N94-34045

Reinforcement Learning in Scheduling

Tom G. Dietterich* DoKyeong Ok Wei Zhang Prasad Tadepalli

Department of Computer Science
Oregon State University
Corvallis, Oregon 97331-3202

Abstract

The goal of this research is to apply reinforcement learning methods to real-world problems like scheduling. In this preliminary paper, we show that learning to solve scheduling problems such as the Space Shuttle Payload Processing and the Automatic Guided Vehicle (AGV) scheduling can be usefully studied in the reinforcement learning framework. We discuss some of the special challenges posed by the scheduling domain to these methods and propose some possible solutions we plan to implement.

1 Introduction

Scheduling is a well-studied problem and there are a variety of scheduling problems [BAKER74, FOX87, SADEH91]. Unfortunately, almost all of the nontrivial versions of the scheduling problem are at least NP-hard [GAREY79]. However, since the scheduling problems that occur in the real world may not be entirely arbitrary, there is reason to believe that there exists some structure or regularity which may not be directly apparent. The goal of this research is to build learning systems that are capable of discovering such hidden regularities in the environment of the scheduler and exploit them to efficiently build reasonably good schedules.

We are currently focusing on two specific scheduling problems. One is the problem of Space Shuttle Payload Processing [ZWEBEN92]. The goal in this domain is to schedule a set of tasks so that they finish before their respective deadlines while obeying a given set of precedence and resource constraints. The second application domain is Automatic Guided Vehicle (AGV) scheduling. The problem here is to make an on-line assignment of transportation tasks to AGVs such that the average expected cost of transport over the long run is minimized. The transportation tasks are randomly generated, and need to be serviced on-line and hence the system cannot plan the order of task execution in advance.

These two application domains are somewhat different in that the first one emphasizes the problem of efficiently searching the scheduling space, and the second one emphasizes optimal real-time decision making in a stochastic environment. Nevertheless, the same basic approach of reinforcement learning is applicable in both cases.

Reinforcement learning is an unsupervised learning method where an agent learns from the feedback or 'reinforcement' provided by the environment as a result of its actions. The reinforcement can be positive (reward) or negative (punishment) and can be delayed in time from the action that is responsible for it. The goal of the learner is to learn to behave in such a way that maximizes its total expected reward.

*The first and the last authors are supported by a grant from NASA, number 1293.

It is easy to see the correspondence between the AGV scheduling and reinforcement learning. Assume that the AGV gets several requests to transport objects from one place to another. The AGV might choose to serve these requests in some order. Whenever a request is served, the AGV gets a reinforcement which might be inversely proportional to the time delay with which the request is served. The problem then is to learn an optimal policy (a mapping from states of the world to AGV actions) that results in the maximum expected reward rate.

Although less straightforward, it is also possible to frame the space shuttle scheduling problem in the reinforcement learning context. In this context, the scheduler takes abstract scheduling actions such as moving tasks from one time slot to another, and from one machine to another. It gets a negative reinforcement proportional to the cost of the final conflict-free schedule. If the scheduler is able to correctly predict the cost of the final conflict-free schedule from intermediate states, then the search complexity can be reduced by choosing the scheduling action that will minimize the final cost of the schedule. The problem thus reduces to learning to predict the final cost of schedule from intermediate states with scheduling conflicts.

Many of the reinforcement learning methods are based on on-line versions of dynamic programming. In dynamic programming, the cost or value of a state is computed by backing up the costs of the succeeding states. Instead of computing the value of a state once and for all from the values of all its neighbors, in reinforcement learning, the value of a state is incrementally updated, as and when its neighbor's values are updated. Both the on-line and off-line versions of dynamic programming store a table of state-value pairs. The on-line version uses this table to choose an action that minimizes the expected cost of the final state.

One of the problems with the table-based reinforcement learning techniques like Q-learning is their space complexity [WATKINS92]. In the worst case, they need tables as big as the entire state space and some more, which is unrealistic in nothing but the trivial of domains. This also translates to very slow convergence of learning, because most states in the state space have to be updated many times before the learner's actions converge to optimal performance.

Reinforcement learning researchers use supervised learning methods to store the tables compactly and to converge quickly to the correct policy. For example, the table of state-value pairs can be used to train a neural net which can then predict the values of states that have not been stored. Similarly we can also consider storing the state-value pairs without generalizing them and use approaches like the nearest neighbor algorithms to predict the values for the unseen states by interpolating between the stored states which are nearest to the unseen state. Another approach would be to approximate the state-value table by a set of peiecewise polynomial functions using methods like spline interpolation. Such "structural generalization" methods give rise to a compact storage of states as well as a quicker convergence when the function they are trying to approximate suits their structure.

One of the problems in reinforcement learning is trading off exploration with optimal decision making. Exploration facilitates learning new knowledge while optimal decision making exploits old knowledge. However, most widely studied exploration strategies are random. We plan to investigate more sophisticated strategies that explore "near miss" states. These strategies seem effective for learning piece-wise polynomial functions with many locally irrelevant features.

The rest of the paper is organized as follows. The next section introduces reinforcement learning. Section 3 introduces the NASA space shuttle payload processing domain and puts it in the framework of reinforcement learning. Section 4 does the same for the AGV scheduling domain. Section 5 discusses the structural generalization problem in reinforcement learning and proposes a number of solutions that we plan to implement. Section 6 discusses some of the other challenges that the scheduling domain offers to reinforcement learning, and some proposed solutions. The last section concludes with a summary.

2 Reinforcement Learning

Reinforcement learning is best suited to a class of stochastic optimal control problems called Markovian decision problems [BARTO93].

The reinforcement learning problem can be described by a 4-tuple $\langle S, A, p, C\rangle$. S is a finite set of states and A is the set of actions. $p_{i,j}(a)$ is the probability for every state pair (i,j) and action $a \in A$, that executing action a in state i results in state j. The Markovian assumption means that this probability is independent of the states before i. $c_t = C(i_t, a_t)$ denotes the immediate cost or reward of executing an action a_t in a state i_t at time t. In some versions of the problem, when the horizon is not finite, the cost or reward is discounted by multiplying it with a discount factor γ^t. where

A policy μ is a mapping from the state i_t at time t to a recommended action in A. f^μ denotes the expected value of the infinite-horizon discounted cost, given by:

$$f^\mu(i_t) = E_\mu\left[\Sigma_{j=0}^{\infty}\gamma^{t+j}c_{t+j}\right]$$

where E_μ is the expectation assuming that the controller always follows policy μ. An optimal policy μ^* is one that minimizes f^μ, and f^* is its expected discounted cost.

Knowing f^* would allow one to construct μ^*, because, by the results of dynamic programming, any policy which is greedy (chooses the action that results in the least cost state) with respect to f^* is optimal.

Various versions of dynamic programming (DP) compute f^* by backing up costs from the last state to previous states in different orders [BARTO93]. Reinforcement learning methods use on-line versions of dynamic programming. Some of these methods, including Q-learning, do not assume that the transition probability matrix p is known. Q-learning eliminates the need for separately learning the p matrix, by learning a so called Q-function from state action pairs to the expected discounted costs of taking that action in that state. In particular, if i_t is the current state and a_t is the action taken, then

$$Q(i_t, a_t) = \Sigma_{j=0}^{\infty}\gamma^j c_{t+j}$$

Since the value of a state $V(i)$ is the value of the best action in that state, we can write

$$V(i_t) = \min_a Q(i_t, a)$$

and, it follows that

$$Q(i_t, a_t) = c_t + \gamma V(i_{t+1})$$

In Q-learning, the Q values of the final "absorbing states" can be immediately calculated, which are backed up from last to first for the states the system has passed through. More formally, the Q-function is computed in stages as follows. If S_k is the sequence of observed states and actions $(s_1, a_1, s_2, \ldots)$ at stage k, and α is the learning rate, Q_k is updated for states in S_k from last to first as follows:

$$Q_k(i_t, a_t) = (1-\alpha)Q_{k-1}(i_t, a_t) + \alpha(c_t + \gamma V_{k-1}(i_{t+1})),$$
$$\text{where, } V_k(i_t) = \min_a Q_k(i_t, a)$$

The theoretical results show that if the system explores every state infinitely often, then it eventually converges to the optimal Q values for all the state action pairs.

3 The Space Shuttle Payload Processing

The NASA Space Shuttle Payload Processing domain is an example of Job Shop Scheduling problem [ZWEBEN92]. Each 'mission' consists of a set of payload/carrier pairs, and a launch date. Each carrier requires a distinct set of tasks to prepare and process the payloads for a mission. The tasks are constrained by precedence and resource relationships. The resources are grouped into resource pools. The goal is to schedule all the tasks needed to load the carriers onto the orbiter, avoiding the resource contention problems, satisfying all precedence constraints while minimizing the total expected length of the schedule. More details can be found in [ZHANG93].

The Space Shuttle Payload Processing problem can be viewed as a state space search problem, where states are partial schedules with possible conflicts, and operators move from state to state by moving tasks. The problem is to find a conflict-free schedule of minimum length. Unlike in some other domains, there is a lot of flexibility in defining the operators in the scheduling domain. Individual tasks can be moved by a constant amount, or by an amount that depends on the availability of resources and the schedule of other tasks, or groups of tasks can be rescheduled using a single operator. One could also consider a hierarchy of abstract to more primitive operators. We plan to experiment with all these different options.

The search control knowledge for scheduling is expressed as an evaluation function that estimates the discounted final cost of the schedule reachable from the current state. In reinforcement learning methods like $TD(\lambda)$, this amounts to an estimate of f^* [SUTTON88]. Q-learning estimates it from the state that results by applying a scheduling action to the current state.

If the evaluation function is accurate, then it can be used to select the action that leads to the state with the least cost without search. When the evaluation function is only approximate, as is likely to be the case in complex domains like scheduling, it can be combined with look ahead search, as done in 2-person games like chess, to exploit the benefits of both knowledge and search.

4 The AGV Scheduling Problem

Automatic Guided Vehicles or AGVs are increasingly being used in manufacturing plants to cut down the cost of human labor in transporting materials from one place to another. Optimal scheduling of AGVs is a non-trivial task. In general, there can be multiple AGVs, with some routing constraints, e.g., two AGVs cannot be on the same route fragment going in opposite directions. The AGVs might also have capacity constraints such as the total load and volume they can carry, and the total time they can work without recharging.

The transportation requests are stochastic and hence cannot be planned for in advance. The AGV gets a reward whenever it successfully serves a request. The behavior of the AGV is random in the beginning, but as it accumulates knowledge of the request patterns and the transportation costs involved, we expect it to perform better in the sense of serving more requests in a given time. In the reinforcement learning context, this corresponds to maximizing the average reward per unit time rather than maximizing the discounted reward. We can also associate a non-uniform reward structure with the requests and give more reward for serving some requests and not the others.

A learning AGV is very attractive in a manufacturing plant because the scheduling environment is constantly changing and it is hard to manually optimize the scheduling algorithm to each changed situation. A learning AGV would automatically adapt itself changes in its environment, be they are added machines, changes in the AGV routes, or changes in the request rates and patterns.

Once again, we treat the AGV scheduling as a state space search, and treat the status of various requests and the AGV as a state. The best action to take at any time depends on the current state. The optimal policy can be learned using methods like Q-learning.

5 Structural Generalization in Reinforcement Learning

One of the major issues in reinforcement learning is that of structural generalization. This can also be seen as choosing a representation for the evaluation function. An extreme representation is as a table of mappings from state descriptions to their evaluations. This amounts to no structural generalization at all, since no two state descriptions will have the same entry in the table. Representing it as a table of mappings is infeasible in many scheduling domains due to their size and slow convergence. One of the requirements is also that real-valued functions should be representable.

One of the popular representations of evaluation functions is neural nets. Recently a program that used TD(λ) in combination with neural nets to represent its evaluation function to learn to play Backgammon reached grand-master level performance and is ranked as the best Backgammon program in the world [TESAURO92]. The success of neural net learning crucially depends on being able to find good encodings for states and operators.

Another reasonable representation is piece-wise polynomial functions. These functions can be learned by nearest neighbor algorithms. In the extreme version of these algorithms, no generalization is necessary. Each example is stored as an input-output pair. To predict the output for an unseen example, its nearest (using some distance metric like the Euclideanm distance) K neighbors are examined and the output is calculated to be the weighted sum of their outputs. One of the disadvantages of this approach is that it needs a lot of memory to store all the examples, and hence the name "memory-based" [MOORE93]. An optimization that is usually done is to store an example only when the current set of examples does not make a correct prediction for this example.

6 Research Issues

The scheduling domain offers some interesting challenges to reinforcement learning.

One of the complexities of the scheduling problems like the space shuttle scheduling is that they have a large number of applicable operators at any state leading to a search space with a large branching factor. When the branching factor is large, it is not realistic to choose the best action by trying each possible action and comparing the results, as the standard greedy policy adopted in reinforcement learning methods usually does. In this case, we can use a random sample greedy strategy which chooses the best action from among a randomly sampled subset of all possible actions. We also plan to experiment with methods such as simulated annealing and gradient descent search, which do not involve testing each possible next state.

There are also usually a large number of irrelevant features in the state description of the scheduling problems. The presence of irrelevant features makes it difficult to generalize correctly. For example, in the nearest neighbor approach, the irrelevant features distort the distances between examples so that examples which are close in relevant features appear distant with irrelevant features and vice versa. Since the number of examples needed to converge in this approach varies exponentially with the number of features, a large number of irrelevant features works against this approach [AHA91]. Adjustable feature weights has been proposed as a solution for this problem [AHA91]. While this method eliminates globally irrelevant features, one problem with it is that it does not take local irrelevancy into account.

One way to determine local irrelevancy is through intelligent exploration. Most reinforcement learning methods use random exploration to gain new knowledge about their search spaces. However, if one knows that the evaluation function has certain structure, say that it is representable as a piece-wise polynomial function with many locally irrelevant features, then it may be possible to explore this search space more intelligently. For example, it may be possible to determine locally irrelevant features by generating "near miss" examples, which are examples which differ from a base example by exactly one feature in a small amount, but change the value of the function. The existence of a near miss example in a feature shows the local relevance of that feature. Generating near miss examples and determining the values of the evaluation function at these examples add the ability of intelligent exploration to reinforcement learning.

7 Conclusions

In this paper, we suggested that reinforcement learning can be usefully employed in scheduling domains to learn search control knowledge as well as to learn to do optimal real-time scheduling. The work reported here is preliminary and much remains to be done. We plan to implement the ideas reported in this paper, test them and report the results in the near future.

References

[AHA91] Aha, D. W., Kibler, D., and Albert, M. K. Instance-based learning algorithms. *Machine Learning, 6* (1), 37–66. 1991.

[BAKER74] Baker, K.R., Introduction to Sequencing and Scheduling, John Wiley & Sons, Inc., 1974.

[BARTO93] Barto, A. G., Bradtke, S. J., and Singh S. P. Learning to Act using Real-Time Dynamic Programming. To appear in *AI Journal* special issue on Computational Theories of Interaction and Agency.

[FOX87] Fox, M. S. *Constraint-Directed Search: A Case Study of Job-Shop Scheduling.* Morgan Kaufmann, Los Altos, CA, 1987.

[GAREY79] Garey M.R., and Johnson, D.S. *Computers and Intractability: A Guide to the Theory of NP-Completeness,* W.H. Freeman and Company, New York, 1979.

[MOORE93] Moore, A.W. and Atkeson, C. G. Memory-based Reinforcement Learning: Converging with less data and less real time. (to appear).

[SADEH91] Sadeh, N. Look-ahead Techniques for Micro-opportunistic Job Shop Scheduling. Ph.D. Thesis, School of Computer Science, Carnegie Mellon University, available as CMU-CS-91-102, 1991.

[SUTTON88] Sutton, R. S. Learning to Predict by the Methods of Temporal Difference. In *Machine Learning,* 3, 9-44, 1988.

[TESAURO92] Tesauro, G. Temporal Difference Learning of Backgammon Strategy. In *Proceedings of Machine Learning Conference,* 1992.

[WATKINS92] Watkins, C. J. C. H. and Dayan P. Q-learning. *Machine Learning,* 8:279-292, 1992.

[ZHANG93] Zhang, W. A Study of Reinforcement Learning for Job-shop Scheduling. Ph.D. Thesis proposal, Oregon State University, Corvallis, OR, 1993.

[ZWEBEN92] Zweben, M., Davis, E. Daun, B., Drascher, E., Deale, M., and Eskey, M. Learning to improve constraint-based scheduling. *Artificial Intelligence,* 58:271-296, 1992.