Applying Online Search Techniques to Continuous-State Reinforcement Learning

Scott Davies* **Andrew Y. Ng†** **Andrew Moore***

* School of Computer Science
Carnegie-Mellon University
Pittsburgh, PA 15213

† Artificial Intelligence Lab
Massachusetts Institute of Technology
Cambridge, MA 02139

Abstract

In this paper, we describe methods for efficiently computing better solutions to control problems in continuous state spaces. We provide algorithms that exploit online search to boost the power of very approximate value functions discovered by traditional reinforcement learning techniques. 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 the local 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.

The key to the success of the global methods lies in using aggressive state-space search techniques such as uniform-cost search and A^*, tamed into a tractable form by exploiting neighborhood relations and trajectory constraints that arise from continuous-space dynamic control.

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.

However, particularly in high-dimensional continuous state spaces, it can often be computationally expensive to fit a highly accurate value function, even when our agent is given a perfect model of the world. 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, we investigate the idea that rather than executing greedy policies with respect to approximated value functions in continuous-state domains, we can use online search techniques to find better trajectories. We restrict our attention to deterministic domains. The paper consists of a progression of improvements to conventional action-selection from value functions, along the way using techniques from value function approximation (Davies, 1997), real-time search (Korf, 1990), constrained trajectories (Burghes and Graham 1980), and robot motion planning (Latombe 1991, Boyan *et al.* 1995, Boone 1997). All of the algorithms perform search online to find a good trajectory from some current state. Briefly, the progression is as follows:

- **LS: Local Search.** Takes a forward-dynamics model and an approximate value function, and performs a limited-depth lookahead search of possible trajectories from the current state before suggesting an action.
- **CLS: Constrained Local Search.** Does a similar job as LS, but considers only trajectories in which the action is changed infrequently. This results in substantial computational savings that allow it to search much deeper or faster.
- **UGS: Uninformed Global Search.** For least-cost-to-goal problems, takes a forward-dynamics model and plans an approximately-least-cost path from the current state all the way to the goal, using LS or CLS along with a neighborhood-based pruning technique to permit tractable searches even when they cover large areas of the continuous state space.
- **IGS: Informed Global Search.** Does a similar job as UGS, but uses an approximate value function to guide the search in a manner very similar to A^* (Nilsson 1971), thereby vastly reducing the search time and (somewhat surprisingly) often dramatically improving the solution quality as well.
- **LLS: Learning Local Search.** *Learns* a forward-dynamics model, and uses it to generate approximate value functions for the LS and CLS approaches.
- **LGS: Learning Global Search.** *Learns* a forward-dynamics model, and uses it to generate approximate value functions for the LS and CLS approaches.

In this paper, the approximate value functions are obtained by k-dimensional simplex interpolation combined with value iteration (Davies 1997), but the approaches are applicable for accelerating any modelbased reinforcement learning algorithm that produces approximate value functions, such as an LQR solution to a linearized problem or a neural net value function computed with TD.

With such 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. However, the global search algorithms go beyond shallow lookahead methods and instead use a pruned "bush" of search trajectories to find continuous trajectories all the way to the goal.

We apply these search techniques to several continuous-state problems, and demonstrate that they often dramatically improve the quality of solutions at relatively little computational expense.

MOUNTAIN-PARKING: An example of a continuous-space dynamic control task

Figure 1 depicts a car (idealized as a frictionless puck) on a very steep hill. The car can accelerate forward or backward with a limited maximum thrust. The goal is to park the car in a space near the top of the hill (that is, occupy the space while having a near-zero velocity). Because of gravity, there is a region near the center of the hill at which the maximum forward thrust is not strong enough to accelerate up the slope. This is depicted on the two-dimensional diagram in Figure 2. Thus if the goal is at the top of the slope, a strategy that proceeded by greedily choosing actions to thrust towards the goal would get stuck. Figure 3 shows a sample minimum-time path for one possible initial state. This task, although trivial to solve by dynamic programming on a very fine grid, will be used as an illustration during the exposition because its state space can be drawn as a two-dimensional diagram. In the Experiments section we will see empirical results for problems that would be intractably expensive for dynamic programming on a fine grid.

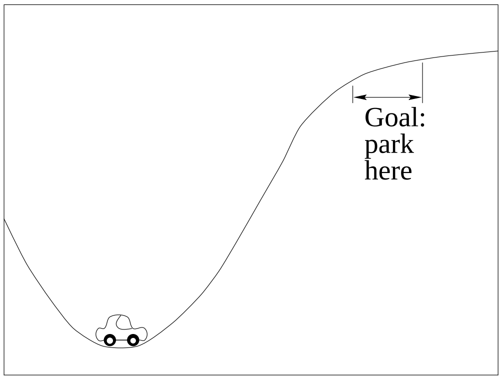

Figure 1: A car acted on by gravity and limited forward/backward thrust. The car must park in the goal area as quickly as possible.

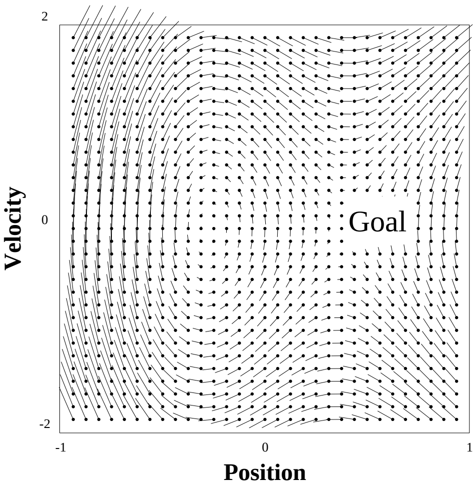

Figure 2: The state transition function for a car constantly thrusting to the right with maximum thrust. A point on the diagram represents a state of the car. Horizontal position denotes the physical car position. Vertical diagram position denote the car's velocity.

LS: 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 $O(d|A|^d)$.

During execution, the LS algorithm iteratively finds the best trajectory T of length d with the search algo-

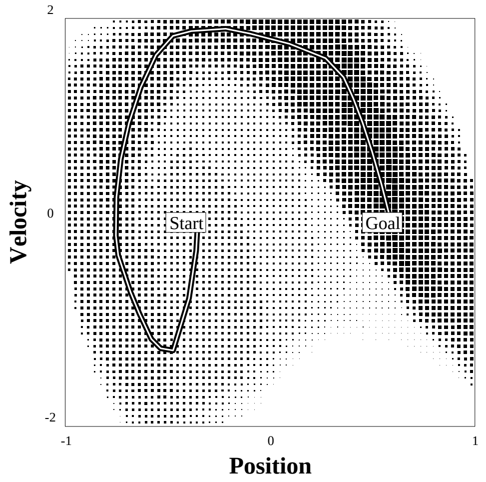

Figure 3: A minimum-time path for the car on the hill. The optimal value function is shown by dots. The shorter the time to goal, the larger the black dot.

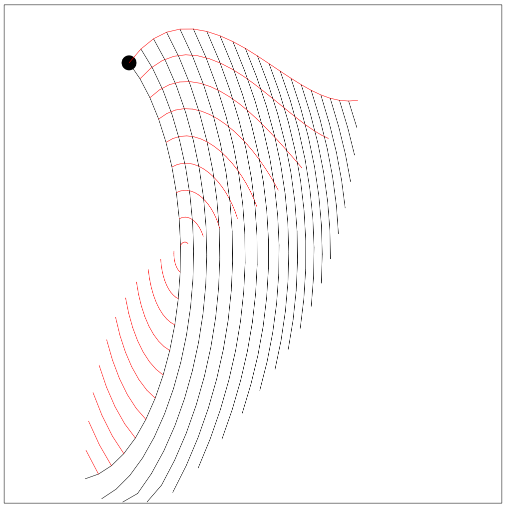

Figure 4: Constrained Local Search (CLS) example: a twenty-step search with at most one switch in actions

rithm 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" (Bertsekas 1995) so that $BV(s) = max_{a \in A} R(s,a) + \gamma V(\delta(s,a))$, 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 $k \rightarrow \infty$, $B^k V$ 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 $\gamma^{d-1}\varepsilon$.

This approach, a form of receding horizon control, has most famously been applied to minimax game playing programs (Russell and Norvig 1995) and has also been used in single-agent systems on discrete domains (e.g. (Korf 1990)). In game-playing scenarios it has also been used in conjunction with automatically learned value functions, such as in Samuel's celebrated checkers program (Samuel 1959) and Tesauro's backgammon player (Tesauro and Galperin, 1997).

CLS: Constrained Local Search

To make deeper searches computationally cheaper, we might consider only a subset of all possible trajectories of depth d. Especially for dynamic control, often an optimal trajectory repeatedly selects and then *holds* a certain action for some time, such as suggested by (Boone 1997). Therefore, a natural subset of the $|A|^d$ possible trajectories 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 4 shows CLS performed in the MOUNTAIN-PARKING task using $d = 20$ and $s = 1$.

Since LS is the same as CLS with the maximum-number-of-switches parameter s set to $d-1$, we may use "LS" or "local search" to refer generically to both CLS and LS at certain points throughout the rest of the paper.

UGS: Uninformed Global Search

Local searches (LS and CLS) are not the only way to more effectively use an approximated 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.

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 (Latombe 1991). We first divide the state space up into a fine uniform grid. A sparse representation is used so that only grid cells that are visited take up memory[1].

A local search procedure (LS or CLS) 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 point at which this least-cost trajectory first enters a grid element is used as the grid element's "representative state," and

[1]This has a flavor not dissimilar to the hashed sparse coarse encodings of (Sutton 1996).

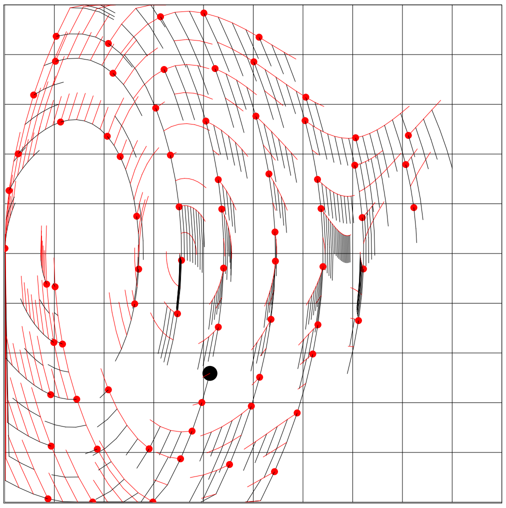

Figure 5: Uninformed Global Search (UGS) example. Velocity on x-axis, car position on y axis. Large black dot is starting state; the small dots are grid elements' "representative states."

acts as the starting point for the local search. The rationale for the pruning is an assumed similarity 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. When the planner finds a trajectory to the goal, it is executed in its entirety.

The overall procedure is essentially a lowest-cost-first search over a graph structure in which the graph nodes correspond to grid elements, and in which the edges between graph nodes correspond to trajectories between grid elements as found by the CLS procedure. A graph showing such a search for the MOUNTAIN-PARKING domain is depicted in Figure 5.

IGS: Informed Global Search

We can modify Uninformed Global Search (UGS) by using an approximated value function to *guide* the search expansions in the style of A^* search (Nilsson 1971), as written out in detail below. The search proceeds from the most promising-looking states first, where the "promise" of a state is the cost to get to the state (along previously searched trajectories) plus the remaining-cost-to-go as estimated with the value function. 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.

As in UGS, the grid is represented sparsely. Notice also that like LS and CLS, 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.

Written 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 LS or CLS as described in the Local Search section, except search trajectories are pruned once they reach a state in a different grid element 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 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.

The above procedure is very similar to a standard A^* search, with two important differences. First, the heuristic function used here is an automatically generated approximate value function rather than a hand-coded heuristic. This has the advantage of being a more autonomous approach requiring relatively little hand-encoding of domain-specific knowledge. On the other hand, it also typically means that the heuristic function used here may sometimes overestimate the cost required to get from some points to the goal, which can sometimes lead to suboptimal solutions – that is, the approximated value function is not necessarily an *optimistic* or *admissible* heuristic (Russell and Norvig 1995). *However, within the context of the search procedure used above, inadmissible but relatively accurate value functions can lead to much better solutions than those found with optimistic but inaccurate heuristics.* (Note that UGS is essentially IGS with an optimistic but inaccurate heuristic "value function" of 0 everywhere.) This is due to the second important difference between our search procedure above and a standard A^* search in a discrete-state domain: IGS uses the approximated value function not only to decide what grid element to search from next, but also from what particular point in that grid element it will search for local trajectories to neighboring grid elements.

The above algorithm is also similar in spirit to algorithms presented in (Atkeson, 1994). Atkeson's algorithms also found continuous trajectories from start states to goals. The search for such trajectories was

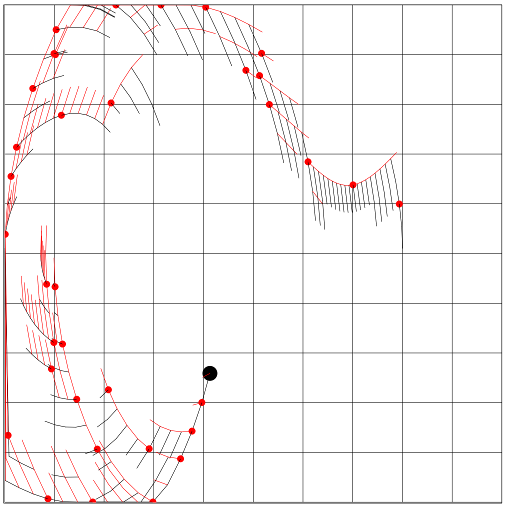

Figure 6: Informed Global Search (IGS) example on MOUNTAIN-PARKING, with a crudely approximated value function.

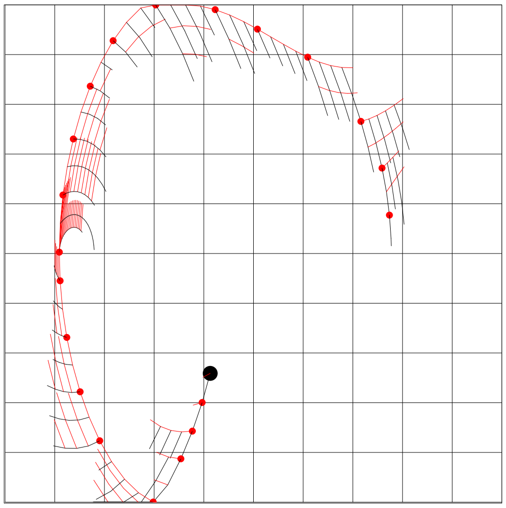

Figure 7: Informed Global Search (IGS) example on MOUNTAIN-PARKING, with a more accurately approximated value function.

performed either within the context of a regular grid (as in the algorithm above) or a pair of constant-cost contours gradually grown out from the start and goal states. Our algorithms differ from Atkeson's in that our algorithm works with a small set of discrete actions and can handle some discontinuities in the dynamics, whereas Atkeson's algorithm requires smooth dynamics (continuous first and second derivatives) with continuous actions. Unlike Atkeson's work, our algorithm does not yet locally optimize the trajectories found by our search algorithms. However, also unlike Atkeson's work, we first compute a crude but quick approximation to the value function (except in the case of uninformed global search), and using this approximate value function speeds up the search considerably.

An example of IGS on the MOUNTAIN-PARKING domain is shown in Figure 6. The value function was approximated with a simplex-based interpolation (Davies 1997) on a coarse 7 by 7 grid, with all other parameters the same as in Figure 5. Much less of state space is searched than by UGS.

In Figure 7, the value function was approximated more accurately with a simplex-based interpolation on a 21 by 21 grid. With this accurate a value function, the search goes straight down a near-optimal path to the goal. Naturally, in such a situation the search is actually unnecessary, since merely greedily following the approximated value function would have produced the same solution. However, when we move to higher-dimensional problems, such as problems examined in the next section, high-resolution approximated value functions become prohibitively expensive to calculate, and IGS can be a very cost-effective way of improving performance.

Experiments

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

- MOUNTAIN-PARKING (2 dimensional): As described in the Introduction. This is slightly more difficult than the normal mountain-car problem, as we require a velocity near 0 at the top of the hill (Moore and Atkeson 1995). 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 (Sutton 1997). 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 (Barto *et al.* 1983) 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 8. 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

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

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

Table 1: Local search (LS) on MOVE-CART-POLE

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

Table 2: Constrained Local search (CLS) on MOUNTAIN-PARKING

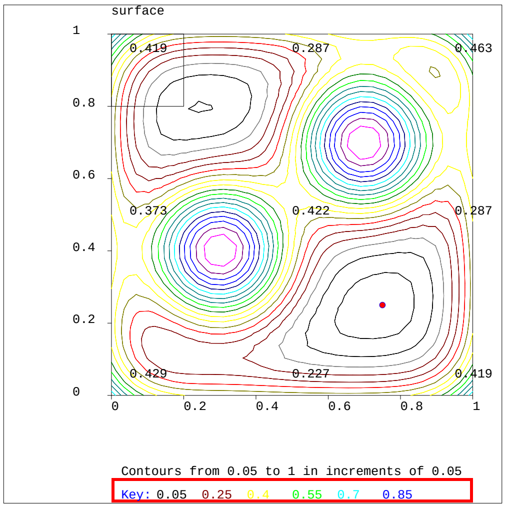

Figure 8: SLIDER's terrain. Goal at upper left.

other three domains cost a constant -1 per step. All results are averages of 1000 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.

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. In this case, the value functions used during search (except by the Uniformed Global Search) are calculated using the simplex-interpolation algorithm described in (Davies 1997); once generated, they need not be updated during the search process.

Local Search

Here, we look at the effects of different parameter settings for Local Search. We first consider MOVE-CART-POLE. Empirically, good trajectories in this domain "switch" actions very often; therefore, we 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. (Calculating the approximate value function even with this seemingly low resolution can take minutes of CPU time and most of the system's memory.) See Table 1; 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.

The next experiment we consider here is MOUNTAIN-PARKING 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 $s = 2$. In Table 2, our experimental results again show solution quality significantly increased by Local Search, but with running times growing much more slowly with d than before.

Comparative Experiments

Table 3 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).

Trends we draw attention to are: Local Search consistently beat No Search, but at the cost of increased computational time. Informed Global Search (IGS) significantly beats No Search; and it also searches *much* less of state space than Uninformed Global Search (UGS), resulting in correspondingly faster running times. In fact, because the solutions found by IGS are often of much shorter length than when using no search at all, the computational time per trial is sometimes essentially *the same* for IGS and No Search, while the quality of the solution found by IGS is many times better — for example, a factor of 4 in the SLIDER domain. (It performs a factor of 10 better in the MOVE-CART-POLE

[3] The parameters for the 4 domains were, in order: value function simplex 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.

	No Search		Local Search		Uninformed Global			Informed Global		
	cost	time	**cost**	time	**cost**	#LS	time	**cost**	#LS	time
MOUNTAIN-PARKING	**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

Table 3: Summary of comparative experimental results

domain, but that is largely a function of the particular penalty associated with the pole falling over.) Also note that because of the sparse representation of Global Search grids, we can comfortably use grid resolutions as high as 50^4 without running out of memory.

While relatively simple, MOUNTAIN-PARKING demonstrates interesting phenomena. Despite the use of a 50^2 grid for the global search, UGS often surprisingly fails to find a path to the goal, where IGS, despite searching much less of the state space, succeeds. This is because IGS 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 IGS finding better solutions than UGS on 2 of the 3 four-dimensional domains.

When the Global Search grid resolution is increased to 100^2 for MOUNTAIN-PARKING, both UGS and IGS consistently succeed. But, UGS (mean cost 109) now finds better solutions than IGS (mean cost 138). The finer search 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 (Russell and Norvig 1995). This is not a problem for UGS, which is effectively using the maximally optimistic "constant 0" evaluation function. It is interesting to note that in the MOVE-CART-POLE domain, in which UGS found better solutions than IGS, the step size was large enough and the dynamics nonlinear enough that single steps often crossed multiple grid elements, and each grid element was typically reached no more than once during the search. Thus, this was a case in which IGS's ability to discriminate between good and bad states within the same grid element was not relevant.

LLS and LGS: 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 9 shows cumulative reward learning curves for MOUNTAIN-PARKING. For each action, a *k*d-tree implementation of 1-nearest-neighbor (Friedman *et al.* 1977) 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

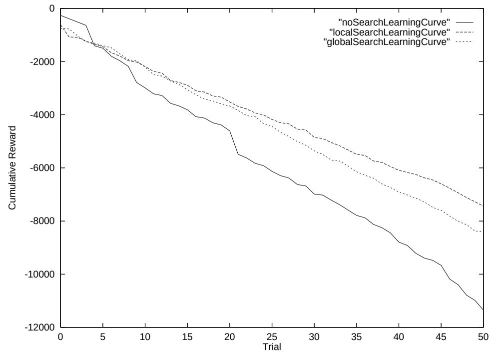

Figure 9: Cumulative reward curves on MOUNTAIN-PARKING with model learning. (Shallow gradients are good.)

zero-cost absorbing states. A 7-by-7 simplex interpolation grid used for the 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 Learning Global Search (LGS) (search grid resolution 50^2), it quickly (after about 5 trials) achieves an average cost of 155; with Learning Local Search (LLS) ($d = 20, s = 1$), it achieves an average cost of 127 (also after about 5 trials).

As before, when the planner finds a trajectory to the goal, it is executed in its entirety in an open-loop fashion. But in the case where we are learning a model of the world, it is possible to successfully plan a continuous trajectory using the learned world model, but for the agent 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.

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. Optimistic exploration policies can be used to help the system gather enough data to reduce these inaccuracies, but in even moderately high-dimensional spaces such exploration would become very expensive. 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.

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

1. Informed Global Search (IGS) is often much more tractable than Uninformed Global Search (UGS), even with relatively crudely approximated value functions.
2. However, more accurate (yet computationally tractable) value function approximators may be needed than the simplex-grid-based approximators used here.
3. Variable resolution methods (e.g. extensions to (Moore and Atkeson 1995)) would probably be needed for the Global Search's state-space partitions rather than the uniform grids used here.

The algorithms tested in this paper calculated the approximate value functions used by their search procedures independently of any particular trajectories that were subsequently searched or executed. However, it might be better to use points along such trajectories to further update the value function in order to concentrate computational time and value function approximator accuracy on the most relevant parts of the state space. The resulting algorithm would be reminiscent of Korf's RTA^* (Korf 1990) and Barto's $RTDP$ (Barto *et al.* 1995).

The trajectories found by the algorithms described in this paper use a small discrete set of actions, and do not always switch between these actions in a completely locally optimal manner. In domains where the action space is actually continuous, it would be useful to use a local trajectory optimization routine such as that used in (Atkeson, 1994) in order to fine-tune the discovered trajectories.

Lastly, algorithms to learn reasonably accurate yet consistently "optimistic" (Russell and Norvig 1995) value functions might be helpful for Informed Global Search.

References

Atkeson, C. G. 1989. Using Local Models to Control Movement. In *Proceedings of Neural Information Processing Systems Conference*.

Barto, A. G.; Bradtke, S. J.; and Singh, S. P. 1994. Real-time Learning and Control using Asynchronous Dynamic Programming. *AI Journal, to appear (also published as UMass Amherst Technical Report 91-57 in 1991)*.

Barto, A. G.; Sutton, R. S.; and Anderson, C. W. 1983. Neuronlike Adaptive elements that that can learn difficult Control Problems. *IEEE Trans. on Systems Man and Cybernetics* 13(5):835–846.

Bertsekas, D. P. 1995. *Dynamic Programming and optimal control*, volume 1. Athena Scientific.

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

Boyan, J. A.; Moore, A. W.; and Sutton, R. S., eds. 1995. *Proceedings of the Workshop on Value Function Approximation*. Machine Learning Conference: CMU-CS-95-206. Web: http://www.cs.cmu.edu/~reinf/ml95/.

Burghes, D., and Graham, A. 1980. *Introduction to Control Theory including Optimal Control*. Ellis Horwood.

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

Friedman, J. H.; Bentley, J. L.; and Finkel, R. A. 1977. An Algorithm for Finding Best Matches in Logarithmic Expected Time. *ACM Trans. on Mathematical Software* 3(3):209–226.

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

Latombe, J. 1991. *Robot Motion Planning*. Kluwer.

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

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

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

Samuel, A. L. 1959. Some Studies in Machine Learning using the Game of Checkers. *IBM Journal on Research and Development* 3. Reprinted in E. A. Feigenbaum and J. Feldman, editors, *Computers and Thought*, McGraw-Hill, 1963.

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

Tesauro, G., and Galperin, G. R. 1997. On-line Policy Improvement using Monte-Carlo Search. In Mozer, M. C.; Jordan, M. I.; and Petsche, T., eds., *Advances in Neural Information Processing Systems 9*. Morgan Kaufmann.