Determinize, Solve, and Generalize: Classical Planning for MDP Heuristics

Andrey Kolobov Mausam Daniel S. Weld
{akolobov, mausam, weld}@cs.washington.edu
Dept of Computer Science and Engineering
University of Washington, Seattle
WA-98195

Abstract

Heuristics make MDP solvers practical by reducing their space and memory requirements. Some of the most effective heuristics (e.g. the FF heuristic) first determinize the MDP to a classical approximation and then solve a relaxation of the resulting classical problem (e.g., one which ignores the actions' delete effects). While these heuristics can be computed quite quickly, they frequently yield overly-optimistic value estimates.

This paper proposes a novel class of heuristics, called THUDS, which improve on the existing methods by using full-fledged classical planners to solve the non-relaxed determinizations. THUDS produces more informative state value estimates than those given by the FF heuristic, causing many fewer states to be explored. Of course, invoking a deterministic planner can be very slow; to overcome this high cost THUDS *generalizes* the heuristic value of one state to many others by extracting *basis functions* from the plans discovered in the process of heuristic computation. Thus, the classical planner is only called for states without basis functions — amortizing its costly invocation. Experiments show that THUDS can provide large time and memory savings compared to the FF heuristic and that generalization is vital in making THUDS computationally feasible.

INTRODUCTION

Heuristics are a popular means of reducing space and memory requirements of probabilistic planning algorithms. In heuristically guided MDP solvers, e.g. LRTDP [1] and LAO* [4], heuristics help avoid many states (or state-action pairs) that are not part of the optimal policy.

While there are many ways of constructing a good heuristic function, some of the most effective ones were taken from classical planning. A notable example is the FF heuristic [5], which we will refer to as h_{FF} in this paper. They presently rely on *determinizing* the MDP at hand and trying to solve the resulting classical planning problems. However, since deterministic planning is hard in itself, the heuristics relax the problem further in various ways. h_{FF}, for instance, ignore the *delete* lists of all the actions and look for the cheapest sequence of these modified actions that brings it to the goal. The cost of such a sequence is an approximation of the true optimal cost of a solution to the deterministic problem and is taken as the heuristic value of the state where the sequence originates. However, since the delete effects are ignored, these heuristics don't take negative interactions among actions into account, and as a result often yield overly optimistic plan cost estimates.

In this paper, we propose a new class of heuristics called THUDS (**T**emplate for **H**euristics that **U**se **D**eterministic **S**olvers). THUDS heuristics solve the *non-relaxed* determinizations of an MDP with full-fledged classical planners. We empirically demonstrate that such heuristics are typically more informative than h_{FF} and provide greater space savings by causing fewer states to be explored. However, even though modern classical planners are very efficient, invoking them for *every* explored state would be too costly. To cope with this issue, we *generalize* the heuristic values of some states to many others by using conjunctions of literals called *basis functions*. The basis functions are computed by regressing each of the deterministic plans obtained as part of the heuristic computation, and are therefore $certificates$ of positive-probability trajectories in the original problem. Any state that subsumes a basis function is thus proved to have the corresponding trajectory to the goal. Minimizing over costs of these trajectories yields a heuristic estimate for the state's value. We call the deterministic planner (and hence augment the basis function set) only from the states which subsume none of the basis functions we have so far. The set of basis functions we accrue requires little extra space but lets us amortize the time cost of using the deterministic planner. Thus, the primary novelty of our work is the insight that we can obtain informative heuristic estimates offered by classical planners while avoiding a potentially enormous overhead of the deterministic planner invocation by using the cached basis functions. Our experimental results show that THUDS can reduce the time and memory consumption of probabilistic planning algorithms by orders of magnitude more than the FF heuristic. They also demonstrate that generalization is vital for THUDS's performance, making THUDS at least 20-25 times faster than our heuristic would be without it. We conclude by identifying limitations of our heuristic and ways of addressing them.

BACKGROUND

MDPs. In this paper, we focus on probabilistic planning problems that are modeled by factored indefinite-horizon MDPs. They are defined as tuples of the form $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{C}, \mathcal{G}, s_0 \rangle$, where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a finite set of actions, $\mathcal{T}$ is a transition function $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ giving the probability of moving from s_i to s_j by exe-

cuting a, $\mathcal{C}$ is a map $\mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}^+$ specifying action costs, s_0 is the start state, and $\mathcal{G}$ is a set of (absorbing) goal states. *Indefinite horizon* refers to the fact that the total action cost is accumulated over a finite-length action sequence whose length is unknown.

In factored MDPs, each state is represented as a conjunction of values of the domain variables. Solving an MDP means finding a good (i.e. cost-minimizing) policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ that specifies the actions the agent should take to eventually reach the goal. The optimal expected cost of reaching the goal from a state s satisfies the following conditions, called *Bellman equations*:

$$V^*(s) = 0 \text{ if } s \in \mathcal{G}, \text{otherwise}$$
$$V^*(s) = \min_{a \in \mathcal{A}} [\mathcal{C}(s,a) + \sum_{s' \in \mathcal{S}} \mathcal{T}(s,a,s') V^*(s')]$$

Given $V^*(s)$, an optimal policy may be computed as follows: $\pi^*(s) = \text{argmin}_{a \in \mathcal{A}} [\mathcal{C}(s,a) + \sum_{s' \in \mathcal{S}} \mathcal{T}(s,a,s') V^*(s')]$.

Solution Methods. The above equations suggest a dynamic programming-based way of finding an optimal policy, called *value iteration* (VI), that iteratively updates state values using Bellman equations in a *Bellman backup* and follows the resulting policy until the values converge.

VI has given rise to many improvements. Trial-based methods, e.g. RTDP, try to reach the goal multiple times (in multiple *trials*) and update the value function over the states in the trial path using Bellman backups. A popular variant, LRTDP, adds a termination condition to RTDP by labeling those states whose values have converged as 'solved' [1]. Compared to VI, trial-based methods save space by considering fewer irrelevant states. LRTDP serves as the testbed in our experiments, but the approach we present can be used by many other search-based MDP solvers as well, e.g., LAO*.

Heuristics. We define a *heuristic* as a value function used to initialize the state values before the first time an algorithm updates these values. In heuristically guided algorithms, heuristics help avoid visiting irrelevant states. To guarantee convergence to an optimal policy, MDP solvers require a heuristic to be admissible, i.e. to never overestimate the optimal value of a state (importantly, admissibility is not a requirement for convergence to *a* policy). However, inadmissible heuristics tend to be more *informative* in practice, approximating V^* better on average. Informativeness often translates into a smaller number of explored states (and the associated memory savings) with reasonable sacrifices in optimality.

Determinization. Some of the most effective domain-independent heuristics known today are based on *determinizing* the probabilistic domain at hand. Determinizing domain D removes the uncertainty about D's action outcomes in a variety of ways. For example, the *all-outcomes* determinization, for each action a with precondition c and outcomes $o_1, \ldots, o_n$ with respective probabilities $p_1, \ldots, p_n$, produces a set of deterministic actions $a_1, \ldots, a_n$, each with precondition c and effect o_i, yielding a classical domain D_d. To obtain a value for state s in D, determinization heuristics try to approximate the cost of a plan from s to a goal in D_d (finding a plan itself is generally NP-hard). For instance, h_{FF} ignores the delete effects of all actions in D_d and attempts to find the cheapest solution to this relaxed problem.

THUDS ALGORITHM

Given a problem P over a probabilistic domain D, THUDS starts by determinizing D into its classic counterpart, D_d. This operation needs to be done only once, at initialization time. Our implementation performs the all-outcomes determinization because it is likely to give much better value estimates than the single-outcome one [11]. However, more involved flavors of determinization described in the Related Work section may yield even better estimation accuracy.

Calling a Deterministic Planner. Once D_d has been computed, the probabilistic planner starts exploring the state space. For every state s that needs to be evaluated heuristically, THUDS first checks if it is an *explicit dead end*, i.e. has no actions applicable in it. This check is in place for efficiency reasons. For states that aren't explicit dead ends, THUDS has a more sophisticated estimation method.

This method entails constructing a problem P_s with the original problem's goal and s as the initial state, feeding P_s along with D_d to a classical planner $DetPlan$, and setting a timeout (in our setup, 25 seconds). If s is an *implicit dead end* (i.e., has actions applicable in it but no plan to the goal), $DetPlan$ either quickly proves this or unsuccessfully searches for a plan until the timeout. In either case, it returns without a plan, at which point s is presumed to be a dead end and assigned a very high value. If s is not a dead end, $DetPlan$ usually returns a plan from s to the goal. The cost of this plan is taken as the heuristic value of s. In rare cases, $DetPlan$ may fail to find a plan before the timeout, leading the MDP solver to falsely assume s to be a dead end. In practice, we haven't seen this hurt THUDS' performance.

Regression-Based Generalization. By using a full-fledged classical planner, THUDS produces more informative state estimates than h_{FF}, as evidenced by our experiments. However, invoking the classical planner for every newly encountered state is costly; as it stands, THUDS would be prohibitively slow. To ensure speed we modify the procedure based on the following insight. Regressing a successful deterministic plan in domain D_d yields a set of literal conjunctions with an important property: each such conjunction is a precondition for the plan suffix that was regressed to generate it. We call these conjunctions *basis functions*, and define the *weight* of a basis function to be the cost of the plan it enables. *Crucially, every deterministic plan in D_d corresponds to a positive-probability trajectory in the original domain D*; therefore, a basis function is a certificate of such a trajectory. Every state subsumed by a given basis function is thus proved to have a possible trajectory to the goal.

We make this process concrete in the pseudocode of Algorithm 1. Whenever THUDS computes a deterministic plan, it regresses it and caches the resulting basis functions with associated weights. When THUDS encounters a new state s, it minimizes over the weights of all basis functions stored so far that subsume s. In doing so, THUDS sets the heuristic value of s to be the cost of the cheapest currently

Algorithm 1 THUDS

```
1: Input: probabilistic domain D, problem P = ⟨init. state s_0, goal G⟩, determinization routine Det, classical planner DetPlan
2: compute global determinization D_d = Det(D)
3: declare global map M from basis functions to weights
4:
5: function computeTHUDS(state s, timeout T)
6: if no action a of D is applicable in s then
7:    return a large penalty // e.g., 1000000
8: else if some member f' of M holds in s then
9:    return min_{basis functions f that subsume s} {M[f]}
10: else
11:    declare problem P_s ← ⟨init. state s, goal G⟩
12:    declare plan pl ← DetPlan(D_d, P_s, T)
13:    if pl == none then
14:       return a large penalty // e.g., 1000000
15:    else
16:       declare basis function f ← goal G
17:       declare weight ← 0
18:       for all i = length(pl) through 1 do
19:          declare action a ← pl[i]
20:          weight ← weight + Cost(s, a)
21:          f ← (f ∪ precond(a)) − effect(a)
22:          insert ⟨f, weight⟩ into M if f isn't in M yet
23:       end for
24:       return weight
25:    end if
26: end if
```

known trajectory that originates at s. Thus, the weight of one basis function can become *generalized* as the heuristic value of many states. This way of computing a state's value is very fast, and THUDS employs it *before* invoking a classical planner. The caveat, of course, is that by the time state s needs to be evaluated THUDS may have no basis functions that subsume it. In this case, THUDS uses the classical planner as described above, computing a value for s and augmenting its basis function set. Evaluating a state first by generalization and then, if generalization fails, by classical planning greatly amortizes the cost of each classical solver invocation and drastically reduces the computation time compared to using a deterministic planner alone.

Theoretical properties. Two especially noteworthy theoretical properties of THUDS are the informativeness of its estimates and its inadmissibility. The former ensures that, compared to h_{FF}, THUDS causes MDP solvers to explore fewer states. At the same time, THUDS is inadmissible, for three reasons. One source of inadmissibility comes from the general lack of optimality of deterministic planners. Even if they were optimal, however, employing timeouts to terminate the classical planner occasionally causes THUDS to falsely assume states to be dead ends. Finally, the basis function generalization mechanism also contributes to inadmissibility; the set of discovered basis functions is almost never complete, and hence even the smallest basis function weight known so far may be an overestimate of a state's true value. In spite of theoretical inadmissibility, in practice THUDS usually finds very good policies whose quality is often better than of those yielded by h_{FF}.

EXPERIMENTAL RESULTS

In our experiments we compare THUDS performance to h_{FF}, a representative determinization heuristic, across a wide range of domains. Our implementation of THUDS uses a portfolio of two classical planners, FF and LPG [2]. To evaluate a state, it launches both planners as in line 12 of Algorithm 1 in parallel and takes the heuristic value from the one that returns sooner. We tested THUDS and h_{FF} as a part of the LRTDP planner available in the miniGPT package. Our benchmarks were three probabilistic domains: Machine Shop [8], Triangle Tire (IPPC-08 version) and Exploding Blocksworld (IPPC-06 version), each having 10 problems. Additionally, we perform a brief comparison of LRTDP+THUDS against ReTrASE [7], since the latter uses basis functions in a somewhat related fashion.

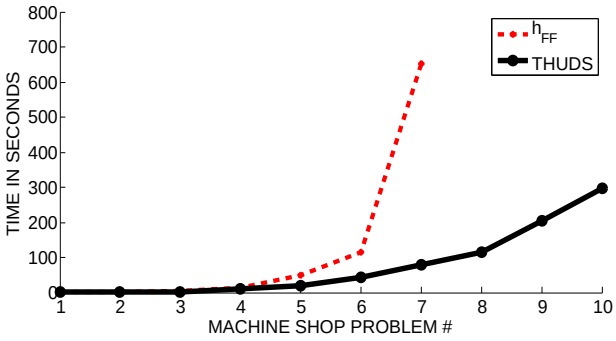
Comparison against h_{FF}. The Machine Shop domain involves two machines and the number of objects equal to the ordinal of the corresponding problem. Each object needs to go through a series of manipulations, of which each machine is able to do only a subset. The effects of some manipulations may cancel the effects of others (e.g., shaping an object destroys the paint sprayed on it). Thus, the order of actions in a plan is critical. This domain illuminates the drawbacks of h_{FF}, which ignores delete effects and doesn't distinguish good and bad action sequences as a result. Machine Shop has no dead ends.

Figure 1 shows the speed and memory performance of LRTDP equipped with the two heuristics. For LRTDP+THUDS, the memory consumption is measured by the number of states and basis functions whose values need to be maintained (THUDS caches basis functions and LRTDP caches states). In the case of LRTDP+h_{FF} all memory used is only due to LRTDP's state caching because h_{FF} by itself does not memoize anything. On Machine Shop, the edge of LRTDP+THUDS is clearly vast, reaching several orders of magnitude. In fact, LRTDP+h_{FF} runs out of memory on the three hardest problems, whereas LRTDP+THUDS is far from that.

Concerning the policy quality, we found the use of THUDS to yield optimal or nearly-optimal policies on Machine Shop. This contrasts with h_{FF} whose policies were on average 30% more costly than the optimal ones.

The Triangle Tire domain, unlike Machine Shop, doesn't have structure that is particularly inconvenient for h_{FF}. However, LRTDP+THUDS noticeably outperforms LRTDP+h_{FF} on it too, as Figure 2 indicates. Nonetheless, neither heuristic saves enough memory to let LRTDP solve past problem 8.

Results on Exploding Blocksworld are not as favorable for LRTDP+THUDS. Even though the use of THUDS does yield some memory savings, it also slows down LRTDP considerably. Statistics in Table 1 reveal the reason for this behavior. In this domain, a very large fraction of the explored states are implicit dead ends. Calling a deterministic planner on a dead-end state does not yield any basis functions, and thus does not add to the heuristic's



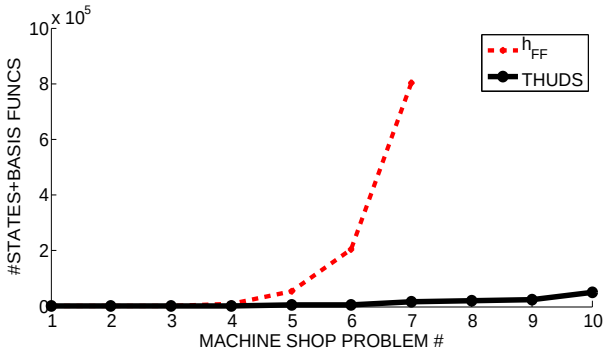
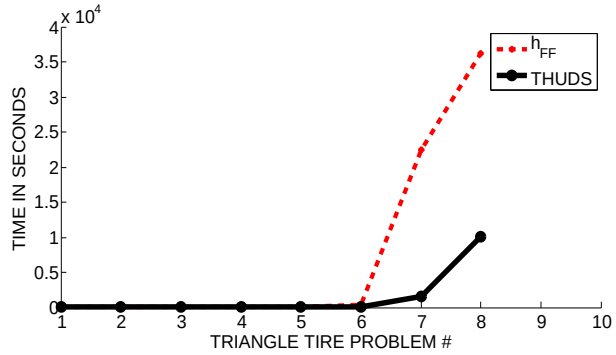


Figure 1: THUDS outperforms h_{FF} by a large margin both in speed and memory on Machine Shop...



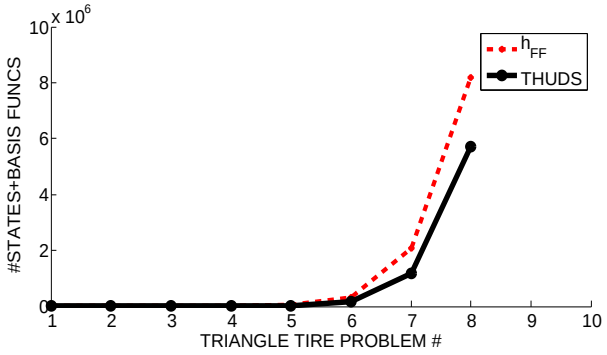


Figure 2: ...and on Triangle Tire.

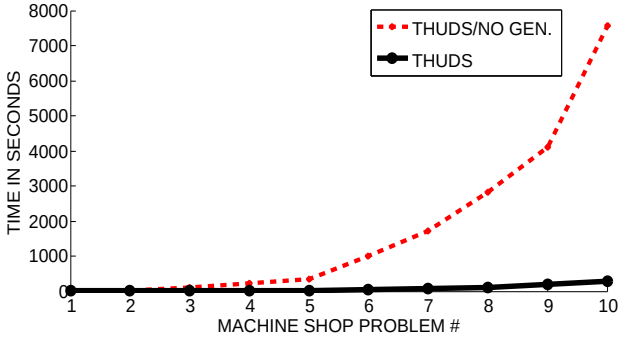


Figure 3: THUDS with generalization is much faster than without it.

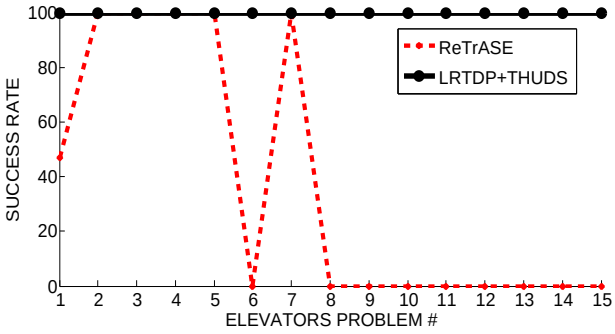


Figure 4: LRTDP+THUDS vastly outperforms ReTrASE on Elevators.

p1	p2	p3	p4	p5	p6	p7	p8	p9	p10
45	51	67	42	0	57	44	63	50	55

Table 1: Percentage of implicit dead ends in the explored state space across the first 10 problems of Exploding Blocksworld.

generalization ability. This causes the cost of using the deterministic planner to become prohibitive, since it is not well amortized. Therefore, in domains with many dead ends relevant to finding a good policy we expect the effectiveness of THUDS in its current form to be reduced. We outline a possible solution in the Discussion section.

Benefit of Basis Functions. To measure the significance of generalization in THUDS' operation, we also tested a version on THUDS with generalization turned off. It proved to be approximately 20-25 times slower than regular THUDS on Machine Shop (Figure 3), implying that without our generalization technique the speedup over h_{FF} would not have been possible.

Computational Profile. An interesting aspect of THUDS' modus operandi is the fraction of the computational resources an MDP solver uses that is due to THUDS. Across the Machine Shop domain, LRTDP+THUDS spends 75-90% of the time in heuristic computation, whereas LRTDP+h_{FF} only 8-17%. Thus, THUDS is computationally heavier but causes LRTDP to spend drastically less time exploring the state space.

Comparison against ReTrASE. Superficially, ReTrASE

extracts and uses basis functions in a way similar to THUDS. The major difference lies in the fact that ReTrASE tries to learn weights for the basis functions, whereas THUDS only employs basis functions to initialize state values and lets a conventional MDP solver improve on these values. In practice, this discrepancy translates to ReTrASE's learning procedure providing very few quality guarantees. While it is very memory-efficient on many hard problems, the solutions are poor on some domains with rather simple structure, e.g. Elevators from IPPC-06 [7]. In contrast, THUDS admits the use of conventional MDP solvers with strong theoretical machinery, making the outcome of its application more predictable. In particular, LRTDP+THUDS achieves a 100% success rate on all 15 Elevators problems (Figure 4) and takes at most 5 minutes per problem. As another example, LRTDP+THUDS achieves a 100% success rate on the first 8 problems of Triangle Tire. ReTrASE performs equally well on the first 8 problems but, unlike LRTDP+THUDS, can also solve problems 9 and 10. Thus, THUDS's use of basis functions yields qualitatively and quantitatively different results than ReTrASE's.

DISCUSSION

Promise shown by THUDS indicates several directions for its further development. An important issue we need to address is THUDS' underperformance on domains with many implicit dead ends. If THUDS could generalize dead ends, it would potentially need to save many fewer basis functions than it has discovered dead ends, dramatically increasing space savings. Crucially, dead end identification would not be so costly time-wise. Similarly to generalizing heuristic values of ordinary states with basis functions, we propose devising a procedure that would use known dead ends to learn logical formulas to characterize "dead-endness".

Another direction is experimenting with domain determinizations THUDS could rely on, e.g. the one proposed by the authors of HMDPP [6] and described in Related Work.

RELATED WORK

The use of determinization for solving MDPs was inspired by advances in classical planning, most notably the FF solver [5]. The practicality of the new technique was demonstrated by FF-Replan [11] that used the FF planner on an MDP determinization for direct selection of action to execute in a given state. More recent planners to employ determinization that are, in contrast to FF, successful at dealing with probabilistically interesting problems include RFF-RG/BG [10]. Unlike THUDS, they normally use deterministic planners to *learn* the state or action values and not just to initialize their values heuristically. As a consequence, they invoke FF many more times than we do. This, in turn, forces them to avoid all-outcome determinization as invoking FF would be too costly otherwise.

To a large degree, the FF planner owes its performance to h_{FF} [5]. LRTDP [1] and HMDPP [6] adopted this heuristic with no modifications as well. In particular, HMDPP runs h_{FF} on a "self-loop determinization" of an MDP, thereby forcing h_{FF}'s estimates to take into account some of the problem's probabilistic information.

Several algorithms generate basis functions by regression like we do, [3], [9], and [7] to name a few. However, the role of basis functions in them is entirely different. In these methods, basis functions serve to map the planning problems to smaller parameter spaces consisting of basis function weights. Parameter learning in such transformed spaces is usually approximate and gives few theoretical guarantees (see, for instance, [7]). In THUDS, basis functions are used to generalize heuristic values over multiple states and thereby to avoid invoking the classical planner too many times. Importantly, however, the parameter space in which learning takes place is unchanged — it is still the set of state values. We can therefore use conventional techniques like LRTDP in conjunction with THUDS that give substantial predictability of the solution quality. THUDS achieves the reduction in the number of required parameters through the increased informativeness of initial heuristic estimates, not through parameter space transformation.

CONCLUSION

We have proposed a new class of heuristics, THUDS, that use full-fledged deterministic planners to solve MDP determinizations. Although invoking a classical solver is expensive, this cost is amortized by using basis functions to generalize heuristic values across many states, thereby greatly reducing the number of times the deterministic planner has to used. The resulting heuristic is inadmissible but gives more informative state value estimates than h_{FF}, providing significant memory savings to the MDP solvers. In addition, generalization allows THUDS to yield speedups as well. We believe that extending the generalization procedure to handle dead ends will render THUDS an integral component of future successful planners.

References

[1] B. Bonet and H. Geffner. Labeled RTDP: Improving the convergence of real-time dynamic programming. In *ICAPS'03*, pages 12–21, 2003.

[2] A. Gerevini, A. Saetti, and I. Serina. Planning through stochastic local search and temporal action graphs. *Journal of Artificial Intelligence Research*, 20:239–290, 2003.

[3] C. Gretton and S. Thiebaux. Exploiting first-order regression in inductive policy selection. In *UAI'04*, 2004.

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

[5] J. Hoffman and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[6] E. Keyder and H. Geffner. The HMDPP planner for planning with probabilities. In *Sixth International Planning Competition at ICAPS'08*, 2008.

[7] A. Kolobov, Mausam, and D. Weld. ReTrASE: Integrating paradigms for approximate probabilistic planning. In *IJCAI'09*, 2009.

[8] Mausam, P. Bertoli, and D. Weld. A hybridized planner for stochastic domains. In *IJCAI'07*, 2007.

[9] S. Sanner and C. Boutilier. Practical linear value-approximation techniques for first-order mdps. In *UAI'06*, 2006.

[10] F. Teichteil-Koenigsbuch, G. Infantes, and U. Kuter. RFF: A robust, FF-based MDP planning algorithm for generating policies with low probability of failure. In *Sixth International Planning Competition at ICAPS'08*, 2008.

[11] S. Yoon, A. Fern, and R. Givan. FF-Replan: A baseline for probabilistic planning. In *ICAPS'07*, pages 352–359, 2007.