but also a matrix of action-values, it is costly to copy. This is one of the reasons that competitive replacement is used in the algorithm. We assume here that if the child is better than the parent which served as the source of replicated part, then the child inherits all the preference values of that parent. Depending on the results of competition, the update of the preference values is made either in both parents or in the child and the parent and there is no need to copy them.

```
Initialize population and parameters;
Repeat
    Select two agents $A_1$ and $A_2$ from the population using e.g.
           proportional selection based on the fitness of central
           solution;
    For each free parameter with probability $\lambda$ do
           Copy the value of free parameter from $A_1$ to offspring $O$;
    End
    For each unassigned free parameter in $O$ do
     In problem specific order:
     Select a value to be assigned to this free parameter from the set
           of possible value according to some policy based on the
           action-values of $A_2$ and assign it to that free parameter;
    End
    Pass $O$ through local optimizer (optional step);
    Evaluate $O$; $f(O)$ denotes fitness of $O$;
    Compute reward $r$;
    If $f(O)$ is better then the fitness of central solution of $A_1$ then
           Copy $O$ to central solution of $A_1$;
    End
    Update action-values of $A_1$ and $A_2$ using reward $r$;
Until termination condition;
Output best solution in population
```

Figure 1: High level pseudocode for RL-agent approach.

The high-level pseudocode of the approach is shown in Figure 1. There is a population of RL agents where each is comprised of a locally best solution, a matrix of action-values and the parameters for the RL algorithm. To produce a new agent, two solutions are selected from a population using proportional or another type of selection. The new solution is formed using the solution of one parent and the action-values of the other. After calculating the fitness of the new solution, the child competes with the parents for inclusion in the population. Then the value-functions are updated that completes the generation cycle.

The reward could be based on the difference of the fitness of the new solution and the average fitness of the parents or some other baseline. Depending on the problem being solved and the particular RL algorithm used, local rewards could also be employed.

4 Application to the QAP

The Quadratic Assignment Problem is a *NP-hard* problem of finding a permutation φ minimizing:

$$Z = \sum_{i=1}^{n} C_{i\varphi(i)} + \sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij} \cdot B_{\varphi(i)\varphi(j)} \qquad (4.1)$$

where n is the number of facilities/locations, C_{ij} cost of locating facility i at location j, A_{ij} is cost of transferring a material unit from location i to location j, B_{ij} is the flow of material from facility i to facility j. The permutation φ indicates the assignment of a facility to a location. The double summation of the products term makes the QAP highly non-linear. The preference values can estimate the goodness of assigning a specific location to some facility. The result of assigning facility to location is highly dependent on how other facilities are assigned. This property makes this problem to be very interesting subject for testing the presented approaches on.

4.1 Population-Based Approach

In accordance with the approach, the new feasible solution is formed in part by replicating the fragments of the best solution discovered by one of the agents and filling in the remaining part using the value function of another agent. To construct a new feasible solution, unoccupied locations were selected in random order and assigned facilities using ε-greedy proportional policy. This policy with the probability ε chooses the facility from the set of not-yet-assigned facilities having maximum action-value, or with probability 1-ε, chooses one of the remaining options with probability proportional to the estimate of desirability for that assignment.

The balance between copying the fragments of the best solution and generating the rest using preference values is controlled by the parameter λ, which is the fraction of copied values among the total number of assignments. Thus, λ=0 corresponds to use of the RL algorithm to make all assignments. Each cell had a probability of being copied equal to λ / n and the remaining on average $n(1-\lambda)$ positions were filled using action-values $Q(l_i, f_i)$ reflecting desirability of assigning facility f_j to location l_i.

In the QAP, there is no obvious order in which assignments should be made. It makes the application of bootstrapping RL algorithms such as Q-learning difficult unless some order is imposed that would put a strong bias on solution generation. There are a number of ways to resolve this difficulty, but in the context of the present approach, a simple Monte Carlo update (4.2) that does not require a particular order was used, at the price of slower convergence. The authors used a bootstrapping Q-learning update rule in application of this approach to the Asymmetric Traveling Salesman Problem (ATSP) described in Miagkikh and Punch (1999). In this application, the action-values, $Q(l_i, f_i)$, were learned using simple Monte Carlo update:

$$Q(l_i, f_i) = Q(l_i, f_i) + \alpha(r - Q(l_i, f_i)) \qquad (4.2)$$

where reward r was calculated on the basis of the average fitness of two parents according to (2.6). Generated solutions were improved by a simple 1-Opt optimizer.

4.2 Bayesian Approach

To implement the approach capturing interdependencies directly, one RL algorithm, a replica of the agent in the population from the population-based approach, was used. This RL algorithm was augmented with an n^4 matrix of conditional average rewards. The $O(n^2)$ procedure (2.5) was used to update the entries of this matrix. The action-values were computed according to (2.4). The reward r for these two updates was calculated by (2.6), where *avFit* was an average fitness of last M solutions produced. The process of feasible solution generation was identical to the RL-agent approach with a few changes: posterior action-values computed by (2.5) were used instead of action-values in (4.2). The procedure for accepting a new solution was also relaxed in comparison to the population-based approach. The new solution is accepted as a new center if the reward is greater than some constant threshold T. Since the average fitness of solution decreases during the course of minimization, this acceptance rule resembles an annealing schedule. The same ε-greedy rule was used as found in the population-based approach, however a different parameter ε was generated for each iteration to improve the ability of escaping local minima. As in the first approach, a simple 1-Opt optimizer was used to improve newly created solutions.

5 Results

The experimental runs of the RL agent approach were based on a population of 50 agents, which is relatively small for a GA, but was enough to obtain good results using this scheme. Roulette wheel selection was used. The parameter λ was randomly generated in range the [0.7,0.95] for each application of RL crossover. A generation-based approach with a crossover rate 0.1 was used. The learning and selection greediness parameters were in ranges [0.05,0.15] for α, and [0.4,0.95] for ε, respectively. These ranges of parameters were found in a series of preliminary (not described) experiments. Each of the agents in the population was assigned a combination of parameters in these ranges during initialization. Thus there was a broad range of agent types, based on a random selection of the various control parameters.

For the approach based on Bayes' rule, the parameters were β=0.1, α=0.1, T=0.55, M=300. The parameters ε and

λ were randomly generated in the intervals [0.3,0.99] and [0.5,0.9], respectively, for each iteration.

Results of RL agents and Bayesian approaches based on the averaging of 10 runs over some of the benchmark problems from QAPLIB by Burkard *et al.* (1997) are given in Tables 1 and 2, respectively (in Appendix). Since the RL-agent approach has many features in common with AS and GA, those approaches are used as a comparison. The columns AS and GA+LS in Table 1 show the results obtained with AS due to Maniezzo and Colorni (1998) and GA with local search by Merz and Freisleben (1997) respectively.

The RL algorithm with Bayesian correction was compared with two other one-point methods to contrast the quality of search and the ability to scale. The columns AS and GRASP in Table 2 show the results obtained with AS due to Maniezzo and Colorni (1998) and greedy randomized search procedure (GRASP) by Li *et al.* (1994). The RL algorithm with Bayesian correction was tested on the same set of benchmarks as AS in Maniezzo and Colorni (1998). Unfortunately, only the values of the best-found solution are available for AS and GRASP. Thus, we cannot compare them on average. However, the RL algorithm with Bayesian correction obtained better results in terms of the best-found solution in 7 out of 34 benchmarks. There was a tie in the remaining 27 test problems. However, the presented RL algorithm does not scale well for the larger problems.

In the case of the population-based approach, the results are much better. The RL-agents achieved the same or better performance on all test problems in comparison to AS-based algorithms. In comparison to GA+LS, the approach presented showed results which were better on some benchmarks and slightly worse on the others (of 15 problems, 8 better and 7 worse). It can be concluded that the RL-agents approach and GA+LS were quite competitive.

One of the remarkable features is the consistency of the search in the RL-agents approach: the presented algorithms found the optimum or best-known solution in each of the 10 runs on all small and moderate-sized instances. This is not the case with GA+LS, which had a non-zero standard deviation of the best-found solutions even on relatively simple QAP benchmarks such as Nug30 or Kra30a. Unfortunately, only a small number of benchmark results for GA+LS are available, which precludes more detailed comparison.

Comparing the two approaches presented in this paper, the population of RL agents certainly wins. The RL algorithm with Bayesian correction cannot provide a quality of search equivalent to the first approach. Even though our experiments show some advantage to the Bayesian approach over other known non-population-based techniques, it does not compare well with either RL agents or GA. RL with Bayesian correction has an advantage in the number of function evaluations in comparison with population-based approaches, though it has high space requirements.

6 Conclusions and Future Work

The results of the approach using a population of RL agents are competitive with the other search techniques. The authors also applied this approach to ATSP and obtained good results (Miagkikh and Punch 1999). The approaches presented addresses the two major problems of RL algorithms in application to optimization, namely, the local character of search and coarse state representation. It has been shown that these problems can be overcome to obtain a global search technique capable of producing good results.

There are still many issues to be addressed. One of them is to show that the preference values are a computationally cheaper and more precise way of maintaining desirability in comparison to GA and other search techniques, and if so, under what conditions? There are many other problems, such as the absence of natural ordering in QAP and many other search problems, which can result in complications when bootstrapping RL update rules are used. The high space complexity of the Bayesian approach gives it more theoretical rather than practical value if a tabular representation for the value function is used. The authors are working on implementation of this approach using function approximation. However, in spite of all these and other difficulties, the results obtained are very encouraging.

References

Sutton, R. and Barto, A. (1997). *Reinforcement Learning: An Introduction*. MIT Press.

Bellman, R. (1956). A Problem in Sequential Design of Experiments", *Sakhuya*, 16:221-229.

Dorigo, M. (1992). *Optimization, Learning and Natural Algorithms*. Ph.D.Thesis, Politecnico di Milano, Italy, in Italian.

Dorigo, M., Maniezzo, V., and Colorni, A. (1996). The Ant System: Optimization by a Colony of Cooperating Agents, *IEEE Trans. on SMC-Part B*, 26 (1):29-41, IEEE Press.

Gambardella, L. and Dorigo, M. (1995). Ant-Q: A Reinforcement Learning Approach to the Traveling Salesman Problem. In Proc. *12th Int. Conf. on Machine Learning*, 252-260, Morgan Kaufmann.

Crites, R. and Barto, A. (1996). Improving Elevator Performance using Reinforcement Learning, *Advances in Neural Information Processing Systems: Proc. of the 1999 Conf.*, 1017-1023, MIT Press.

Singh, S. and Bertsekas, D. (1996). Reinforcement Learning for Dynamic Channel Allocation in Cellular Telephone Systems. In Proc. of *Advances in*

Neural Information Processing Systems, 974-980, MIT Press.

Zhang W. and Dietterich T. (1996). High Performance Job-Shop Scheduling with a Time-delay TD(λ) Network. In Proc. of *Advances in Neural Information Processing Systems*, 1024-1030, MIT Press.

Holland, J. (1975). *Adaptation in Natural and Artificial Systems*. University of Michican Press.

Grefenstette, J. *et al* (1985). Genetic algorithms for the traveling salesman problem. In Proc. of *1st Int. Conf. of Genetic Algorithms and their applications*,160-165, Lawrence Erlbaum Associates Publishers.

Boyan, J. and Moore A. (1998). "Learning Evaluation Functions for Global Optimization and Boolean Satisfiability", *15 National Conf. on AI*, AAAI.

Miagkikh, V. and Punch, W. (1999). "An Approach to Solving Combinatorial Optimization Problems Using a Population of Reinforcement Learning Agents", To appear *in Proc. of the Genetic and Evolutionary Computation Conference (GECCO-99)*, Morgan Kaufmann.

Burkard, R., Karisch, S., and Rendl, F. (1997). QAPLIB - A Quadratic Assignment Problem Library. *Journal of Global Optimization*, 10:391-403.

Maniezzo, V. and Colorni, A. (1998). The Ant System Applied to the Quadratic Assignment Problem. To appear *in IEEE transactions on Knowledge and Data Engineering*.

Merz P. and Freisleben B. (1997a). A Genetic Local Search Approach to the Quadratic Assignment Problem. In Proc. of *the 7th Int. Conf. on GA (ICGA'97)*, 465-472.

Appendix

Table 1: Results of RL-Agent approach on QAPLIB by Burkard *et al.* (1997).

The meaning of the columns is as follows: Benchmark – the name of the benchmark; Opt./BKS. – optimal or best known solution for this problem; Best – the best result found by the population of RL agents in 10 runs; Average – average among the best solutions found in 10 runs; Std. Dev. – standard deviation of the distribution of the values of the best solutions found; NFE – average number of the function evaluations to find the best solution; AS - the fitness of the **best** solution obtained by the AS by Maniezzo and Colorni (1998); GA+LS the **average** fitness of solution obtained by GA with local search as described in P. Merz, B. Freisleben (1997). The best solution among the three techniques is bolded.

Benchmark	Opt./BKS.	Best	Average	Std. Dev	NFE	AS	GA+LS
Bur26a	5426670	5426670	5426670	0.0	26187	5426670	N/A
Bur26b	3817852	3817852	3817852	0.0	44086	3817852	N/A
Bur26c	5426795	5426795	5426795	0.0	41360	5426795	N/A
Bur26d	3821225	3821225	3821225	0.0	30020	3821225	N/A
Bur26e	5386879	5386879	5386879	0.0	55209	5386879	N/A
Bur26f	3782044	3782044	3782044	0.0	16630	3782044	N/A
Bur26g	10117172	10117172	10117172	0.0	59161	10117172	N/A
Chr20a	2192	2192	2192	0.0	304419	2192	N/A
Chr20b	2298	2298	**2298**	0.0	628084	2362	N/A
Chr20c	14142	14142	14142	0.0	35636	14142	N/A
Chr22a	6156	6156	6156	0.0	299388	6156	N/A
Chr22b	6194	6194	**6194**	0.0	416755	6254	N/A
Esc32a	130	130	130	0.0	264	130	N/A
Kra30a	88900	88900	88900	0.0	70563	88900	N/A
Kra30b	91420	91420	91420	0.0	524071	91420	N/A
Lipa20a	3683	3683	3683	0.0	6716	3683	N/A
Lipa30a	13178	13178	13178	0.0	39671	13178	N/A
Lipa40a	31538	31538	**31538**	0.0	178556	31859	N/A
Nug20	2570	2570	2570	0.0	17524	2570	N/A
Nug30	6124	6124	**6124**	0.0	488602	**6124**	6125.6
Scr20	110030	110030	110030	0.0	61332	110030	N/A
Ste36a	9526	9526	**9526**	0.0	637048	9598	9535.6
Ste36b	15852	15852	**15852**	0.0	132011	15892	N/A
Ste36c	8239110	8239110	**8239110**	0.0	1239520	8265934	N/A
Sko100a	152002	152250	152374.6	104.51	4925566	N/A	**152253.0**
Tai60a	7208572	7299714	**7305455**	6818.07	5937914	N/A	7309143.4
Tai60b	608215054	608215054	608283498	76833.33	5783030	N/A	**608215040.0**
Tail100a	21125314	21452028	21505981.8	27060.3	7516822	N/A	**21372797.6**
Tail100b	1185996137	1186007112	**1187068525**	998793.5	4328802	N/A	1188197862.44

Benchmark	Opt./BKS.	Best	Average	Std. Dev	NFE	AS	GA+LS
Tai150b	498896643	501198597	502255500.1	704186.2	2547670	N/A	**502200800.0**
Tai256c	44759294	44830390	**44838185.14**	8639.366	68935793	N/A	44839138.3
Tho30	149936	149936	149936	0.0	2951608	149936	N/A
Tho40	240516	240516	**240516**	0.0	3754472	242108	N/A
Tho150	8134030	8166808	8170678	8149.168	9476401	N/A	**8160088.0**

Table 2: Results of RL algorithm with Bayesian correction on QAPLIB by Burkard *et al.* (1997).

The meaning of the columns is as follows: Benchmark – the name of the benchmark; Opt./BKS. – optimal or best known solution for this problem; Best – the best result found by the population of RL agents in 10 runs; Average – average among the best solutions found in 10 runs; Std. Dev. – standard deviation of the distribution of the values of the best solutions found; NFE – average number of the function evaluations to find the best solution; AS - the fitness of the **best** solution obtained by the AS by Maniezzo and Colorni (1998); GRASP the **best** fitness of solution obtained by greedy randomized search procedure (GRASP) by Li, Pardalos and Resende (1994) as given in Maniezzo and Colorni (1998). The best solution among the three techniques is bolded.

Benchmark	Opt./BKS.	Best	Average	Std. Dev.	NFE	AS	GRASP
Bur26a	5426670	5426670	5426670	0.0	195794	5426670	5426670
Bur26b	3817852	3817852	3817852	0.0	31323	3817852	3817852
Bur26c	5426795	5426795	5426795	0.0	18675	5426795	5426795
Bur26d	3821225	3821225	3821225	0.0	18966	3821225	3821225
Bur26e	5386879	5386879	5386879	0.0	15483	5386879	5386879
Bur26f	3782044	3782044	3782044	0.0	6855	3782044	3782044
Bur26g	10117172	10117172	10117172	0.0	7131	10117172	10117172
Chr20a	2192	2192	2192	0.0	368109	2192	2232
Chr20b	2298	**2394**	2394	0.0	667998	2362	2434
Chr20c	14142	14142	14142	0.0	44248	14142	14142
Chr22a	6156	6156	6156	0.0	851158	6156	6298
Chr22b	6194	**6194**	6218.3	19.87	722198	6254	6354
Els19	17212548	17212548	17212548	0.0	5109	N/A	N/A
Esc32a	130	130	130.54	0.93	285046	130	132
Esc32b	168	168	168	0.0	288285	168	168
Esc32c	642	642	642	0.0	866492	642	642
Esc32d	200	200	200	0.0	765784	200	200
Esc32e	2	2	2	0.0	33	2	2
Esc32f	2	2	2	0.0	33	2	2
Esc32g	6	6	6	0.0	36	6	6
Esc64a	116	116	116	0.0	377798	N/A	N/A
Kra30a	88900	88900	88917.27	57.28	290167	88900	88900
Kra30b	91420	91420	91454.5	55.20	295429	91420	91710
Lipa20a	3683	3683	3683	0.0	14229	3683	3683
Lipa30a	13178	13178	13179.6	5.42	123704	13178	13178
Lipa40a	31538	**31538**	31593.18	123.22	124748	31859	31859
Nug20	2570	2570	2570	0.0	48307	2570	2570
Nug30	6124	6124	6125.8	2.08	485881	6124	6150
Scr20	110030	110030	110030	0.0	21265	110030	110030
Ste36a	9526	**9526**	9533.091	9.73	220754	9598	9698
Ste36b	15852	**15852**	15887.8	54.62	215278	15892	15998
Ste36c	8239110	**8239110**	8239187	253.87	135984	8265934	8312752
Tho30	149936	149936	149967.3	103.71	320404	149936	149936
Tho40	240516	**240632**	241338.2	838.35	177399	242108	243320
Sko56	34458	34502	34617.0	96.69	845275	N/A	N/A