Using Domain-Configurable Search Control for Probabilistic Planning

Ugur Kuter and **Dana Nau**
Department of Computer Science
and Institute for Systems Research
University of Maryland
College Park, MD 20742, USA

Abstract

We describe how to improve the performance of MDP planning algorithms by modifying them to use the search-control mechanisms of planners such as TLPlan, SHOP2, and TALplanner. In our experiments, modified versions of RTDP, LRTDP, and Value Iteration were exponentially faster than the original algorithms. On the largest problems the original algorithms could solve, the modified ones were about 10,000 times faster. On another set of problems whose state spaces were more than 14,000 times larger than the original algorithms could solve, the modified algorithms took only about 1/3 second.

Introduction

Planning algorithms for MDPs typically have large efficiency problems due to the need to explore all or most of the state space. For complex planning problems, the state space can be quite huge. For problems expressed using probabilistic STRIPS operators (Hanks & McDermott 1993; Kushmerick, Hanks, & Weld 1994) or 2TBNs (Hoey *et al.* 1999; Boutilier & Goldszmidt 1996), planning is EXPTIME-hard (Littman 1997). This paper focuses on a way to improve the efficiency of planning on MDPs by adapting the techniques used in domain-configurable classical planners.

A domain-configurable planner consists of a domain-independent search engine that can make use of domain-specific (but problem-independent) search-control knowledge that is given to the planner as part of its input. Examples include planners such as TLPlan (Bacchus & Kabanza 2000) and TALplanner (Kvarnström & Doherty 2001) in which the search-control knowledge consists of pruning rules written in temporal logic, and *Hierarchical Task Network (HTN)* planners such as SIPE-2 (Wilkins 1990), O-Plan (Currie & Tate 1991), and SHOP2 (Nau *et al.* 2003), in which the search-control knowledge consists of HTN "methods" (task-decomposition templates).

Domain-configurable planners have been highly successful. In the AI planning competitions, such planners consistently worked in most domains, solved the most problems, and solved them fastest (Bacchus 2001; Fox & Long 2002). They are used in a large variety of real-world applications (Wilkins 1990; Currie & Tate 1991; Nau *et al.* 2003).

Our contributions are as follows:

- We describe how to modify any forward-chaining MDP planning algorithm, by incorporating into it the search-control algorithm from any forward-chaining domain-configurable planner.
- We describe conditions under which our modified MDP planning algorithms are guaranteed to find optimal answers, and conditions under which they can do so exponentially faster than the original ones.
- We have applied our approach to Real-time Dynamic Programming (RTDP) (Bonet & Geffner 2000), Labeled RTDP (LRTDP) (Bonet & Geffner 2003), and a forward-chaining version of Value Iteration (Berstekas 1995). Our experimental results show our modified algorithms running exponentially faster than the original ones. On the largest problems the original algorithms could solve, the modified ones ran about 10,000 times faster. In only about 1/3 second, the modified algorithms could solve problems whose state spaces were more than 14,000 times larger.

Background

Domain-Configurable Classical Planners. We use the usual definition of a classical planning problem. In Fig. 1, Controlled-Plan is an abstract version of a forward-chaining domain-configurable planner. s is the current state, G is the goal, π is the current plan, D is the domain description, and x is the auxiliary information used by the *search-control* function $\mathsf{acceptable}(s, a, x, D)$. $\mathit{result}(s, a)$ is the state produced by applying the action a to s, and $\mathsf{progress}(s, a, x, D)$ is the auxiliary information to use in the next state. Here are two examples:

- TLPlan (Bacchus & Kabanza 2000) maintains a *control formula* written in a modal temporal logic, and backtracks whenever the current control formula evaluates to FALSE. TLPlan is an instance of Controlled-Plan in which x is the current control formula, D is the set of all actions in the domain, $\mathsf{progress}(s, a, x, D)$ is the next control formula generated by TLPlan's temporal-progression algorithm,

```
Procedure Controlled-Plan(s, G, π, x, D)
  if s ∈ G then return(π)
  actions ← {a | a is applicable to s
                 and acceptable(s, a, x, D) holds}
  if actions = ∅ then return(failure)
  nondeterministically choose a ∈ actions
  s' ← result(s, a)
  π' ← append(π, a)
  x' ← progress(s, a, x, D)
  return(Controlled-Plan(s', G, π', x', D))
```

Figure 1: An abstract version of a forward-chaining domain-configurable classical planner.

and acceptable(s, a, x, D) holds for all actions a applicable in s such that the state $result(s, a)$ satisfies progress(s, a, x, D).

- SHOP2 (Nau *et al.* 2003), an HTN planner, is an instance of Controlled-Plan in which x is the current task network and D contains all actions and methods in the domain. acceptable(s, a, x, D) holds for all actions a such that (i) a appears in some task network x' that is produced by recursively decomposing tasks in x, and (ii) a has no predecessors in x'. progress(s, a, x, D) is the task network produced by removing a from x'.

MDP-based Planning. We consider MDPs of the form $M = (S, A, app, Pr, C, R)$, where:

- S and A, the sets of states and actions, are finite.
- $app(s)$ is the set of all actions applicable in s.
- For every $a \in app(s)$, $Pr(s, a, s')$ is the probability of the state-transition (s, a, s').
- For every $s \in S$ and every a applicable to s, $C(s, a) \geq 0$ is the *cost* of applying a to s.
- For every $s \in S$, $R(s) \geq 0$ is the *reward* for s.

An *MDP planning problem* is a triple $\mathcal{P} = (M, S_0, G)$, where $M = (S, A, app, Pr, C, R)$ is an MDP, $S_0 \subseteq S$ is the set of initial states, and $G \subseteq S$ is the set of goal states.

For a state s in an MDP problem $\mathcal{P} = (M, S_0, G)$, we define $results(s, a) = \{s' \mid Pr(s, a, s') > 0\}$ if $a \in app(s)$; $results(s, a) = \emptyset$ otherwise. We define $succ(s) = \bigcup\{s' \in results(s, a) \mid a \in app(s)\}$. We say that s is a *terminal* state if s is a goal state (i.e., $s \in G$) or $app(s) = \emptyset$.

The *value* of a state s and a state-action pair (s, a) are defined recursively as follows:

$$V(s) = \begin{cases} R(s), & \text{if } s \text{ is terminal} \\ \max_{a \in app(s)} Q(s, a), & \text{otherwise} \end{cases} \quad (1)$$

$$Q(s, a) = R(s) - C(s, a) + \gamma \sum_{s' \in results(s,a)} Pr(s, a, s')\ V(s') \quad (2)$$

where $0.0 \leq \gamma \leq 1.0$ is the discount factor. The *residual* of a state s is defined as the difference between the left and right sides of the Eqn. 1.

A *policy* is a partial function from S into A; i.e., $\pi : S_\pi \rightarrow A$ for some set $S_\pi \subseteq S$. The *size* of π is $|S_\pi|$. $V_\pi(s)$, the value of s induced by the policy π, is defined similarly to $V(s)$ except that the actions applied to each state s are only the ones in $\pi(s)$.

An *optimal solution* for an MDP planning problem $\mathcal{P} = (M, S_0, G)$ is a policy π such that when π is executed in any of the initial states S_0, it reaches a goal state in G with probability 1, and for every state $s \in S_0$, there is no other policy π' such that $V_{\pi'}(s) > V_\pi(s)$.

Note that the value function V defined in Eqn. 1 need not to be a total function. Given an MDP planning problem $\mathcal{P} = (M, S_0, G)$ and an optimal solution π for it, the set of states that are reachable from the initial states S_0 by using π constitute a minimal set of states over which the value function V needs to be defined.

Researchers have often defined MDP versions of classical planning problems. An MDP planning problem $\mathcal{P}^F$ is an *MDP version* of a classical problem $\mathcal{P}$ iff the two problems have the same states, initial state, and goal, and if there is a one-to-one mapping *det* from $\mathcal{P}^F$'s actions to $\mathcal{P}$'s actions such that for every action a in $\mathcal{P}^F$, a and $det(a)$ are applicable to exactly the same states, and for each such state s, $result(s, det(a)) \in results(s, a)$. The additional states in $results(s, a)$ can be used to model various sources of the uncertainty in the domain, such as action failures (e.g., a robot gripper may drop its load) and exogenous events (e.g., a road is closed). We call $det(a)$ the *deterministic version* of a, and a the *MDP version* of $det(a)$.

Search Control for MDPs

We now describe how to incorporate the search-control function acceptable of Fig. 1 into MDP planning algorithms. Many MDP planning algorithms can be viewed as forward-search procedures embedded inside iteration loops.[1] The forward-search procedure starts at S_0 and searches forward by applying actions to states, computing a policy and/or a set of utility values as the search progresses. The iteration loop continues until some sort of convergence criterion is satisfied (e.g., until two successive iterations produce identical utility values for every node, or until the residual of every node becomes less than or equal to a termination criterion $\epsilon > 0$).

Planners like RTDP and LRTDP fit directly into this format. These planners repeatedly (1) do a greedy search going forward in the state space, then (2) update the values of the visited states in a dynamic-programming fashion. Even the well known Value Iteration (VI) algorithm can be made to fit into the above

[1] For example, in Fig. 2, a forward-chaining version of Value Iteration, the iteration loop is the outer while loop, and the forward-search procedure is everything inside that loop.

```
Procedure Fwd-VI
 select any initialization for V; π ← ∅
 while V has not converged do
  S ← S_0; Visited ← ∅
  while S ≠ ∅ do
    for every state s ∈ S ∩ G, V(s) ← R(s)
    S ← S \ G; S' ← ∅
    for every state s ∈ S
      for every a ∈ app(s)
        Q(s,a) ← (R(s) − C(s,a))
                 +γ Σ_{s'∈results(s,a)} Pr(s,a,s') V(s')
        S' ← S' ∪ {s | s ∈ results(s,a)}
      V(s) ← max_{a∈app(s)} Q(s,a)
      π(s) ← argmax_{a∈app(s)} Q(s,a)
    Visited ← Visited ∪ S
    S ← S' \ Visited
 return π
```

Figure 2: Fwd-VI, a forward-chaining version of Value Iteration.

format, by making sure to compute the values in a forward-chaining manner (see the Fwd-VI algorithm in Fig. 2).

During the forward search, at each state s that the planner visits, it needs to know $app(s)$, the set of all actions applicable to s. For example, the inner for loop of Fwd-VI iterates over the actions in $app(s)$; and RTDP and LRTDP choose whichever action in $app(s)$ currently has the best value.

Let Z be a forward-chaining MDP planning algorithm, F be an instance of Controlled-Plan (see Fig. 1), and $\mathsf{acceptable}_F$ be F's search-control function. We will now define Z^F, a modified version of Z in which every occurrence of $app(s)$ is replaced by

$$\{a \in app(s) \mid \mathsf{acceptable}_F(s, det(a), x, D) \text{ holds}\}.$$

The reason we require Z to be a forward-chaining MDP algorithm is because the auxiliary information x is computed by progression from s's parent.

Here are some examples of Z^F:

- Fwd-VI$^{\mathsf{TLPlan}}$ is the forward-chaining version of VI combined with TLPlan's search control function,
- RTDP$^{\mathsf{TALplanner}}$ is RTDP combined with TALplanner's search control function,
- LRTDP$^{\mathsf{SHOP2}}$ is LRTDP combined with SHOP2's search control function.

Formal Properties

Let Z be a forward-chaining MDP planning algorithm that is guaranteed to return an optimal solution if one exists, F be an instance of Controlled-Plan, and $\mathsf{acceptable}_F$ be F's search control function. Suppose $M = (S, A, app, Pr, C, R)$ is an MDP and $\mathcal{P} = (M, S_0, G)$ be a planning problem over M. Then we can define the *reduced* MDP M^F and planning problem $\mathcal{P}^F$ as follows:

$$app^F(s) = \{a \in app(s) \mid \mathsf{acceptable}_F(s, det(a), x, D) \text{ holds}\};$$

$$results^F(s,a) = \begin{cases} results(s,a) & \text{if } a \in app^F(s), \\ \emptyset & \text{otherwise;} \end{cases}$$

$$succ^F(s) = \bigcup \{s' \in results^F(s,a) \mid a \in app^F(s)\};$$

$$S^F = \text{transitive closure of } succ^F \text{ over } S_0;$$

$$G^F = G \cap S^F;$$

$$M^F = (S^F, A, app^F, Pr, C, R);$$

$$\mathcal{P}^F = (M^F, S_0, G^F).$$

Recall that in every place where the algorithm Z uses $app(s)$, the algorithm Z^F instead uses $\{a \in app(s) \mid \mathsf{acceptable}_F(s, det(a), x, D) \text{ holds}\}$. Thus from the above definitions, it follows that running Z^F on $\mathcal{P}$ is equivalent to running Z on $\mathcal{P}^F$.

We say that $\mathsf{acceptable}_F$ is *admissible* for $\mathcal{P}$ if for every state s in $\mathcal{P}$, there is an action $a \in app(s)$ such that $\mathsf{acceptable}_F(s, det(a), x, D)$ holds and $V(s) = Q(s,a)$, where $V(s)$ and $Q(s,a)$ are as in Eqs. 1−2. From this we get the following:

Theorem 1 *Suppose Z returns a policy π for $\mathcal{P}$. Then, Z^F returns a policy π' for $\mathcal{P}$ such that $V_\pi(s) = V_{\pi'}(s)$ for every $s \in S_0$, if $\mathsf{acceptable}_F$ is admissible for $\mathcal{P}$.*

Next, we consider the computational complexity of Z and Z^F. This depends heavily on the search space. If $\mathcal{P} = (M, S_0, G)$ is a planning problem over an MDP $M = (S, A, app, Pr, C, R)$, then the search space for Fwd-VI on P is a digraph $\Gamma_\mathcal{P} = (N, E)$, where N is the transitive closure of $succ$ over S_0, and $E = \{(s, s') \mid s \in N, s' \in succ(s)\}$. For algorithms like RDTP and LRTDP, the search space is a subgraph of $\Gamma_\mathcal{P}$.

If F is an instance of Controlled-Plan, then the search space for Fwd-VI^F is $\Gamma^F_\mathcal{P} = (S^F, E^F)$, where S^F is as defined earlier, and $E^F = \{(s, s') \mid s \in S^F, s' \in succ^F(s)\}$.

The worst case is where $\Gamma_\mathcal{P} = \Gamma^F_\mathcal{P}$; this happens if F's search-control function, $\mathsf{acceptable}_F$, does not remove any actions from the search space.

On the other hand, there are many planning problems in which $\mathsf{acceptable}_F$ will remove a large number of applicable actions at each state in the search space (some examples occur in the next section). In such cases, $\mathsf{acceptable}_F$ can produce an exponential speedup, as illustrated in the following simple example.

Suppose $\Gamma_\mathcal{P}$ is a tree in which every state at depth d is a goal state, and for every state of depth $< d$, there are exactly b applicable actions and each of those actions has exactly k possible outcomes. Then $\Gamma_\mathcal{P}$'s branching factor is bk, so the number of nodes is $\Theta((bk)^d)$. Next, suppose F's search-control function eliminates exactly half of the actions at each state. Then $\Gamma^F_\mathcal{P}$ is a tree of depth d and branching factor $(b/2)k$, so it contains $\Theta(((b/2)k)^d)$ nodes. In this case, the ratio between the

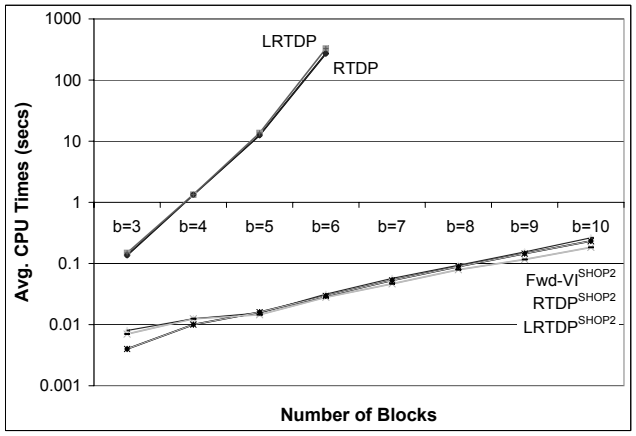

Figure 3: Running times for PBW using h_{500}, plotted on a semi-log scale. With 6 blocks ($b = 6$), the modified algorithms are about 10,000 times as fast as the original ones.

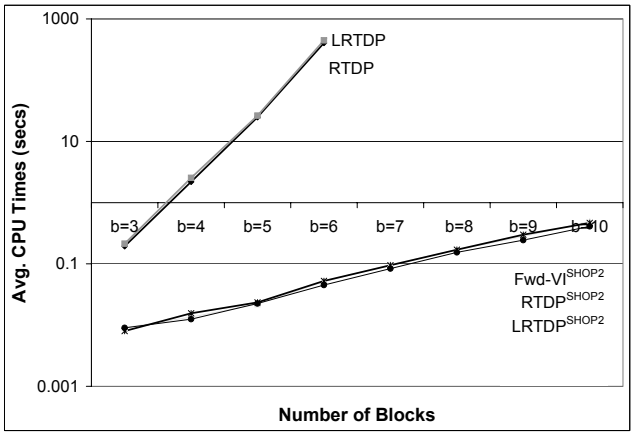

Figure 4: Running times for PBW using h_{max}, plotted on a semi-log scale. Like before, when $b = 6$ the modified algorithms are about 10,000 times as fast as the original ones.

number of nodes visited by Fwd-VI and Fwd-VI^F is 2^d, so Fwd-VI^F is exponentially faster than Fwd-VI.

Experimental Evaluation

For our experiments, we used Fwd-VI, RTDP, and LRTDP, and their enhanced versions Fwd-VI$^{\text{SHOP2}}$, RTDP$^{\text{SHOP2}}$, and LRTDP$^{\text{SHOP2}}$. For meaningful tests of the enhanced algorithms, we needed planning problems with much bigger state spaces than in prior published tests of RTDP and LRTDP. For this purpose, we chose the following two domains.

One was the Probabilistic Blocks World (PBW) from the 2004 International Probabilistic Planning Competition, with a 15% probability that a pickup or putdown action would drop the block on the table. The size of the state space grows combinatorially with the number of blocks: with 3 blocks there are only 13 states, but with 10 blocks there are 58,941,091 states.

The other was an MDP adaptation of the Robot Navigation domain (Kabanza, Barbeau, & St-Denis 1997; Pistore, Bettin, & Traverso 2001). A building has 8 rooms and 7 doors. Some of the doors are called *kid doors*. Whenever a kid door is open, a "kid" can close it randomly with a probability of 0.5. If the robot tries to open a closed kid door, this action may fail with a probability of 0.5 because the kid immediately closes the door. Packages are distributed throughout the rooms, and need to be taken to other rooms. The robot can carry only one package at a time. When there are 5 packages, the state space contains 54,525,952 states.

In both domains, we used a reward of 500 for goal states and 0 for all other states, a cost of 1 for each action, a discount factor $\gamma = 1.0$, and a termination criterion $\epsilon = 10^{-8}$.

The RTDP and LRTDP algorithms use domain-independent heuristics to initialize their value functions. We used two such heuristics. One, h_{500}, initializes the value of every state to 500. The other, h_{max}, is Bonet & Geffner's (2003) h_{min} heuristic adapted to work on maximization problems:

$$Q(s,a) = R(s) - C(s,a) + \max_{s' \in results(s,a)} Pr(s,a,s')\ V(s') \qquad (3)$$

We implemented all six of the planners in Lisp,[2], and tested them on a HP Pavilion N5415 with 256MB memory running Linux Fedora Core 2.

On most of the problems, Fwd-VI failed due to memory overflows, so we do not report any results for it. Figures 3 and 4 show the average running times of all the other five planners in the PBW domain. Each running time includes the time needed to compute h_{500} or h_{max}, and each data point is the average of 20 runs. The running times for RTDP and LRTDP were almost the same, and so were those of the three enhanced planners. Every algorithm's running time grows exponentially, but the growth rate is much smaller for the enhanced algorithms than for the original ones—for example, at $b = 6$ they have about 1/10,000 of the running time of the original algorithms. Once we got above 6 blocks (4,051 states in the state space), the original algorithms ran out of memory.[3] In contrast, the modified algorithms could easily have handled problems with more than 10 blocks (more than 58,941,091 states).

[2]The authors of RTDP and LRTDP were willing to let us use their C++ implementations, but we needed LISP in order to use SHOP2' search-control mechanism.

[3]Each time RTDP or LRTDP had a memory overflow, we ran it again on another problem of the same size. We omitted each data point on which there were more than five memory overflows, but included the data points where it happened 1 to 4 times. Thus our data make the performance of RTDP and LRTDP look better than it really was—but this makes little difference since they performed so much worse than RTDP$^{\text{SHOP2}}$ and LRTDP$^{\text{SHOP2}}$.

Table 1: Running times using h_{500} on Robot-Navigation problems with one kid door. p is the number of packages. Each data point is the average of 20 problems.

$p =$	1	2	3	4	5
RTDP	10.21	254.64	-	-	-
LRTDP	11.80	1622.68	-	-	-
Fwd-VI$^{\text{SHOP2}}$	0.01	0.03	0.06	0.08	0.14
RTDP$^{\text{SHOP2}}$	0.01	0.02	0.05	0.07	0.11
LRTDP$^{\text{SHOP2}}$	0.02	0.03	0.06	0.09	0.17

Table 2: Running times using h_{max} on Robot-Navigation problems with one kid door. p is the number of packages. Each data point is the average of 20 problems.

$p =$	1	2	3	4	5
RTDP	23.85	629.46	-	-	-
LRTDP	15.08	383.17	-	-	-
Fwd-VI$^{\text{SHOP2}}$	0.01	0.03	0.09	0.14	0.25
RTDP$^{\text{SHOP2}}$	0.01	0.03	0.08	0.13	0.22
LRTDP$^{\text{SHOP2}}$	0.01	0.04	0.09	0.14	0.26

Table 3: Running times using h_{max} on Simplified Robot-Navigation problems, with no kid doors. p is the number of packages. Each data point is the average of 20 problems.

$p =$	1	2	3	4	5
RTDP	4.69	211.22	-	-	-
LRTDP	4.66	209.87	-	-	-
Fwd-VI$^{\text{SHOP2}}$	0.01	0.03	0.05	0.09	0.15
RTDP$^{\text{SHOP2}}$	0.01	0.02	0.04	0.08	0.14
LRTDP$^{\text{SHOP2}}$	0.01	0.03	0.05	0.09	0.15

The reason for the fast performance of the modified algorithms is that in an HTN planner like SHOP2, it is very easy to specify domain-specific (but problem-independent) strategies like "if there is a clear block that you can move to a place where it will never need to be moved again, then do so without considering any other actions," and "if you drop a block on the table, then pick it up again immediately." Such strategies reduce the size of the search space tremendously.

Tables 1 and 2 show the running times for the planners in the Robot Navigation domain. The times for RTDP and LRTDP grew quite rapidly, and they were unable to solve many of the problems at all because of memory overflows. RTDP$^{\text{SHOP2}}$ and LRTDP$^{\text{SHOP2}}$ had no memory problems, and their running times were quite small.

To try to alleviate RTDP's and LRTDP's memory problems, we created a Simplified Robot Navigation domain in which there are no kid doors. When the robot tries to open a door, it may fail with probability 0.1, but once the door is open it remains open.

Table 3 shows the results on this simplified domain using h_{max}. RTDP and LRTDP took less time than before, but still had memory overflows. As before, RTDP$^{\text{SHOP2}}$ and LRTDP$^{\text{SHOP2}}$ had no memory problems and had very small running times.

An explanation about the performance of RTDP and LRTDP in our experiments is in order. LRTDP is an extension of RTDP that uses a labeling mechanism to mark states whose values have converged so that the algorithm does not visit them again during the search process. In most our problems, we observed that the value of a state did not converge until towards the end of the planning process. In each such case, LRTDP spent a significant portion of its running time to unsuccessfully attempt to label the states it visited. As a result, RTDP was able to perform better than LRTDP in those problems since it is free from such overhead.

Related Work

In addition to the RTDP and LRTDP algorithms described earlier, another similar algorithm is LAO* (Hansen & Zilberstein 2001), which is based on the classical AO* search algorithm. LAO* can do Value Iteration or Policy Iteration in order to update the values of the states in the search space, and can generate optimal policies under certain conditions.

Domain-specific knowledge has been used to search MDPs in reinforcement learning research (Parr 1998; Dietterich 2000). These approaches are based on hierarchical abstraction techniques that are somewhat similar to HTN planning. Given an MDP, the hierarchical abstraction of the MDP is analogous to an instance of the decomposition tree that an HTN planner might generate. However, the abstractions must be supplied in advance by the user, rather than being generated on-the-fly by the HTN planner.

In *envelope-based* MDP planning (Dean *et al.* 1995), an *envelope* of an MDP M is a smaller MDP $M' \subseteq M$. Envelope-based planning algorithms are anytime algorithms. An envelope-based planner begins with an initial envelope, and computes an optimal policy for this envelope. On subsequent iterations, it does the same thing on larger and larger envelopes, stopping when time runs out or when the current envelope contains all of M. The initial envelope is typically generated by a search algorithm, which often is a classical planning algorithm such as Graphplan. This suggests that domain-configurable planning algorithms such as TLPlan and SHOP2 would be good candidates for the search algorithm, but we do not know of any case where they have been tried.

Our ideas in this paper, and in particular the notion of a search-control function, were influenced by the work of Kuter & Nau (2004), in which they described a way to generalize a class of classical planners and their search-control functions for use in *nondeterministic planning domains*, where the actions have nondeterministic effects but no probabilities for state transitions are known.

Conclusions and Future Work

In this paper, we have described a way to take any forward-chaining MDP planner, and modify it to include the search-control algorithm from a forward-chaining domain-configurable planner such as TLPlan,

SHOP2, or TALplanner.

If the search-control algorithm satisfies an "admissibility" condition, then the modified MDP planner is guaranteed to find optimal solutions. If the search-control algorithm generates a smaller set of actions at each node than the original MDP algorithm did, then the modified planner will run exponentially faster than the original one.

To evaluate our approach experimentally, we have taken the search-control algorithm from the SHOP2 planner (Nau *et al.* 2003), and incorporated it into three MDP planners: RTDP (Bonet & Geffner 2000), LRTDP (Bonet & Geffner 2003), and Fwd-VI, a forward-chaining version of Value Iteration (Berstekas 1995). We have tested the performance of the modified algorithms in two MDP planning domains. For the original planning algorithms, the running time and memory requirements grew very quickly as a function of problem size, and the planners ran out of memory on many of the problems. In contrast, the modified planning algorithms had no memory problems and they solved all of the problems very quickly.

In the near future, we are planning to do additional theoretical and experimental analyses of our technique. We also are interested in extending the technique to MDPs that have continuous state spaces; such state spaces often arise in fields like control theory and operations research. We are currently working on that problem jointly with researchers who specialize in the fields of control theory and operations research.

Acknowledgments

This work was supported in part by NSF grant IIS0412812, DARPA's REAL initiative, and ISR seed funding. The opinions expressed in this paper are those of authors and do not necessarily reflect the opinions of the funders.

References

Bacchus, F., and Kabanza, F. 2000. Using Temporal Logics to Express Search Control Knowledge for Planning. *Artificial Intelligence* 116(1-2):123–191.

Bacchus, F. 2001. The AIPS '00 Planning Competition. *AI Magazine* 22(1):47–56.

Berstekas, D. 1995. *Dynamic Programming and Optimal Control*. Athena Scientific.

Bonet, B., and Geffner, H. 2000. Planning with Incomplete Information as Heuristic Search in Belief Space. In *AIPS-00*, 52–61.

Bonet, B., and Geffner, H. 2003. Labeled RTDP: Improving the Convergence of Real-Time Dynamic Programming. In *ICAPS-03*, 12–21.

Boutilier, C., and Goldszmidt, M. 1996. The Frame Problem and Bayesian Network Action Representation. In *Canadian Conference on AI*, 69–83.

Currie, K., and Tate, A. 1991. O-Plan: The Open Planning Architecture. *Artificial Intelligence* 52(1):49–86.

Dean, T.; Kaelbling, L. P.; Kirman, J.; and Nicholson, A. 1995. Planning under Time Constraints in Stochastic Domains. *Artificial Intelligence* 76(1–2):35–74.

Dietterich, T. G. 2000. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *JAIR* 13:227–303.

Fox, M., and Long, D. 2002. International Planning Competition. http://www.dur.ac.uk/d.p.long/competition.html.

Hanks, S., and McDermott, D. 1993. Modeling a Dynamic and Uncertain World I: Symbolic and Probabilistic Reasoning about Change. Technical Report TR-93-06-10, U. of Washington, Dept. of Computer Science and Engineering.

Hansen, E. A., and Zilberstein, S. 2001. LAO*: A Heuristic Search Algorithm that Finds Solutions With Loops. *Artificial Intelligence* 129:35–62.

Hoey, J.; St-Aubin, R.; Hu, A.; and Boutilier, C. 1999. SPUDD: Stochastic Planning using Decision Diagrams. In *UAI-99*.

Kabanza, F.; Barbeau, M.; and St-Denis, R. 1997. Planning Control Rules for Reactive Agents. *Artificial Intelligence* 95(1):67–113.

Kushmerick, N.; Hanks, S.; and Weld, D. S. 1994. An Algorithm for Probabilistic Planning. *Artificial Intelligence* 76(1-2):239–286.

Kuter, U., and Nau, D. 2004. Forward-Chaining Planning in Nondeterministic Domains. In *AAAI-04*, 513–518.

Kvarnström, J., and Doherty, P. 2001. TALplanner: A Temporal Logic-based Forward-chaining Planner. *Annals of Math and AI* 30:119–169.

Littman, M. L. 1997. Probabilistic Propositional Planning: Representations and Complexity. In *AAAI/IAAI Proceedings*, 748–761.

Nau, D.; Au, T.-C.; Ilghami, O.; Kuter, U.; Murdock, W.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN Planning System. *JAIR* 20:379–404.

Parr, R. 1998. *Hierarchical Control and learning for Markov decision processes*. Ph.D. Dissertation, Univ. of California at Berkeley.

Pistore, M.; Bettin, R.; and Traverso, P. 2001. Symbolic Techniques for Planning with Extended Goals in Nondeterministic Domains. In *ECP-01*.

Wilkins, D. 1990. Can AI Planners Solve Practical Problems? *Computational Intelligence* 6(4):232–246.