An Approach to Solving Combinatorial Optimization Problems Using a Population of Reinforcement Learning Agents

Victor V. Miagkikh and William F. Punch III
Genetic Algorithms Research and Application Group (GARAGe)
Department of Computer Science and Engineering
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 an approach that uses reinforcement learning (RL) algorithms to solve combinatorial optimization problems. In particular, the approach combines both local and global search characteristics: local information as encoded by typical RL schemes and global information as contained in a population of search agents. The effectiveness of the approach is demonstrated on both the Asymmetric Traveling Salesman (ATSP) and the Quadratic Assignment Problem (QAP). These results are competitive with other well-known search techniques and suggest that the presented RL-agent approach can be used as a basis for global optimization techniques.

1 INTRODUCTION

Generate-and-test optimization algorithms are known to be efficient in finding optimal or near-optimal solutions in complex optimization problems. 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 particular free parameter. The quality of the generated solution 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 is 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 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 schemata in a GA has an implicit probability of appearing in generated solution where the better a schemata is, the higher the probability of it occurring in a solution. Such 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 schema grows exponentially with the size of the problem, maintaining values associated with each individual schemata becomes prohibitive. On the other hand, GA 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. To therefore 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. This problem is clearly seen in AS convergence on the TSP when all ants begin to follow the same tour.

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 looses Markovian 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 function-approximation and other means as will be discussed further.

The remainder of this paper will explore how to improve the two difficulties discussed in the application of RL techniques to optimization problems. In particular, we will show how to add global search characteristics to RL problem-solving by using a population of RL search agents and how to make the state information more effective for the purpose of capturing interdependencies.

2 GLOBAL OPTIMIZATION USING RL ALGORITHMS

One approach to capturing parameter interdependencies is the use of Bayesian approach. Integrating Bayes rule into RL allows one to compute the posterior probability of a particular assignment given that some assignments have already been made. However, even in the case of a naive Bayesian scheme, such an approach is expensive ($O(n^4)$ in space, $O(n^2)$ computational complexity of update rule in TSP and QAP). This approach is therefore only feasible for problem instances of moderate size unless used in combination with a function-approximation technique to reduce the space requirements. The authors describe this approach in detail in Miagkikh and Punch (1999).

This paper concentrates on another possibility: 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 its 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 preferences 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 preference 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. For example, if a city-city representation of the state-action pair is used, the assignments made via preferences assign values based on correlations with previous assignments. 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 the values of the state-action pairs. 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.

Another possible approach is to average the action-va

lues and use this average to make assignments. However, such averaging has a tendency to produce very poor results, which is more evidence for the stated principle of locality for action-values. 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 serach. 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.

Another reason for choosing this type of selection was mentioned earlier: each agent in the population is more like an independent search agent which occasionally exchanges results with other agents. Thus strong selection pressure, such as found in proportional selection, can be used. However, since offspring must replace a parent to be part of the population, crowding is reduced. That is, similar preferences and solution parts must be generated independently. This also means that local optimization is less risky because one solution cannot easily dominate the population.

We do not explicitly encode a mutation operation in this algorithm. By copying a large part of one solution, the whole process could be seen as a directed mutation, which allows extrication from local minima.

The overall architecture of the approach is depicted 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. The global 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.

3 APPLICATION: ATSP

ATSP is a classic *NP-hard* problem of finding the shortest Hamiltonian cycle in a complete weighted graph, where the weight matrix is not necessarily symmetric.

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 parents and filling in the remaining part using the action-values of another agent. An adaptation of the standard one-step Q-learning by Watkins and Dayan (1992) was used:

$$Q(c_n, c_{n+1}) = Q(c_n, c_{n+1}) + \\ + \alpha\left[r(n) + \gamma \max_a Q(c_{n+1}, a) - Q(c_n, c_{n+1}) \right]$$

where $Q(c_n, c_{n+1})$ reflects the desirability of choosing city c_{n+1} to follow c_n, a denotes all actions available in c_{n+1}. α and γ are learning and discounting parameters respectively. The reward r is a weighted combination of immediate and global rewards:

$$r(n) = \beta\left(\frac{L_{tour}}{N \cdot d(c_n, c_{n+1})} \right) + (1-\beta) R_{global}$$

where N is the size of the problem, L_{tour} is the length of the tour, $d(c_n, c_{n+1})$ is the cost of traveling from city c_n to c_{n+1}, R_{global} is the global reward, and β is a parameter controlling the balance between goodness of the action in local and global contexts. This update procedure with complexity $O(N^2)$ was run after each new tour was obtained. The value of global reward R_{global} was calculated on the basis of the average fitness of the parents:

$$R_{global} = \frac{avParentsFit - L_{tour}}{avParentsFit} \text{, where}$$

$$avParentsFit = \frac{L_{tour}(P1) + L_{tour}(P2)}{2}$$

Other ways of calculating global reward using the basis of the entire population are possible. For example, instead of the average fitness of the parents, the average fitness of the last M solutions can be used. This approach for

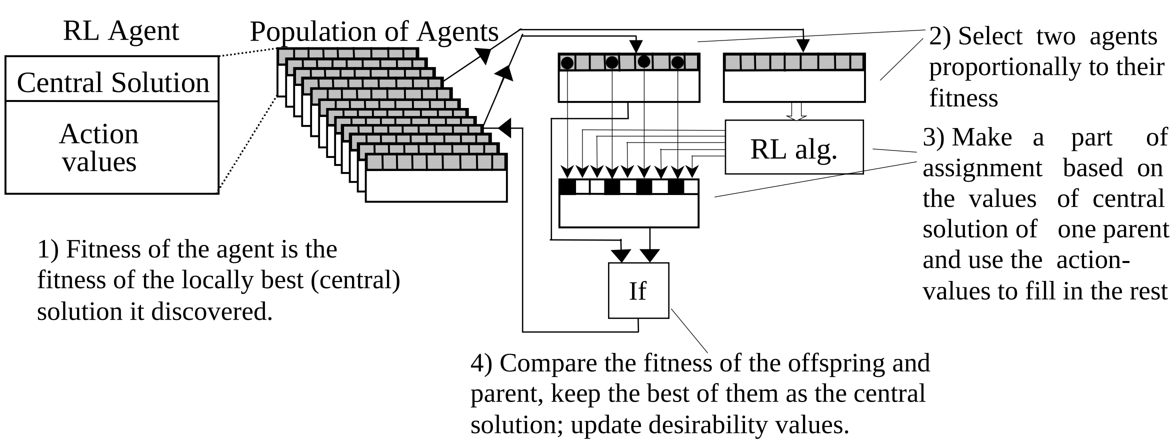

Figure 1: The structure of the search using a population of RL agents

calculating the baseline is smoother and has a slight advantage over basing the average fitness on two parents, but not enough to justify the increased computational efforts.

An individual step, that is, from city a choosing the next city b is based on the Q-values $Q(a,b_i)$, where b_i belongs to the set of not-yet-visited cities. Among different selection strategies tried, the authors chose a simple ε-greedy proportional policy: make the most desirable action with the probability ε, or with probability 1-ε choose one of the remaining options with probability proportional to the action-value estimate.

The balance between copying the fragments of the best solution and generating the rest using Q-values was 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 Q-learning to make all assignments. Since a feasible solution is a sequence of cities, there is an advantage to replicating adjacent values in groups rather than uniformly. To correspond with the definition of λ, k fragments of size s_i are randomly chosen for replication in the range [1,l]. The maximum length of the fragment, l, was set to $l = N/10$. The number of fragments k is calculated from equation $\sum_{i=1}^{k} s_i = \lambda N$. We also observed that if the value of λ is close to 1, the simpler approach described in the section on the QAP produces almost identical results.

After a new solution is generated, it passes through a relatively inexpensive local optimization procedure which, including the fitness evaluation, has a complexity of $O(N^2)$. One of the following two local optimization procedures was randomly selected each time: attempt to insert a randomly chosen city between all pairs of cities, or test order swaps of two random cities. The complexity of ATSP in comparison to symmetric TSP comes in part from the fact that in ATSP, exchange of any fixed number of cities requires $O(N)$ to reevaluate the solution, while in TSP it can be done in $O(1)$. This is one of the reasons ATSP is harder and more interesting than its symmetric case.

4 APPLICATION: QAP

The Quadratic Assignment Problem is a problem of finding a permutation φ minimizing:

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

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 action-values can estimate the goodness of assigning a specific location to some facility. The result of assigning a facility to a location is highly dependent on how other facilities are assigned.

Since there is no obvious order in which assignments should be made, it makes the application of bootstrapping 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 rule that does not require a particular order was used, at the price of slower convergence. In our application, the values of the estimates $p(l_i\ f_j)$ of assigning facility f_j to location l_i were trained by:

$$p(l_i, f_i)_{t+1} = p(l_i, f_i)_t + \alpha\left(r_{t+1} - p(l_i, f_i)_t\right)$$

where reward r_{t+1} was calculated as the global reward R_{global} in ATSP on the basis of the average fitness of two parents. The choice of the values for replication was also simplified since there is no adjacency relationship between the cells in a permutation. Each cell had a probability of being copied equal to λ / N and the remaining on average $N(1-\lambda)$ positions were filled using the values of $p(l_i\ f_j)$. The order of assigning facilities to locations was random each time. ε-greedy proportional selection was used to determine which facility from the set of not-yet-assigned facilities was to be assigned to a given location. Generated solutions were improved by a simple 1-Opt optimizer.

5 RESULTS

One of the interesting results regarding the usefulness of this approach is the quality of the search as controlled by the parameter λ, which controls the amount of replication. To show this relationship, we use one of the ATSP benchmarks (ftv44) and plot the average best value found against λ. The results were averaged over 10 runs and shown in Figure 2. The setting λ=0 corresponds to the generation of an entire solution from scratch based on Q-learning. As λ increases, the algorithm finds better solutions, and optimal performance was found in the range [0.75,0.95]. Further increases of λ lead to stagnation of the search due to lowered variability. The average number of function evaluations required to obtain a tour with fitness 1650 or lower depending on the value of λ is shown in Figure 3, curve 1. Furthermore, a low value of λ resulted in increased computational effort to reach a similar level of performance. As λ was increased, the curve got almost flattened in the range [0.4, 0.85]. Further increase of λ resulted in a growing amount of computational effort to reach similar performance.

Curve 2 in Figure 3 shows the average number of function evaluations required to reach the optimum. The value of λ=0.9, which corresponds on average to 4-5 modifications in problem instances of this size, required the least number of function evaluations to find the optimum each time, out of ten runs. Additional research on a number of different benchmarks is required to make general conclusions about the optimal value of λ, but these graphs show that generating an entire solution from scratch or exhaustive replication results in a performance decrease.

The setup for experimental runs for the QAP and ATSP were similar. The population consisted of 50 agents, which is relatively small for GA, but was enough to obtain good results with the approach presented. 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, global/local, discounting, and selection greediness parameters were in ranges the [0.005,0.015] for α, [0.5,1.0] for β, [0.8,1.0] for γ, and [0.4,0.95] for ε respectively. These ranges of parameters were determined by a series of preliminary experiments similar to those on the value of λ, (not described). 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 random selection of the various control parameters.

Results based on the average of 10 runs over each of the benchmark problems from QAPLIB by Burkard *et al.* (1997) and TSPLIB by Reinelt (1991) are given in the Tables 1 and 2, respectively. Since the presented 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 (1997a) respectively. In ATSP, we compare the results obtained by pure AS by Gambardella and Dorigo (1996), MAX-MIN AS with local search (MMAS) due to Stützle and Hoos (1997) and GA with local search (GA+LS) by Merz and Freisleben (1997b).

The RL-agents approach achieves the same or better results on all test problems in comparison to AS-based algorithms. In comparison to GA+LS, the approach presented showed results that 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: 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 a more detailed comparison.

6 CONCLUSIONS AND FUTURE WORK

Our results are competitive with the other search techniques which suggests that adapted RL algorithms deserve more attention as a search paradigm. The presented approach 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 is to show that RL algorithms provide 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

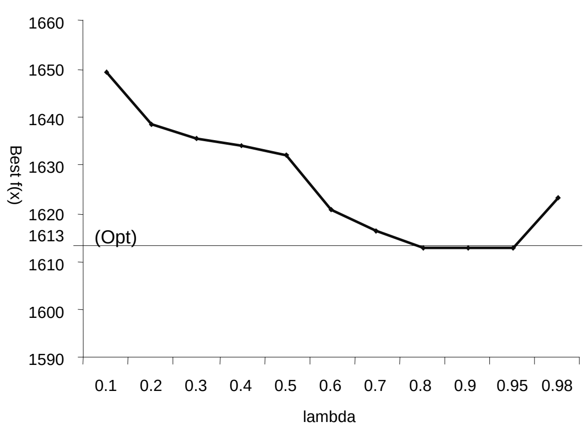

Figure 2: Average best value in ftv44 depending λ

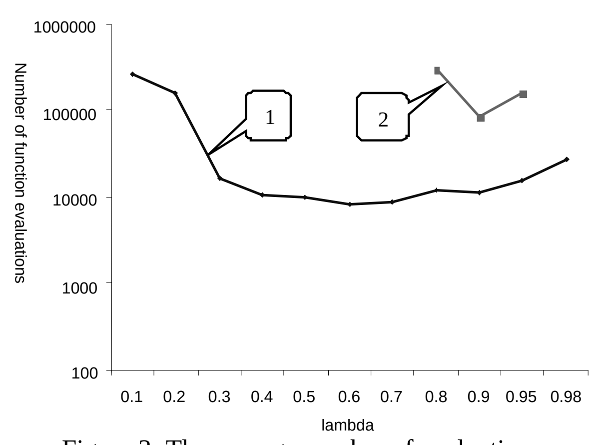

Figure 3: The average number of evaluation computations to reach the fitness 1650 (curve 1) and optimum (curve 2) in ftv44 depending on λ

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 ($O(n^4)$) requires function approximation. However, in spite of all these difficulties, the results are very encouraging.

References

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

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

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

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

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

C. Watkins, P. Dayan (1992). Q-learning. *Machine Learning*, 8:279-292, Kluwer Academic Publishers.

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

W. Zhang and T. Dietterich (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,

S. Singh and D. Bertsekas (1996). Reinforcement Learning for Dynamic Channel Allocation in Cellular Telephone Systems. In Proc. of *Advances in Neural Information Processing Systems*, 974-980, MIT Press.

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

J. Grefenstette *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.

J. Boyan, and A. Moore (1998). "Learning Evaluation Functions for Global Optimization and Boolean Satisfiability", *Fifteenth National Conference on Artificial Intelligence*, AAAI.

V. Miagkikh and W. Punch (1999). "Global Search in Combinatorial Optimization using Reinforcement Learning Algorithms", To appear *in Proc. of 1999 Congress on Evolutionary Computations*.

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

G. Reinelt (1991). TSPLIB-A Traveling Salesman Problem Library. *ORSA Journal on Computing*, 3(4): 376-384.

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

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

L.Gambardella and M. Dorigo (1996). Solving Symmetric and Asymmetric TSPs by Ant Colonies. in Proc. of *IEEE Conf. on Evolutionary Computation*, 622-627, IEEE Press.

T. Stützle and H. Hoos (1997). MAX-MIN Ant System and Local Search for the Traveling Salesman Problem. In Proc. of *1997 IEEE 4th Int. Conf. On Evolutionary Computation*, 308-313, IEEE Press.

P. Merz and B. Freisleben (1997b). Genetic Local Search for the TSP: New Results. In Proc. of *the 1997 IEEE Int. Conf. on Evolutionary Computation*, 159-164, IEEE Press.

Appendix

Table 1: Results 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 (1997a). 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

Benchmark	Opt./BKS.	Best	Average	Std. Dev.	NFE	AS	GA+LS
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
Els19	17212548	17212548	17212548	0.0	8375	N/A	N/A
Esc32a	130	130	130	0.0	264	130	N/A
Esc32b	168	168	168	0.0	264	168	N/A
Esc32c	642	642	642	0.0	264	642	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
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
Sko42	15812	15812	15812	0.0	1352249	N/A	N/A
Sko49	23386	23386	23397.8	6.21	3539170	N/A	N/A
Sko56	34458	34458	34465.2	6.87	6180615	N/A	N/A
Sko64	48498	48498	48513.8	13.11	5577520	N/A	N/A
Sko72	66256	66324	66355.6	21.11	7702169	N/A	N/A
Sko81	90998	91090	91186.8	43.34	2759429	N/A	N/A
Sko90	115534	115782	115863.4	40.49	2517729	N/A	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
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 on TSPLIB by Reinelt (1991).

The meaning of the columns 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 presented algorithm 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 average fitness of the best solution obtained by the AS in Gambardella and Dorigo (1996); MMAS - the average fitness of the best solution found by MAX-MIN AS described in T. Stützle, H. Hoos (1997); GA+LS the average fitness of solution obtained by GA with local search by P. Merz, B. Freisleben (1997b). The best solution among the three techniques is bolded. The only available result for Ant-Q by L. Gambardella, M. Dorigo (1995) on the problems from TSPLIB is the ry48p benchmark. Average: 14690, best: 14422.

Benchmark	Opt./BKS.	Best	Average	Std. Dev.	NFE	AS	MMAS	GA+LS
Ft53	6905	6905	6905	0.0	101181	N/A	N/A	N/A
Ft70	38673	38673	**38674.0**	8.50	10443706	39099.1	38707	38674.2
Ftv44	1613	1613	1613	0.0	971701	N/A	N/A	N/A
Ftv47	1776	1776	1776	0.0	52792	N/A	N/A	N/A
Ftv55	1608	1608	1608	0.0	91670	N/A	N/A	N/A
Ftv64	1839	1839	1839	0.0	51421	N/A	N/A	N/A
Ftv70	1950	1950	1950	0.0	148583	N/A	N/A	N/A
Ftv170	2755	2810	2824.7	16.13	6830936	2826.5	2807	**2762.2**
Kro124p	36230	36230	36263.3	59.78	7934886	36857.0	36655	**36231.5**
P43	5620	5620	**5620**	0.0	79676	N/A	5623.8	5620.1
Rbg323	1326	1342	1350.4	5.68	3117520	N/A	N/A	N/A
Rbg358	1163	1181	1190.5	5.81	3182564	N/A	N/A	N/A
Rbg403	2465	2465	2466.3	1.86	10536418	N/A	N/A	N/A
Rbg443	2720	2720	2722.8	3.55	12116328	N/A	N/A	N/A
Ry48p	14422	14422	**14422**	0.0	51947	14565.45	14494	14451.2