Applying Online Search Techniques to Reinforcement Learning

Scott Davies Andrew Y. Ng Andrew Moore*
School of Computer Science
Carnegie-Mellon University
Pittsburgh, PA 15213

Abstract

In reinforcement learning it is frequently necessary to resort to an approximation to the true optimal value function. Here we investigate the benefits of online search in such cases. We examine "local" searches, where the agent performs a finite-depth lookahead search, and "global" searches, where the agent performs a search for a trajectory all the way from the current state to a goal state. The key to the success of these methods lies in taking a value function, which gives a rough solution to the hard problem of finding good trajectories from every single state, and combining that with online search, which then gives an accurate solution to the easier problem of finding a good trajectory *specifically* from the current state.

1 Introduction

A common approach to Reinforcement Learning involves approximating the value function, and then executing the greedy policy with respect to the learned value function. But particularly in continuous high-dimensional state spaces, it is often the case that even when our agent is given a perfect model of the world, it can be computationally expensive to fit a highly accurate value function. This problem is even worse when the agent is learning a model of the world and is repeatedly updating its dynamic programming solution online. What's to be done?

In this paper, restricted to deterministic domains, we investigate the idea that rather than executing the greedy policy with respect to the approximated value function, we can utilize search techniques to find a better trajectory. Briefly, we may perform a "local" search for a "good" short multi-step path, or a "global" search for a trajectory all the way from the current state to a goal state, and where the search

*all email addresses are Firstname.Lastname@cs.cmu.edu

may be guided by the learned value function. With the searches, we perform online computation that is directed towards finding a good trajectory *from the current state*; this is in contrast to, say, offline learning of a value function, which tries to solve the much harder problem of learning a good policy for every point in the state space.

2 Local Search

Given a value function, agents typically execute a greedy policy using a one-step lookahead search, possibly using a learned model for the lookahead. The computational cost per step of this is $O(|A|)$ where A is the set of actions. This can be thought of as performing a depth 1 search for the 1-step trajectory T that gives the highest $R_T + \gamma V(s_T)$, where R_T is the reinforcement, s_T is the state (possibly according to a learned world model) reached upon executing T, and γ is the discount factor. A natural extension is then to perform a search of depth d, to find the trajectory that maximizes $R_T + \gamma^d V(s_T)$, where discounting is incorporated in the natural way into R_T. The computational expense is $O(d|A|^d)$.

To make deeper searches computationally cheaper, we might consider only a subset of these trajectories. Especially for dynamic control, often an optimal trajectory repeatedly selects and then *holds* a certain action for some time, such as suggested by [2]; a natural subset of the $|A^d|$ trajectories, therefore, are trajectories that switch their actions rarely. When we constrain the number of switches between actions to be s, the time for such a search is then $O(d \binom{d}{s} |A|^{s+1})$–considerably cheaper than a full search if $s \ll d$. We also suggest that s is easily chosen for a particular domain by an expert, by asking how often action switches can reasonably be expected in an optimal trajectory, and then picking s accordingly to allow an appropriate number of switches in a trajectory of length d. Figure 1 shows a search performed in the MODIFIED-CAR task (described later) using $d = 20$ and $s = 1$.

During execution, the local search algorithm iteratively finds the best trajectory T with the search algorithm above, executes the first action on that trajectory, and then does a new search from the resulting state. If B is the "parallel backup operator" [1] so that $BV(s) = max_{a \in A} R(s, a) + V(\delta(s, a))$, then executing the full $|A|^d$ search is formally equivalent to executing the greedy policy with respect to the value function $B^{d-1}V$. Noting that, under mild regularity assumptions, as $k \to \infty$, $B^k V$ becomes the optimal value function, we can generally expect $B^{d-1}V$ to be a better value function than V. For example, in discounted problems, if the largest absolute error in V is ε, the largest absolute error in $B^{d-1}V$ is $\gamma^{d-1}\varepsilon$. Lastly, for well chosen s, we expect the cheaper, restricted search to approximate this full search well.

This approach, a form of receding horizon control, has most famously been applied to minimax game playing programs [8] and has also been used in single-agents on small discrete domains (e.g. [4]).

3 Global Search

Local search is not the only way to improve an approximate value function. Here, we describe global search for solving least-cost-to-goal problems in continuous state spaces with non-negative costs. We assume the set of goal states is known.

3.1 Uninformed Global Search

Instead of searching locally, why not continue growing a search tree until it finds a goal state? The answer is clear—the combinatorial explosion would be devastating. In order to deal with this problem, we borrow a technique from robot motion planning [5]. We first divide the state space up into a fine uniform grid. A local search procedure is then used to find paths from one grid element to another. Multiple trajectories entering the same grid element are pruned, keeping only the least-cost trajectory into that grid element (breaking ties arbitrarily). The rationale for the pruning is an assumed similarity between among points in the same grid element. In this manner, the algorithm attempts to builds a complete trajectory to the goal using the learned or provided world model. A graph showing a such a search for the MODIFIED-CAR domain (to be described shortly) is depicted in Figure 4. It is worth noting that Uninformed Global Search is equivalent to Informed Global Search (described below) using a constant 0 heuristic evaluation function.

When the planner finds a trajectory to the goal, it is executed in its entirety. But in the case where we are learning a model of the world, it is possible for the agent to successfully plan a continuous trajectory using the learned world model, but to fail to reach the goal when it tries to follow the planned trajectory. In this case, failure to follow the successfully planned trajectory can directly be attributed to inaccuracies in the agent's model; and in executing the path anyway, the agent will naturally reach the area where the actual trajectory diverges from the predicted/planned trajectory and thereby improve its model of the world in that area.

3.2 Informed Global Search

We can modify Uninformed Global Search by using an approximated value function to *guide* the search expansions in the style of A^* search [7], (written out in detail below). With the perfect value function, this causes the search to traverse exactly the optimal path to the goal; with only an approximation to the value function, it can still dramatically reduce the fraction of the state space that is searched.

The search uses a sparse representation so that only grid cells that are visited take up memory[1]. Notice also that like Local Search, we are performing online computation in the sense that we are performing a search only when we know the "current state," and to find a trajectory specifically from the current state; this is in contrast to offline computation for finding a value function, which tries to solve the much more difficult problem of finding a good trajectory to the goal from every single point in the state space.

Writen out in full, the search algorithm is:

1. Suppose $g(s_0)$ is the grid element containing the current state s_0. Set $g(s_0)$'s "representative state" to be s_0, and add $g(s_0)$ to a priority queue P with priority $V(s_0)$, where V is an approximated value function.
2. Until a goal state has been found, or P is empty:
   - Remove a grid element g from the top of P. Suppose s is g's "representative state."
   - Starting from s, perform a "local" search as in Section 2, except search trajectories are pruned once they reach a state in a different grid g'. If g' has not been visited before, add g' to P with a priority $p(g') = R_T(s_0, \ldots, s') + \gamma^{|T|}V(s')$, where R_T is the reward accumulated along the recorded trajectory T from s_0 to s', and set g''s "representative state" to s'. Similarly, if g' has

[1] This has a flavor not dissimilar to the hashed sparse coarse encodings of Sutton [9].

been visited before, but $p(g') \leq R_T(s_0, \ldots, s') + \gamma^{|T|} V(s')$, then update $p(g')$ to the latter quantity and set g''s "representative state" to s'. Either way, if g''s "representative state" was set to s', record the sequence of actions required to get from s to s', and set s''s predecessor to s.

3. If a goal state has been found, execute the trajectory. Otherwise, the search has failed, because our grid was too coarse, our state transition model inaccurate, or the problem insoluble.

An example of the search performed by this algorithm on the MODIFIED-CAR domain is shown in Figure 5. The value function was approximated with a simplex-based interpolation [3] on a coarse 7 by 7 grid, with all other parameters the same as in Figure 4. Much less of state space is searched than by Uninformed Global Search.

4 Experiments

We tested our algorithms on the following domains[2]:

- MODIFIED-CAR (2 dimensional): A car has to reach and park at the top of a one dimensional hill; the car needs to back up first in order to gather enough momentum to get to the goal. Note this is slightly more difficult than the normal mountain car, as we require a velocity near 0 at the top of the hill [6]. State consists of x-position and velocity. Actions are accelerate forward or backward.
- ACROBOT (4 dimensional): An acrobot is a two-link planar robot acting in the vertical plane under gravity with only one weak actuator at its elbow joint. The goal is to raise the hand at least one link's height above the shoulder [9]. State consists of joint angles and angular velocities at the shoulder and elbow. Actions are positive or negative torque.
- MOVE-CART-POLE (4 dimensional): A cart-and-pole system starting with the pole upright is to be moved some distance to a goal state, keeping the pole upright (harder than the stabilization problem). It terminates with a huge penalty (-10^6) if the pole falls over. State consists of the cart position and velocity, and the pole angle and angular velocity. Actions are accelerate left or right.
- SLIDER (4 dimensional): Like a two-dimensional mountain car, where a "slider" has to reach a goal region in a two-dimensional terrain. The terrain's contours are shown in Figure 2. State is two dimensional position and two dimensional velocity. Actions are acceleration in the NE, NW, SW, or SE directions.

All four are undiscounted tasks. MOVE-CART-POLE's cost on each step is quadratic in distance to goal. The other three domains cost a constant -1 per step. All results are averages of 1000 of trials with a start state chosen uniformly at random in the state space, with the exception of the MOVE-CART-POLE, in which only the pole's initial distance from its goal configuration is varied. The algorithm we used to learn a value function is the simplex-interpolation algorithm described in [3]. But note that the choice of a function approximator is orthogonal to the search techniques we describe here, which can readily be applied to any other value function computed by any other algorithm, such as an LQR solution to a linearized problem or a neural net value function computed with TD.

[2] C code for all 4 domains (implemented with numerical integration and smooth dynamics) will shortly be made available on the Web.

For now, we consider only the case where we are given a model of the world, and leave the model-learning case to the next section.

4.1 Local Search

Here, we look at the effects of different parameter settings for Local Search. We first consider MOVE-CART-POLE. Empirically, a good trajectory "switches" actions very often, and we therefore chose not to assume much "action-holding," and set $s = d - 1$. The approximate value function was found using a four-dimension simplex-interpolation grid with quantization 13^4, which is about the finest resolution simplex-grid that we could reasonably afford to use. As we increase the depth of the search from 1 (greedy policy with respect to V) up to 10 (greedy policy with respect to B^9V), we see that performance is significantly improved, but with CPU time per trial (on a 100MHz HP C300 9000, given in seconds) increasing exponentially:

d	1	2	3	4	5	6	7	8	9	10
cost	49994	42696	31666	14386	10339	27766	11679	8037	9268	10169
time	0.66	0.64	1.24	1.02	1.13	2.07	3.32	3.84	7.30	15.50

The next experiment we consider here is MODIFIED-CAR on a coarse (7^2) grid. Empirically, entire trajectories (of > 100 steps) to the goal can often be executed with 2 or 3 action switches, and the optimal trajectory to the goal from the bottom of the hill at rest requires only about 3 switches. Thus, for the depth of searches we performed, we very conservatively chose $s = 2$. Experimental results again show solution quality significantly increased by Local Search, but with running times growing much slower with d than before:

d	1	2	3	4	6	8	12	16	24
cost	187	180	188	161	140	133	133	134	112
time	0.02	0.05	0.10	0.16	0.36	0.70	2.08	4.62	12.44

4.2 Experimental Results

The table below summarizes our experimental results[3]. "cost" is average cost per trial, "time" is average CPU seconds per trial, and #LS is the average number of local searches performed by the global search algorithms (which indicates the amount of state space considered).

	No Search		Local Search		Uninformed Global			Informed Global		
	cost	time	**cost**	time	**cost**	#LS	time	**cost**	#LS	time
MODIFIED-CAR	**187**	0.02	**140**	0.36	**FAIL**	–	–	**151**	259	0.14
ACROBOT	**454**	0.10	**305**	1.2	**407**	14250	5.8	**198**	914	0.47
MOVE-CART-POLE	**49993**	0.66	**10339**	1.13	**3164**	7605	3.45	**5073**	1072	0.64
SLIDER	**212**	1.9	**197**	51.72	**104**	23690	94	**54**	533	2.0

Trends we draw attention to are: Local Search consistently beat No Search, but at the cost of increased computational time. Informed Global Search significantly beats No Search;and it also searches *much* less of state space than Uninformed Global Search, resulting in correspondingly faster running times. Also, because of the sparse representation of Global Search grids, we can comfortably use a grid resolutions as high as 50^4.

[3]The parameters for the 4 domains were, in order: value function interpolation grid resolution: $7^2, 13^4, 13^4, 13^4$, Local Search: $d = 6, s = 2, d = 5, s = 4, d = 5, s = 4, d = 10, s = 1$, Global Search Grid resolution: $50^2, 50^4, 50^4, 20^4$, Local search within Global search: $d = 20, s = 1$ for all 4.

While small, MODIFIED-CAR demonstrates interesting phenomena. Despite the use of a 50^2 grid for the global search, Uninformed Global Search often surprisingly fails to find a path to the goal, where Informed Global Search, despite searching much less of the state space, succeeds. This is because Informed Global Search uses a value function to guide its pruning of multiple trajectories entering the same grid cell, and therefore makes better selection of "representative states" for grid elements. This also helps explain Informed Global Search beating Uninformed Global Search on 3 of 4 domains. When the Global Search grid resolution is increased to 100^2 for MODIFIED-CAR, both global searches consistently succeed. But, Uninformed Global Search (mean cost 109) now beats Informed Global Search (mean cost 138). The finer grid causes good selection of "representative states" to be less important; meanwhile, inaccuracies in the value function guiding Informed Global Search causes it to "miss" certain good trajectories. This is a phenomenon that often occurs in A^*-like searches when one's heuristic evaluation function is not strictly "optimistic" [8]. This is not a problem for Uninformed Global Search, which is effectively using the maximally optimistic "constant 0" evaluation function.

5 Learning a Model Online

Occasionally, the state transition function is not known but rather must be learned online. This does not preclude the use of online search techniques; as a toy example, Figure 3 shows cumulative cost learning curves for MODIFIED-CAR. For each action, a kd-tree implementation of 1 nearest neighbor is used to learn the state transitions, and to encourage exploration, states sufficiently far from points stored in both trees are optimistically assumed to be zero-cost absorbing states. A 7^2 value function approximator is updated online with the changing state transition model. Without search, the learner eventually attains an average cost per trial of about 212; with Informed Global Search (search grid resolution 50^2), it quickly (after about 5 trials) achieves an average cost of 155; with Local Search ($d = 20, s = 1$), it achieves an average cost of 127 (also after about 5 trials).

However, several interesting issues do arise when the state transition function is being approximated online. Inaccuracies in the model may cause the Global Searches to fail in cases where more accurate models would have let them find paths to the goal. Furthermore, trajectories supposedly found during search will certainly not be followed exactly by an open-loop controller; adaptive closed-loop controllers may help alleviate this problem to some extent. Finally, using the models to predict state transitions should be computationally cheap, since we will be using them to update the approximated value function with the changing model, as well as to perform searches.

6 Future Research

How well will these techniques extend to non-deterministic systems? They may work for problems in which certain regularity assumptions are reasonable, but more sophisticated state transition function approximators may be required when learning a model online.

How useful is Local Search in comparison with building a local linear controller for trajectories? During execution some combination of the two may be best. Local Search also plays an important role in the inner loop of global search: it is unclear how local linear control could do the same.

The experiments presented here are low dimensional. It is encouraging that informed search permits us to survive 50^4 grids, but to properly thwart the curse of dimensionality we can conclude that (i) stronger approximation than simplex grids are needed, (ii) Informed Global Search is much more tractable than Uninformed Global Search, and (iii) variable resolution methods (e.g. extensions to [6]) might be needed.

Lastly, algorithms to learn reasonably accurate yet consistently "optimistic" [8] value functions might be helpful for Informed Global Search.

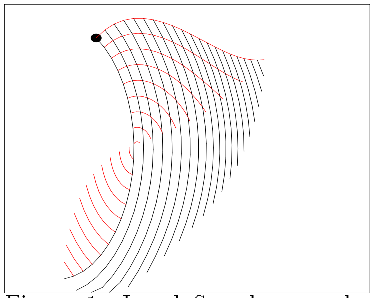

Figure 1: Local Search example: a twenty-step search with at most one switch in actions

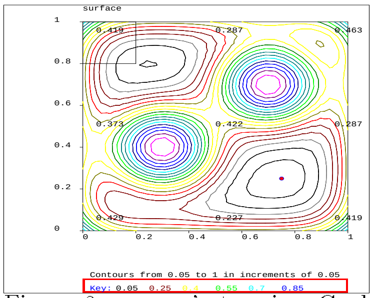

Figure 2: SLIDER's terrain. Goal at upper left.

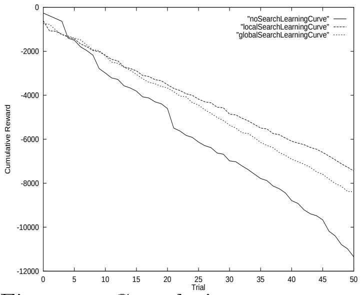

Figure 3: Cumulative cost curves on MODIFIED-CAR with model learning.

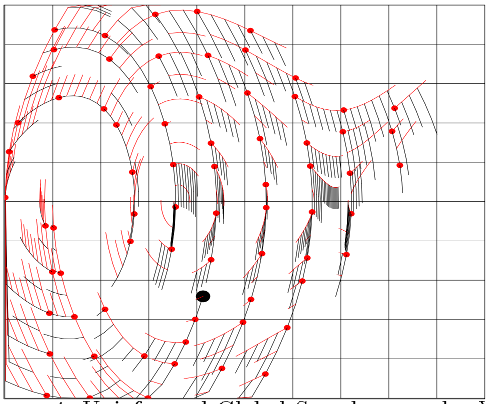

Figure 4: Uninformed Global Search example. Velocity on x-axis, car position on y axis. Large black dot is starting state; the small dots are grid elements' "representative states."

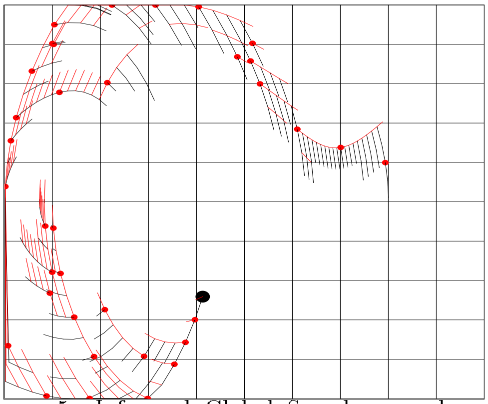

Figure 5: Informed Global Search example. on MODIFIED-CAR.

References

[1] Dimitri P. Bertsekas. *Dynamic Programming and optimal control*, volume 1. Athena Scientific, 1995.

[2] G. Boone. Minimum-Time Control of the Acrobot. In *International Conference on Robotics and Automation*, 1997.

[3] S. Davies. Multidimensional Triangulation and Interpolation for Reinforcement Learning. In *Neural Information Processing Systems 9, 1996*. Morgan Kaufmann, 1997.

[4] R. E. Korf. Real-Time Heuristic Search. *Artifical Intelligence*, 42, 1990.

[5] J. Latombe. *Robot Motion Planning*. Kluwer, 1991.

[6] A. W. Moore and C. G. Atkeson. The Parti-game Algorithm for Variable Resolution Reinforcement Learning in Multidimensional State-spaces. *Machine Learning*, 21, 1995.

[7] N. J. Nilsson. *Problem-solving Methods in Artificial Intelligence*. McGraw Hill, 1971.

[8] S. Russell and P. Norvig. *Artificial Intelligence A Modern Approach*. Prentice Hall, 1995.

[9] R. S. Sutton. Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding. In D. Touretzky, M. Mozer, and M. Hasselmo, editors, *Neural Information Processing Systems 8*, 1996.