FSP*: Optimal Forward Stochastic Planning using relaxed PPDDL operators

Florent Teichteil-Königsbuch and **Guillaume Infantes**
ONERA-DCSD
2 Avenue Édouard-Belin – BP 74025
31055 Toulouse Cedex 4
France

Ugur Kuter
University of Maryland
Department of Computer Science and
Institute for Advanced Computer Studies,
College Park, Maryland 20742, USA

Introduction

Most of the existing planning algorithms use admissible heuristics in order to generate fast solutions to planning problems, without sacrificing optimality. This approach especially has been very powerful in deterministic planning domains. Some recent works such as (Bryce, Kambhampati, and Smith 2006; Hoffmann and Brafman 2005) has generalized the use of heuristics for planning under nondeterminism, where actions may have more than one possible outcome, but there are no probabilities, rewards, and costs.

Despite these strides in the use of heuristics for planning, planning with heuristics in stochastic domains is still open to further research. In this paper, we describe:

- An optimal forward stochastic planner, FSP*, that uses admissible heuristics for MDP planning. FSP* is based on the IMPROVED-LAO* algorithm (Hansen and Zilberstein 2001); it does a forward-only heuristic search starting from the initial states of an MDP and computes the value of a partial policy using dynamic programming techniques, as in IMPROVED-LAO*. The primary difference between FSP* and IMPROVED-LAO* is the termination criterion for planning. IMPROVED-LAO* stops when the Bellman-error of the states explored so far is below a given threshold, whereas FSP* stops when the set of states, which are reachable with the best current policy, does not change anymore. While this seems to be a slight difference, our experiments demonstrated strong differences between IMPROVED-LAO* and FSP* in terms of memory usage and computation time. Nevertheless, there was no clear winner because in some domains FSP* outperformed IMPROVED-LAO*, and vice versa.
- New admissible heuristics based on relaxation techniques inspired by classical planners like FF. Our heuristics include a state-abstraction technique for MDPs that allows us to efficiently compute upper and lower bounds on the value function for pruning, a generalization of FF's planning graphs for MDP planning problems.
- A way to compute the partial policies during planning using a partial state representation, in contrast with existing optimal stochastic planners of the previous International Probabilistic Planning Competitions which were mostly exploiting symbolic representations based on Binary and Algebraic Decision Diagrams. Our preliminary experiments with symbolic representations demonstrated that planners such as RTDP (Feng, Hansen, and Zilberstein 2003), and LAO* and IMPROVED-LAO* (Feng and Hansen 2002) usually solved MDP problems more efficiently by using explicit state representations than using symbolic ones.

Background

We consider MDP planning problems of the form $P = (\mathcal{S}, \mathcal{A}, T, \gamma, Pr, Re, \mathcal{S}_0, \mathcal{G})$, where $\mathcal{S}$ is a finite set of states and $\mathcal{A}$ is a finite set of actions. $T : \mathcal{S} \times \mathcal{A} \to 2^{\mathcal{S}}$ is a transition function, γ is a discount factor, $Pr : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0; 1]$ is a transition-probability function, $Re : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is a bounded one-stage reward function, $\mathcal{S}_0$ is a set of initial states, and $\mathcal{G}$ is a set of goal states. The set of all actions applicable to a state s is $\mathcal{A}_T(s) = \{a \in \mathcal{A} : T(a, s) \neq \emptyset\}$.

Although a policy is often defined to be a function $\pi : \mathcal{S} \to \mathcal{A}$, π does not need always be total. If a state in $\mathcal{S}$ is unreachable from $\mathcal{S}_0$ using π, then it can safely be omitted from the domain of π. Thus, we define a policy to be a partial function from $\mathcal{S}$ into $\mathcal{A}$ (i.e., a function from some set $\mathcal{S}_\pi \subseteq \mathcal{S}$ into $\mathcal{A}$) such that $\mathcal{S}_0 \subseteq \mathcal{S}_\pi$ and $\mathcal{S}_\pi$ is *closed* under π and $\mathcal{A}$ (i.e., if $s \in \mathcal{S}_\pi$ and $s' \in T(s, \pi(s))$, then $s' \in \mathcal{S}_\pi$). If $a \in \mathcal{A}_T(s)$, then $Pr(s, a, s')$ is the probability of going to the state s' if one applies the action a in s.

Given a policy π, the *value function* $V^\pi(s)$ is the expected sum of the future discounted rewards, i.e., $V^\pi(s) = E_\pi[\sum_{t=0}^{\infty} \gamma^t \; Re(s_t, \pi(s_t),) \; | s_0 = s]$, where s_t is the state of the MDP at time t, and $Re(\cdot)$ is understood with respect to the sample space induced by the transition probabilities.

An *optimal solution* is a policy π^* such that when executed in the initial state s_0, π^* maximizes the value function. It is well-known (Puterman 1994) that the optimal value $V^{\pi^*}(s)$ for state s can be computed by solving

$$V^{\pi^*}(s) = \max_{a \in \mathcal{A}_T(s)} Q(s, a)$$

$$Q(s, a) = \sum_{s' \in T(s,a)} Pr(s, a, s') \left(\gamma V^{\pi^*}(s') + Re(s, a, s') \right)$$

FSP*: Optimal Forward Stochastic Planning

FSP* is a forward search algorithm that uses any admissible heuristic to expand the states. An admissible heuristic is a function $h : \mathcal{S} \rightarrow \mathbb{R}$ such that for any state s, $h(s) \geqslant V^{\pi^*}(s)$. The closer h is to V^{π^*}, the less states are explored by FSP*. In this section, we assume the existence of such heuristic. In the following section, we will present different domain-independent heuristics based on relaxed PPDDL operators we developed.

FSP*' pseudo-code is presented in Algorithm 1. The version of FSP* used in the competition is based on a graph implementation defined as:

- a graph is a mapping from PPDDL states to nodes ;
- a PPDDL state is a set of true predicates ;
- a node is a tuple containing the current best action and best value in this state, as well as a mapping from applicable actions in this state to a list of all stochastic outcomes (edges) for each action ;
- an edge (outcome) is a tuple containing the probability, the reward, and the next state of the outcome.

This data structure allows us to generate on-the-fly a graph G of states which have been explored so far by FSP*. A node in G is a *tip node* if it does not have any outgoing edges.

FSP* first initializes the graph G to contain only the initial state I (lines 1 to 3). Then, the planner alternates between the following two phases until convergence:

1. optimization of the states reachable by the best current policy (line 5) ; any dynamic-programming technique can be used to optimize the policy of these states ;
2. computation of the set of states which are reachable by the (previously updated) best current policy until reaching either goal states or non-expanded states (lines 6 to 26).

The algorithm convergences to an optimal policy when the set of states reachable by the current policy does not change anymore (all reachable states were previously reachable).

During the forward search phase, the expansion of tip nodes (line 18) is crucial for performances. This function, detailed in Algorithm 2, is responsible for creating and adding new transitions to the tip nodes, and for initializing values of new created nodes with an admissible heuristic function (line 10). We use `mdpsim` as a black-box whose input is a set of true predicates (graph node) and whose output is a set of applicable actions and stochastic outcomes. In the following, we present three different domain-independent admissible heuristics for MDPs based on relaxation of PPDDL states and operators, inspired by FF's problem relaxation technique.

Admissible Heuristics in FSP*

Heuristics computation usually requires to explore all paths which start from a given state. Mathematically, this is equivalent to exploring all states which are reachable from a given state by successively applying all possible actions (breadth-first search). While enumerating all reachable states may result in blowing up the computer's memory, it is possible to discard all delete effects so that reachable states are instead abstractly represented as a set of true predicates. Instead of memorizing a set of states, we now only memorize a single set of true predicates, which grows as long as new reach-

Algorithm 1: FSP*

```
   // I: initial state
   // G = (nodes_map): explored states
   // nodes_map = state ⇝ (action, value, edges_map)
   // edges_map = A_T(state) ⇝ edges_list
   // edge = (probability, reward, node)
   // T: set of tip nodes
   // R: set of nodes reachable with the
      best current policy
   // F: forward search frontier
 1 G.nodes ← {(I, nil)};
 2 expand_node(G.nodes_map.get(I));
 3 R ← {G.nodes_map.get(I)};
 4 repeat
 5     optimize(R, G);
 6     prevR ← R;
 7     R ← {G.nodes_map.get(I)};
 8     F ← {G.nodes_map.get(I)};
 9     continue_search ← false;
10     repeat
11         prevF ← F;
12         for s ∈ prevF do
13             for e ∈ s.edges_map[s.action] do
14                 if e.node ∉ prevR then
15                     continue_search ← true;
16                 end
17                 if e.node.state ∈ T then
18                     expand(e.node.state, G, T);
19                     T.remove(e.node);
20                 else if s' ∉ R then
21                     F.add(e.node);
22                     R.add(e.node);
23                 end
24             end
25         end
26     until F = ∅ ;
27 until continue_search = false ;
```

Algorithm 2: $expand(s, G, \mathcal{T})$ function

```
 1 n ← G.nodes_map.get(s);
 2 for a ∈ A_T(s) do
 3     el ← n.edges_map.insert(a, nil);
 4     for s' ∈ T(s, a) do
 5         if s' ∈ G.nodes_map then
 6             n' ← G.nodes_map.get(s')
 7         else
 8             n' ← G.nodes_map.insert(s', nil);
 9             T.add(n');
10             n'.value ← compute_heuristic(s');
11         end
12         el.insert(Pr(s, a, s'), Re(s, a, s'), n');
13     end
14 end
```

able states are explored. Formally, if $\mathcal{R}$ is the implicit set of reachable states, $True(s)$ is the set of all ground atoms that are true in a state s, and $TrueSet = \{p_1, \cdots, p_k\}$ is the explicit set of true predicates, then:

$$\forall s \in \mathcal{R},\ True(s) \subseteq TrueSet.$$

$TrueSet$ is a relaxation of $\mathcal{R}$ because it may contain more states than the actual reachable states.

We define a relaxed transition in an MDP as the tuple (a, pre, eff, r) where a is a PPDDL action, pre is the disjunction of all positive atoms in the preconditions of a, eff is the disjunction of all positive effects of a (i.e., the atoms that become true in the world state after a is applied), and r is the one-step reward-to-go value for a. Such transition is relaxed because some actions may become applicable in states where they are not applicable in the real domain.

Relaxed Distance Heuristic (RDH) This heuristic only applies for shortest stochastic path problems (i.e. rewards discarded). We assume all transitions have an immediate -1 reward, except transitions to goal states, which have a zero reward. Given a state s, if d is a lower bound on the length of all paths to the goals starting in s, then $h(s) = -\sum_{k=0}^{d-2} \gamma^k$ is an admissible heuristic value of s.

Precisely, the above domain relaxation allows us to easily compute such a distance lower bound d. We start constructing $TrueSet$ as the set of all true predicates in s. Then, we successively apply our relaxed transition operator on $TrueSet$ until either all true predicates of some goal state are included in $TrueSet$, or $TrueSet$ is stable. The number of iterations until convergence is our distance lower bound.

Relaxed Reward Heuristic (RRH) This heuristic is a generalization of RDH to general MDPs. Given a state s, assuming $\mathcal{T}(s)$ is the set of all reachable states starting in s, we can prove the following formula:

$$V^{\pi^*}(s) \leqslant \frac{\max_{s' \in \mathcal{T}(s)} \max_{a \in \mathcal{A}_T(s')} \max_{s'' \in T(s',a)} r(s', a, s'')}{1 - \gamma}$$

This upper bound can be easily approximated by replacing all exact operators by relaxed operators applied on $TrueSet$. Like RDH, $TrueSet$ is iteratively computed by applying successive relaxed operators until it becomes stable. The maximum over all relaxed rewards gathered during $TrueSet$ expansion is an upper bound of the above formula's numerator.

Relaxed Bound Heuristic (RBH) Like RRH, this heuristic is designed for general MDPs. While easily computable, RRH is not very informative because it tends to give the same heuristic value to all states.

A *relaxed MDP graph* is a directed, acyclic, and leveled graph in which each level consists of a set of relaxed MDP transitions as described above. The preconditions of each relaxed transition at level i must be provided by the effects of at least one relaxed transition at level $i-1$. In the first level of the relaxed MDP graph, the preconditions of the relaxed MDP transitions must appear in the set of atoms of the initial state(s) of the MDP.

A relaxed MDP graph M provides a framework to compute value estimates of propositions in the MDP. Suppose p is a proposition in the MDP planning problem that appears as a precondition of an action a in the graph; i.e., p is in pre of some relaxed transition (a, pre, eff, r) in the graph. Suppose the transition t is at level i. Then, we have

$$V(p) = r + max_{q \in eff}(\gamma\, max_{(a', pre', eff', r') \in M, q \in pre'} V(q)),$$

where t' is a relaxed transition at level $i+1$ in the relaxed MDP graph M and γ is the discount factor. The value of a transition t in M is the maximum of the values of the propositions that appear in the preconditions of that transition: i.e., $V(t = (a, pre, eff, c)) = max_{P \in pre} V(p)$.

Given an MDP planning problem with an initial state (or a set of initial states) and a set of goal atoms, the heuristic first generates a relaxed MDP graph in a forward expansion phase, in which it successively generates new transition levels until goals are reached or there are no new transitions can be generated. Then, the heuristic does a backward search on the graph M to compute the values of the transitions. Finally, the value of a state s, $V(s)$, is defined as follows:

$$V(s) = max_{p \in True(s)} V(p).$$

Note that RBH is an improvement on RRH described above since RBH keeps track of the successive computed $TrueSet$s in a relaxed MDP graph, so that backtracking through this graph provides value estimates of the predicates in the initial $TrueSet$.

Acknowledgments. This research was supported in part by the French *Délégation générale pour l'armement* grant 07.60.031.00.470.75.01., DARPA's Transfer Learning and Integrated Learning programs, and NSF grant IIS0412812. The opinions in this paper are those of the authors and do not necessarily reflect the opinions of the funders.

References

Bryce, D.; Kambhampati, S.; and Smith, D. E. 2006. Planning Graph Heuristics for Belief Space Search. *Journal of Artificial Intelligence Research* 26:35–99.

Feng, Z., and Hansen, E. A. 2002. Symbolic heuristic search for factored markov decision processes. In *AAAI/IAAI*.

Feng, Z.; Hansen, E. A.; and Zilberstein, S. 2003. Symbolic generalization for on-line planning. In *UAI*.

Hansen, E. A., and Zilberstein, S. 2001. LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence* 129(1-2):35–62.

Hoffmann, J., and Brafman, R. 2005. Contingent Planning via Heuristic Forward Search with Implicit Belief States. In *ICAPS-05*.

Puterman, M. L. 1994. *Markov Decision Processes*. John Wiley & Sons, INC.