Integrating Paradigms for Approximate Probabilistic Planning

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

Humans are often able to solve extremely large probabilistic planning problems reasonably well by exploiting problem structure, heuristics, and various approximations. Each of these aspects seems indispensable for achieving high scalability and has been studied in detail by the automated planning community. However, most existing solvers use only a proper subset of them.

In an initial attempt to bridge this gap we introduce RETRASE, a novel MDP solver that derives approximate policies by extracting problem structure and learning its parameters under heuristic guidance. RETRASE uses classical planning to discover basis functions for value-function approximation and applies expected-utility analysis to this compact space. Experiments demonstrate that RETRASE outperforms winners from the past three probabilistic-planning competitions on many hard problems. We outline several extensions of RETRASE and new directions for unifying paradigms in probabilistic planning prompted by RETRASE's success.

1 INTRODUCTION

As humans, we are often able to solve probabilistic planning problems far larger than state-of-the-art planners are capable of. Even though we tend to miss the problems' probabilistic subtleties, our solutions are in many cases reasonably good. We find them by using heuristics and various approximations ("intuition"), and exploiting problem structure (i.e. disregarding weakly relevant information). Each of these aspects has been studied in detail by the automated planning community.

A popular framework for formulating probabilistic planning problems is Markov Decision Processes (MDPs). One of the most popular algorithms for solving MDPs that yields high-quality solutions is RTDP [1], a technique that explores the state space under the guidance of a heuristic. Unfortunately, being based on dynamic programming, it suffers from a critical drawback — it represents the value function extensionally, i.e., as a table, thus requiring memory (and time) exponential in the number of domain features.

Two broad approaches have been proposed to avoid creating a state/value table. One method involves *domain determinization* and uses a classical planner as a subroutine in computing a policy. Such determinization planners, e.g., FFReplan [11], tend to disregard the probabilistic nature of actions and often have trouble with *probabilistically interesting* [7] domains. In other words, their approximation, while computationally efficient, frequently results in poor solution quality.

The other method, *dimensionality reduction*, maps the state space to a parameter space of lower dimension. Typically, the mapping is done by constructing a small set of basis functions, learning weights for them, and combining the weighted basis function values into the values of states. This can be viewed as discovering problem structure while abstracting away unimportant details. Researchers have successfully applied dimensionality reduction to domains after manually defining a domain-specific mapping but automatic basis function discovery in nominal (e.g., "discrete" or "logical") domains, such as those used in the IPPC, remains a challenge.

Thus, each technique has its advantages but also drawbacks that prevents it from dominating others. In fact, all of them seem indispensable for achieving high scalability and acceptable solution quality simultaneously. However, most existing solvers use only a proper subset of them. This paper bridges the gap — proposing a fusion of these ideas that removes the drawbacks of each. Our algorithm RETRASE, which stands for **Re**gressing **Tra**jectories for **A**pproximate **S**tate **E**valuation, learns a compact value function approximation successful in a range of nominal domains. It discovers problem structure by planning in a determinized version of the domain at hand and thereby automatically obtaining a set of basis functions, learns the weights for these basis functions by heuristically-guided decision-theoretic means, and aggregates them to compute state values. The set of basis functions is normally much smaller than the set of reachable states, thus giving our planner a large reduction in memory requirements as well as in the number of parameters to be learned.

We demonstrate the practicality of our framework by comparing it to the top IPPC-04, 06 and 08 performers and other state-of-the-art planners, on challenging problems from these competitions. RETRASE demonstrates orders of magnitude better scalability than the best optimal planners, and frequently finds significantly better policies than the state-of-the-art approximate solvers. The success of RETRASE

prompts several promising ideas for future work, outlined at the end of the paper.

2 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 executing 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*:

$$V^*(s) = 0 \text{ if } s \in \mathcal{G}, \text{otherwise} \quad (1)$$

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

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. Heuristically-guided methods, e.g. RTDP, explore the state space in a series of *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' [2].

3 ReTrASE

On a high level, RETRASE explores the state space in the same manner as RTDP, but, instead of performing Bellman backups on states themselves, backups are performed over *properties* of the visited states. For each property, modified RTDP learns a weight that reflects the quality of the plans enabled by that property. A state's value may then be computed by aggregating the weights of all its properties. Conceptually, there are three kinds of states at runtime: ones that have been deemed dead ends, ones for which some properties are known, and ones not yet assigned to the other two categories. When RETRASE encounters a state s of the third type, it applies a classical planner (e.g., FF [5]) to a determinized version of the domain starting from s. If no classical plan exists, then every probabilistic policy from s has zero probability of reaching the goal, and s is marked as a dead end. If FF finds a plan, however, RETRASE regresses the goal conjunction through the plan to generate a logical formula which is a property holding in s. Learning in the property space supports information transfer between similar states (e.g., all states that share a given property) even before some of these states are visited. Our approach is efficient because fast classical planners can quickly derive these properties, and because the number of properties is typically far smaller than the number of reachable states.

Definitions. We define a *trajectory* to be a sequence $t = s, a_1(o_{j_1}), \ldots, a_n(o_{j_n})$ where s is the trajectory's starting state, and each action $a_k(o_{j_k})$ represents the j_k-th outcome of the probabilistic action a_k. We say that t is a *goal trajectory* if s modified by t's action sequence is a goal state. Further, we define a state *property* to be a conjunction of literals[1]. We say that a state s *possesses* property p if p holds in s. With each property p, we associate a unique *basis function* that has value 1 in s iff s possesses p.We say that a property p (and the corresponding basis function b_p) *enables* a set of trajectories T to the goal if the goal can be reached from any state possessing p by following any of the trajectories in T. [2] A *dead-end* is a state with no trajectory to the goal.

Algorithm Intuition. Consider a trajectory $t_g = s, a_1(o_{j_1}), \ldots, a_n(o_{j_n})$ that ends in a goal state. This is an indication that the sequence of *probabilistic* actions $a_1, \ldots, a_n$ is potentially causally important, since their outcomes $o_{j_1}, \ldots, o_{j_n}$ have positive probability. To discover the causal properties $p_1, \ldots, p_n$ that allow the successful execution of $a_1, \ldots, a_n$, we simply regress sequence t from the goal conjunction. We can now claim that action sequence $a_k, \ldots, a_n$ executed starting from *any* state possessing property p_k will lead us to the goal with positive probability, though the magnitude of the probability is yet unknown. Note that t essentially chooses specific outcomes per action and thus the execution of $a_1, \ldots, a_n$ may not always reach the goal. Nevertheless, all properties that enable any positive-probability trajectory to the goal may be important for our purposes because they act as a basis for further planning. In essence, this step can be thought of as unearthing the relevant causal structure necessary for the planning task at hand.

To obtain goal trajectories all we need is to find plans that reach the goal in the *deterministic* version of the domain (by using a classical planner). Every such plan corresponds to a positive-probability trajectory in the original domain.

We can now define a new probabilistic planning problem over a state space consisting of these properties. In practice, the space of properties is much smaller than the original state space, since only the relevant causal structure is retained[3], giving us large reductions in space requirements. Solving this new problem amounts to learning the *weights* for the properties. The weights will be a quantitative measure of each property's importance. There are many imaginable ways to learn them; in this paper, we try one of such methods — a modified version of RTDP. The use of RTDP enables us to leverage the power of heuristic guidance in exploring the state space.

[1] Our algorithm can easily extend to properties specified using general logical formulas.

[2] assuming that the desired outcome is obtained for each action on the trajectory.

[3] We may approximate this further by putting a bound on the

Algorithm 1 ReTrASE

Input: probabilistic domain D, problem $P = \langle$init. state s_0, goal $G\rangle$, trial length L
declare global map M from basis functions to weights
declare global set DE of dead ends
compute global determinization D_d of D
// Do modified RTDP over the basis functions
for all $i = 1 : \infty$ **do**
  declare state $s \leftarrow s_0$
  declare $numSteps \leftarrow 0$
  while $numSteps < L$ **do**
    declare action $a' \leftarrow \arg\min_a\{ExpActCost(a, s)\}$
    $ModifiedBellmanBackup(a', s)$
    $s \leftarrow$execute action a' in s
    $numSteps \leftarrow numSteps + 1$
  end while
end for

function ExpActCost(action a, state s)
declare array $S_o \leftarrow$ successors of s under a
declare array $P_o \leftarrow$ probs of successors of s under a
return $cost(a) + \sum_i P_o[i]Value(S_o[i])$

function Value(state s)
if $s \in DE$ **then**
  return a large penalty // e.g., 1000000
else if some member f' of M holds in s **then**
  return $\min_{\text{basis functions } f \text{ that hold in } s}\{M[f]\}$
else
  $GetBasisFuncsForS(s)$
  return $Value(s)$
end if

function GetBasisFuncsForS(state s)
declare problem $p' \leftarrow \langle$init. state s, goal $G\rangle$
declare plan $pl \leftarrow$ DeterministicPlanner(D_d, p')
if $pl == none$ **then**
  insert s into DE
else
  declare basis function $f \leftarrow$ goal G
  declare $cost \leftarrow 0$
  for all $i = length(pl)$ through 1 **do**
    declare action $a \leftarrow pl[i]$
    $cost \leftarrow cost + cost(a)$
    $f \leftarrow (f \cup precond(a)) - effect(a)$
    insert $\langle f, cost\rangle$ into M if f isn't in M yet
  end for
end if

function ModifiedBellmanBackup(action a, state s)
for all basis functions f in s that enable a **do**
  $M[f] \leftarrow ExpActCost(a, s)$
end for

The weights reflect the fact that the properties differ in the total expected cost of trajectories they enable, as well as in the total probability of these trajectories. This happens partly because each trajectory considers only one outcome for each of its actions. The sequence of outcomes the given trajectory considers may be quite unlikely. In fact, getting some action outcomes that the trajectory does not consider may prevent the agent from ever getting to the goal. Thus, it may be much "easier" to reach the goal in the presence of some properties than others. Now, given that each state generally has several properties, what is the connection between the state's value and their weights? In general, the relationship is quite complex: under the optimal policy, trajectories enabled by several properties may be possible. Therefore, the exact value of a state is a summation of weights over a subset of the state's properties. However, determining this subset is at least as hard as solving the MDP exactly. Instead, we approximate the state value by the *minimum* weight of all properties that the state possesses. This amounts to saying that the "better" a state's "best" property is, the "better" is the state itself.

Thus, deriving useful state properties and their weights gives us an approximation to the optimal value function. The algorithm's pseudocode is presented in Listing 1.

Theoretical Properties. A natural question about RETRASE is that of convergence. As it turns out, there exist problems on which RETRASE does not converge. We stress, however, that the lack of theoretical guarantees is not indicative of a planner's practical success or failure. Indeed, the experimental results show that RETRASE performs quite outstandingly on many of the planning community's benchmark problems.

4 EXPERIMENTAL RESULTS

Our experiments explore two important aspects of RETRASE – (1) quality of solutions in complex domains and (2) scalability. We ran RETRASE on six probabilistically interesting hard problem sets — Triangle Tire World (TTW) from IPPC-06 and -08, Drive from IPPC-06, Exploding Blocks World (EBW) from IPPC-06 and -08, and Elevators from IPPC-06. The experiments were conducted under the restrictions resembling those of IPPC: for each problem, RETRASE had a maximum of 40 minutes for training and then had 30 attempts to solve each problem. The parameter we measured was success rate (the percentage of 30 trials in which RETRASE managed to solve the given problem) — the factor that decides the winner in IPPC. The runs were performed on a 2.8 GHz Intel Xeon processor with 2GB of RAM.

While analyzing the results, it is important to be aware that our RETRASE implementation is not optimized. Consequently, RETRASE's efficiency is likely even better than indicated by the experiments.

On TTW-06 and -08, RETRASE achieved the perfect 100% success rate across all the problems. For TTW-06, this result is unmatched by the IPPC participants, while on TTW-08 one other planner, HMDPP, achieved the same result. On Elevators, RETRASE's performance was rather average, as it could not solve many of the problems. This is probably due

number of properties we are willing to handle in this step.

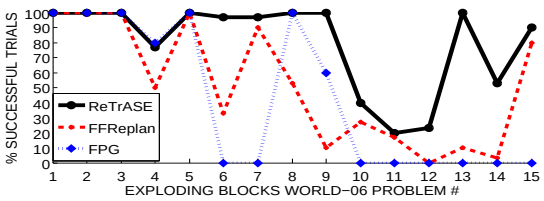

Figure 1: RETRASE dominates on Exploding Blocks World-06.

to the poor set of basis functions it managed to extract for this domain. On Drive, RETRASE's average success rate was at par with the best of IPPC participants (FFReplan and FPG). RETRASE distinguished itself particularly on EBW-08 and EBW-06. On both domains its performance was better than that of all the IPPC competitors. The advantage is especially impressive on EBW-06 because it grows with the complexity of the problem (Figure 1). Space constraints prevent us from presenting the results from other domains in graphical form.

We note also that RETRASE's scalability is far better than that of heuristically guided optimal or suboptimal planners. For instance, LRTDP with FF heuristic ran out of memory on problems 8, 9, and 10 of TTW-08, whereas RETRASE solved them easily.

5 DISCUSSION

RETRASE's potential is clearly indicated by the experimental results, so we would like to extend RETRASE's ideas to other areas of automated planning. One field that would benefit from RETRASE's scalability is POMDP solvers. To be capable of handling POMDPs, RETRASE will need to use a conformant deterministic planner to come up with basis functions. The state of the art in conformant planning will largely determine RETRASE's success as a POMDP solver.

The lack of theoretical guarantees of RETRASE can be perceived by some to be its weakness. To address this concern, we can modify RETRASE's learning mechanism. Space constraints prevent us from specifying the details, but the main idea involves reducing a given MDP to a maximum independent set problem over a "conflict graph" whose vertices are basis functions with associated weights. The independent set problem can then be solved using one of the many theoretically strong approximators for it.

6 RELATED WORK

Besides basis function approximation (discussed in Section 1) other flavors of dimensionality reduction include PCA and algebraic and binary decision diagram (ADD/BDD). In practice algorithms that use ADD/BDD do not scale to large problems. APRICODD [10] is an exception, but it is not clear whether it is competitive with today's top methods. In continuous state spaces, some researchers have applied non-linear techniques like exponential-PCA and NCA for dimensionality reduction [8].

Most basis function based techniques are not applied in nominal domains. A notable exception is FPG [3] but RETRASE outperforms it consistently on several domains.

RETRASE is described in more detail [6]. It is also related in spirit to the probabilistic planners that use determinized domains for probabilistic planning, e.g. FFReplan [11] and FFHop [12].

The idea of using determinization followed by regression has parallels to some research on relational MDPs, e.g., [4; 9].

7 CONCLUSION

Exploiting problem structure, heuristics, and various approximations all seem to be essential components of highly scalable successful probabilistic planners. However, most existing solvers use only a proper subset of them. Our work bridges this gap by introducing RETRASE, an algorithm that combines the power of these approaches. It extracts problem structure in a domain-independent way and learns the parameters in the reduced parameter space under the guidance of a heuristic. We empirically demonstrate that RETRASE outmatches state-of-the-art planners on hard problems from several IPPC competitions and scales drastically better than existing heuristically guided planners. Developing RETRASE's underlying ideas further, we are planning to extend them to POMDPs and add theoretical guarantees on the solution quality by modifying the parameter learning mechanism.

References

[1] A. Barto, S. Bradtke, and S. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72:81–138, 1995.

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

[3] O. Buffet and D. Aberdeen. The factored policy gradient planner (ipc-06 version). In *Fifth International Planning Competition at ICAPS'06*, 2006.

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

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

[7] I. Little and S. Thiebaux. Probabilistic planning vs. replanning. In *ICAPS Workshop on IPC: Past, Present and Future*, 2007.

[8] N. Roy and G. Gordon. Exponential family PCA for belief compression in POMDPs. In *NIPS'02*, pages 1043–1049. MIT Press, 2003.

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

[10] R. St-Aubin, J. Hoey, and C. Boutilier. APRICODD: Approximate policy construction using decision diagrams. In *NIPS'00*, 2000.

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

[12] S. Yoon, A. Fern, S. Kambhampati, and Robert Givan. Probabilistic planning via determinization in hindsight. In *AAAI'08*, 2008.