Factored MCTS for Large Scale Stochastic Planning

Hao Cui and **Roni Khardon**
Department of Computer Science
Tufts University
Medford, MA 02155, USA

Alan Fern and **Prasad Tadepalli**
School of Electrical Engineering and Computer Science
Oregon State University
Corvallis, OR 97331, USA

Abstract

This paper investigates stochastic planning problems with large factored state and action spaces. We show that even with moderate increase in the size of existing challenge problems, the performance of state of the art algorithms deteriorates rapidly, making them ineffective. To address this problem we propose a family of simple but scalable online planning algorithms that combine sampling, as in Monte Carlo tree search, with "aggregation", where the aggregation approximates a distribution over random variables by the product of their marginals. The algorithms are correct under some rather strong technical conditions and can serve as an unsound but effective heuristic when the conditions do not hold. An extensive experimental evaluation demonstrates that the new algorithms provide significant improvement over the state of the art when solving large problems in a number of challenge benchmark domains.

Introduction

Stochastic planning research has made significant progress in recent years in terms of algorithms developed, the type of problems solved, and their scale. The current best approaches use the idea of online planning to optimize the action choice in the current state of the Markov decision process at each time step. Indeed, the two strongest systems in the last two probabilistic planning competitions are based on the online algorithms RTDP (real time dynamic programming) and UCT (a variant of MCTS, Monte Carlo tree search) (Kolobov et al. 2012; Keller and Eyerich 2012). Symbolic dynamic programming, the other main approach, aims to achieve scalability via structured forms of dynamic programming (with or without online planning) (Hoey et al. 1999; Feng, Hansen, and Zilberstein 2002).

In this paper, however, we argue that the focus on optimization algorithms in recent research is at odds with scalability. If we use exactly the same type of challenge problems as used in existing work, but increase problem size even moderately, then the performance of the algorithms deteriorates rapidly, making them ineffective. Thus, despite the great advances and algorithmic insights, the largest solvable problems are still rather small. We therefore propose that it is useful to follow a complementary approach focusing directly on large problems, and developing sub-optimal but scalable algorithms. When full optimization is approachable such approximate approaches can serve as a useful baseline. In many cases, however, approximations can provide scalable quality solutions for online planning.

The paper makes two contributions motivated by this discussion. First, we present an experimental study on current challenge problems and moderately larger problems exposing the above mentioned phenomenon. Second, we develop novel online planning algorithms that combine sampling (as in Monte Carlo tree search) with approximation through symbolic, or algebraic, "aggregation". In particular, we extend two algorithms, Rollout (Tesauro and Galperin 1996; Bertsekas and Tsitsiklis 1996) and MCTS (Browne et al. 2012) to incorporate such approximations. The main novelty in our algorithms is in their use of the symbolic form of a planning domain model to enable approximate but scalable simulation. To the best of our knowledge, this is the first work to integrate models and sampling in this manner.

Our symbolic approximation is closely related to approximating distributions by their product of marginals, and is therefore related to other recent efforts in approximate probabilistic inference. Like that work, our algorithms are correct under some rather strong technical conditions, which we establish. More importantly, however, they are applicable even when these conditions do not hold. As we demonstrate in the experiments, the new algorithms provide significant improvement over the state of the art when problem size is increased in a number of benchmark domains. These algorithms can therefore be seen as baselines for large scale stochastic planning problems. As discussed in the paper, despite these facts, our algorithms are heuristic and they do not always deliver the same quality performance. Identifying the scope and applicability of the aggregate methods, and potential improvements, are important problems for future work.

Background

A Markov Decision Process (MDP) (Puterman 1994) is specified by $\{\mathbb{S}, \mathbb{A}, T, R, \gamma\}$, where $\mathbb{S}$ is a finite state space, $\mathbb{A}$ is a finite action space, $T(s, a, s') = p(s'|s, a)$ defines the transition probabilities, $R : \mathbb{S} \times \mathbb{A} \to \mathcal{R}$ is the immediate reward of taking action a in state s, and γ is the discount factor. A policy $\pi : \mathbb{S} \to \mathbb{A}$ is a mapping from a state s to an action

a, indicating which action to choose at each state. Given a policy π, the value function $V^\pi : S \to \mathcal{R}$ is the expected discounted total reward $E[\sum_i \gamma^i R(s_i, \pi(s_i)) \mid \pi]$, where s_i is the i^{th} state visited by following policy π and s_0 is the initial state. The action-value function $Q^\pi : \mathbb{S} \times \mathbb{A} \to \mathcal{R}$ is the expected discounted total reward when taking action a at state s and following π thereafter.

MDPs with large state and action spaces are typically represented in factored form (Boutilier, Dean, and Hanks 1995; Raghavan et al. 2012). The state space $\mathbb{S}$ is specified by a set of binary variables $\{x_1, \dots x_l\}$ so that $|\mathbb{S}| = 2^l$. Similarly, $\mathbb{A}$ is specified by a set of binary variables $\{y_1, \dots y_m\}$ so that $|\mathbb{A}| = 2^m$. To simplify the presentation, we use $\{b_1, \dots, b_l, b_{l+1}, \dots, b_{l+m}\}$ to denote the union of the state variables (the first l b_j's) and the action variables (the remaining b_j's).

Classical MDP optimization algorithms require enumeration over states and actions making them impractical for large problems. Symbolic versions of those algorithms have been developed for factored state spaces (Hoey et al. 1999; Boutilier, Dearden, and Goldszmidt 1995) and factored action spaces (Raghavan et al. 2012; 2013). However, scalability is still problematic due to the inherently large description length of value functions and policies.

Online planning avoids this complexity by focusing on choosing a high-quality action for the current state and repeatedly applying this procedure to states visited. The **Rollout Algorithm** (Tesauro and Galperin 1996; Bertsekas and Tsitsiklis 1996) is perhaps the simplest such procedure which performs one step lookahead from the current state. In particular, given a state s and baseline policy π the rollout algorithm calculates an estimate of $Q^\pi(s, a)$ for each action a and then picks the a maximizing this value. To estimate $Q^\pi(s, a)$ simply use the MDP model (or a simulator) to sample s' from $p(s'|s, a)$ and then apply π for h steps where h is the desired horizon. Repeating this k times and averaging the total discounted reward we get an estimate for $Q^\pi(s, a)$.

Monte Carlo Tree Search (MCTS) takes this idea one step further by building a search tree over states and action choices where at the leaves we apply the rollout procedure to estimate the value of the corresponding state. The UCT Algorithm (Kocsis and Szepesvári 2006) is a variant of MCTS where action selection at internal nodes of the tree is done by calculating an upper bound on action values and picking the action that maximizes the upper bound. Following theoretical analysis (Kocsis and Szepesvári 2006) and empirical success (Gelly and Silver 2008) UCT has attracted considerable attention and extensions (Browne et al. 2012). In particular, the PROST system (Keller and Eyerich 2012; Keller and Helmert 2013) which extends and adapts UCT with ideas specific to stochastic planning domains, was the winner in the last two international probabilistic planning competitions. The ϵ-greedy action choice is an alternative to UCT which is known to perform better in some cases (Tolpin and Shimony 2012).

As illustrated in our experiments, when problem size increases MCTS search is often not very informative, resulting in significantly reduced performance. This is especially severe for large action spaces where the number of samples per action at the root node is very small. In such cases, the value estimates are clearly not reliable. In addition, in extreme cases there is just one sample per action at the root and the tree is flat, i.e., equivalent to rollout. Ontañón (2013) makes a similar point in the context of real-time strategy games and develops an algorithm that alternates between optimizing individual action variables and the global assignment. The algorithm we develop next takes a different route.

Algebraic Rollout Algorithm (ARollout)

This section presents our main algorithmic tool that generates approximate simulation through an algebraic representation of the transition function. Consider the rollout process when estimating $Q^\pi(s, a)$. The main idea in our algorithm is to efficiently calculate an approximation of the distribution on states that would be visited in this process. By estimating rollout values based on this distribution we can effectively take into account a large number of "potential trajectories", much larger than the number of real trajectories considered by standard rollouts. This introduces an approximation or bias but can help reduce variance in the estimates.

More specifically, given $p(s_t)$, the true distribution over states at time t, one can calculate R_t the expected reward at time t and the distribution over the next state $p(s_{t+1})$. However, these distributions are complex and the calculations are not feasible. Therefore, in this paper we use factored distributions where all state variables are assumed to be independent $p(s = b) = \prod_j p(b_j)$. This is reminiscent of the factored frontier algorithm (Murphy and Weiss 2001) for state estimation in the filtering context and of approximations used in variational inference.

A crucial point for our algorithm is the fact that we can use the symbolic planning model to enable this computation, by translating the MDP model into algebraic expressions. To facilitate the presentation we consider domain models given in the RDDL language (Sanner 2010) which is the current standard for specifying stochastic planning domains.

Example RDDL Domain: We use the following simple RDDL domain to illustrate the constructions. The state description includes 4 variables `bit(?b)` where the initial state is 0100 (that is `bit(b1)=0, bit(b2)=1`, etc). The state also includes the variables `automatic(?b)` which are initially 0000. At each step we can take at most two of the four possible actions, denoted respectively by `set(?b)`. If a bit is already 1, then it remains 1. Otherwise, if we take the action to set it or `automatic(?b)` holds, then with probability 0.7 it turns into 1. Otherwise it remains 0. In addition, `automatic(?b)` is randomly set in each step by some exogenous process. The reward is a conditional additive function over state variables. This is captured in RDDL by:

```
bit'(?b) =
 if (~bit(?b) ^ (set(?b) | automatic(?b)))
     then Bernoulli(0.7)
     else if (bit(?b))
                then KronDelta(true)
                else KronDelta(false)
automatic'(?b) = Bernoulli(0.3)
reward =
  if (bit(b2) | bit(b3))
     then bit(b1) + 5*bit(b2) + 2*bit(b3)
     else  bit(b1) + bit(b4)
```

Problem	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
S	8	10	12	22	25	28	42	58	63	68	73	78	100	106	112	138	145	152	182	190
A	3	3	3	6	6	6	9	12	12	12	12	12	15	15	15	18	18	18	21	21
S	10	10	20	20	30	30	40	40	50	50	80	100	120	130	150	160	180	190	200	200
A	10	10	20	20	30	30	40	40	50	50	80	100	120	130	150	160	180	190	200	200
S	18	18	32	32	50	72	72	72	98	98	98	112	128	144	162	180	200	220	242	264
A	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
S	30	32	44	46	50	50	60	60	73	73	72	78	84	81	84	87	88	96	104	112
A	4	4	4	4	4	4	4	4	4	4	9	9	9	12	12	12	16	16	16	16

Table 1: Number of state/action variables in test problems (top to bottom): one-dir-elevators, sysadmin, crossing traffic, traffic.

Preprocessing: translating a RDDL model into an algebraic expression: Transitions in RDDL are given by expressions of the form

$$b' = [\text{if } C_1 \text{ then } E_1 \text{ else if } C_2 \text{ then } E_2 \dots \text{ else } E_k]$$

where b' is the value of bit b at the next time step, C_i is a Boolean expression in terms of $\{b_j\}$ (at the current time step) and E_i is either a Boolean expression or a Bernoulli distribution constant as in the example above. For the translation we support a subset of RDDL where the Boolean portion allows for logical operators $\wedge$, $\vee$ and $\neg$ and quantifiers $\forall$ and $\exists$ but not summations and comparisons (e.g., `(sum_{?b: bit}[bit(?b)]>5)`). Extensions to the full set of RDDL expressions are possible in a number of ways but are left for future work. We first translate the expression for `b'` into a disjoint-sum form

$$(C_1 * E_1) + ((\overline{C_1} \wedge C_2) * E_2) + \ldots + ((\ldots) * E_k)$$

where the logical terms in the sum are mutually exclusive so that the value is equivalent to the if-then-else expression above. To guarantee correctness of our algorithm we need to further simplify each of the terms (e.g., the term $((\overline{C_1} \wedge C_2) * E_2)$, where C_1 and C_2 are arbitrary Boolean expressions) into a disjoint-sum form where internal terms are mutually exclusive. Putting these simplified terms together we get a disjoint sum representation of the entire expression.

Example: In our example, the first step yields (we use `b, bk, sk, ak` instead of `bit(?b), bit(bk), set(bk), and automatic(bk)`):

```
b' = (    (~b ^ (s | a)) * 0.7) +
     ( (~(~b ^ (s | a)) ^ b) * 1 ) +
     ( (~(~b ^ (s | a)) ^ (~b)) * 0).
```

The disjoint-sum representation of the first disjunct is

```
(~b ^ (s | a) * 0.7) =
  ((~b ^ s) | (~b ^ a)) * 0.7 =
  (a ^ ~b ^ s) * 0.7 + (~a ^ ~b ^ s) * 0.7 +
     (a ^ ~b ^ ~s) * 0.7
```

and the second and third terms simplify to `b` and `0` respectively. The final expression is

```
b'= (a ^ ~b ^ s) * 0.7 + (~a ^ ~b ^ s) * 0.7
      +  (a ^ ~b ^ ~s) * 0.7 + b * 1.
```

We then replace all remaining logical operators where $\wedge$ is replaced by $*$ and $\neg x$ is replaced by $1 - x$.

RDDL expressions for the reward R are similar to the ones for transitions but the E_i portions can include numerical expressions over the variables $\{b_j\}$. The expressions are translated in exactly the same manner, except that in the last step numerical operations need not be substituted.

Example: In our example, the initial portion of the reward `if (b2|b3) then b1+5*b2+2*b3` is translated into

```
((b2 ^ b3) | (b2 ^ ~b3) | (~b2 ^ b3))
        * (b1 + 5 * b2 + 2 * b3)
```

which in turn can be evaluated as

```
(b2 ^ b3)  * (b1 + 5 * 1 + 2 * 1) +
(b2 ^ ~b3) * (b1 + 5 * 1 + 2 * 0) +
(~b2 ^ b3) * (b1 + 5 * 0 + 2 * 1).
```

Aggregate simulation: For each state or action variable b_j in $\{b_1, \dots b_{l+m}\}$ we denote by $b_{j,t}$ the value of this variable at time step t. The current approximate distribution $p(s_t)$ is captured by the product over $p(b_{j,t})$, $j \leq l$ where at $t = 0$ the distribution is concentrated at the start state. Exact simulation to the distribution over next state variables is not always feasible. Instead, we first estimate the marginal probabilities over action variables $p(b_{j,t})$ $j > l$ as explained in the next paragraph. We then substitute $p(b_{j,t})$ for state and action variables $b_{j,t}$ in the expressions calculated in the preprocessing phase of the algorithm. This yields an approximate value of $p(b_{j,(t+1)})$ for all state variables. Finally, repeating this process for all t we get an approximate version of the rollout procedure, performed over distributions.

The estimation of marginal probabilities over action variables $p(b_{j,t})$ depends on the nature of π and on the existence of constraints over action variables (in the example at most 2 out of the 4 action bits are set to 1 at every step). We consider here 3 types of policies for rollout. First, if the domain has no action constraints and π is the random policy then the distribution is simply the uniform distribution over action variables. Second, if the policy is constant (i.e., we always take the same action) then again the setting of $p(b_{j,t})$ is easy. Note that, in both of these cases action variables are independent of the state. This is used below in the analysis.

Third, consider any concrete π, noting that when there are action constraints, even the random policy can induce state dependence and correlation between action variables. We propose two solutions that differ in the type of aggregation used for the approximation. In the variant which does aggregation over states only (denoted by S in the experiments below) we pick a concrete state s according to the distribution over state variables $\prod_j p(b_{j,t})$. We then use the concrete action $\pi(s)$ as the setting for the action variables which are

substituted into the preprocessed expressions. For aggregation over states and actions (denoted by SA in the experiments below) we pick n random states $s_1, \ldots, s_n$ according to the distribution over state variables $\prod_j p(b_{j,t})$ where n is a parameter of the algorithm. Let $\{b^{(i)}\} = \{\pi(s_i)\}$ be the corresponding actions according to π in their binary representations. We then set our estimate to be the bitwise average of these actions, that is, $p(b_{j,t}) = \frac{1}{n}\sum_i b_j^{(i)}$. Clearly aggregation over states only is suboptimal because it picks to use the same action in all states when going to s_{t+1}. But it is semantically meaningful because we are progressing the entire distribution through this action. On the other hand, aggregation over states and actions is harder to justify because it uses "fractional actions". Nonetheless, as our experiments show it is quite effective in practice.

Action Selection: The description above specifies how to perform the simulation part of the rollout algorithm. To pick an action in s we still need to estimate $Q^\pi(s,a)$ for all a. That is, for each a we simulate one step of $p(s'|s,a)$ to pick s' and then perform ARollout from s'. Repeating this multiple times and averaging we get our estimate of $Q^\pi(s,a)$. Since our algorithm performs on-line planning, the number of simulations is determined dynamically from the time per step allocated to the system. For action choice, when there are unvisited actions, we select uniformly among these actions. If all the actions have been tried at least once, we use the ϵ-greedy choice (that is we select the greedy action with probability $1-\epsilon$ and a random action with probability ϵ).

In summary the ARollout algorithm performs approximate aggregate rollout simulations from the desired state in order to estimate $Q^\pi(s,a)$. Each such simulation is not exact but it captures, in aggregate, a large number of potential runs in one sample. Therefore, we may expect that if the time per step is not limited then the simple rollout which is not biased will have more accurate estimates and should be preferred. On the other hand if the time per step is limited so that the number of runs per action is small then the aggregate simulation might be better even though it is approximate.

Correctness: Correctness of the computations follows for the same reasons that model counting is correct for d-DNNF (Darwiche and Marquis 2002). In particular, from the discussion above we see that the transition function is a sum of products in terms of state and action variables where all summands are mutually exclusive and the products are of the form $(\prod_{j_1} b_{j_1})(\prod_{j_2}(1-b_{j_2}))$ where the sets of j_1, j_2 are disjoint. First note that, because summands are mutually exclusive, replacing $\vee$ with $+$ is correct. In addition, when all the random variables in $\{b_j\}$ that are used in the expression are mutually independent we have that:
$p([(\prod_{j_1} b_{j_1})(\prod_{j_2}(1\text{-}b_{j_2}))]{=}1) = (\prod p(b_{j_1}))(\prod(1\text{-}p(b_{j_2})))$
implying that substitution of marginals correctly calculates the marginal probability of each variable. Similar arguments hold for the expression for reward. To guarantee that all the input $\{b_j\}$ are independent, state and action variables must be independent and next state variables must be conditionally independent given state and action, i.e., we cannot allow synchronic arcs in the DBN. This implies (note that condition 2 holds for type 1 and 2 discussed above):

Proposition 1. *If (1) $p(s_t)$ is a product distribution, (2) action choice is a product distribution over action variables and is independent of state variables, (3) the set of variables $\{b_{j,(t+1)}\}$ is conditionally independent given s_t, a_t, (4) the numerical expression for R satisfies $E[R(b_1 \ldots b_{l+m})] = R(E[b_1] \ldots E[b_{l+m}])$ then the the algorithm that aggregates over states correctly calculates marginal probabilities for $p(s_{t+1})$ and expectation for $R(s_t, a_t)$.*

Implementation: The intermediate step of turning disjuncts into disjoint-sum form can be expensive and our implementation avoids it instead performing an algebraic substitution directly on the original RDDL expressions. In this translation we replace internal disjunctions $\alpha \vee \beta$ (where α, β are not mutually exclusive) by $1-((1-\alpha)*(1-\beta))$. This introduces an additional approximation but simplifies the translation process and makes sure that it is efficient.

Algebraic MCTS Algorithm (AMCTS)

Given ARollout, extensions of MCTS algorithms to use algebraic simulation are natural. For example, one can take a state level MCTS algorithm and simply replace the rollout at the leaves with algebraic rollouts. Our experiments report on a different variant which performed better in preliminary experiments, and which works as follows: (1) Tree nodes include "aggregate states" (distributions) exactly as in the rollout representation. (2) Actions in tree nodes are concrete actions exactly as in MCTS and their transitions are calculated using the algebraic expressions. (3) Action selection at tree nodes is done as in ARollout, choosing each action at least once at first, and then using ϵ-greedy choice.

Results

We ran experiments on the Tufts UIT research cluster (each node includes Intel Xeon X5675@ 3GHz CPU, and 24GB memory). We used five domains for the evaluation. Four domains are from IPPC2011, the elevators domain (where people randomly arrive at each floor and go to either top or bottom floor), the sysadmin domain (where failures of computers depend on their neighbors and one can reboot a number of computers at each time step), the crossing traffic domain (where a robot tries to get to the other side of a river with randomly appearing flowing obstacles), and the traffic domain (where one controls traffic lights to enable traffic flow). The fifth domain, from the RDDL distribution[1] is a variant of the elevators domain, denoted one-dir-elevators below, (where the only destination is the top floor). The IPPC provided 10 instances for each domain. To test scalability we added 10 more instances for each domain. Note that we do not change the nature of the domain dynamics but simply add objects, their fluents, and decision nodes. We similarly generated 20 instances for one-dir-elevators domain.

We report quantitative results for four of the domains: one-dir-elevators, sysadmin, crossing traffic and traffic, which illustrate success of our algorithms. The IPPC2011 elevators domain, where our algorithms do not perform well, is discussed after the quantitative results.

[1] http://concurrent-value-iteration.googlecode.com/svn-history/r133/trunk/rddl/elevators_mdp.rddl

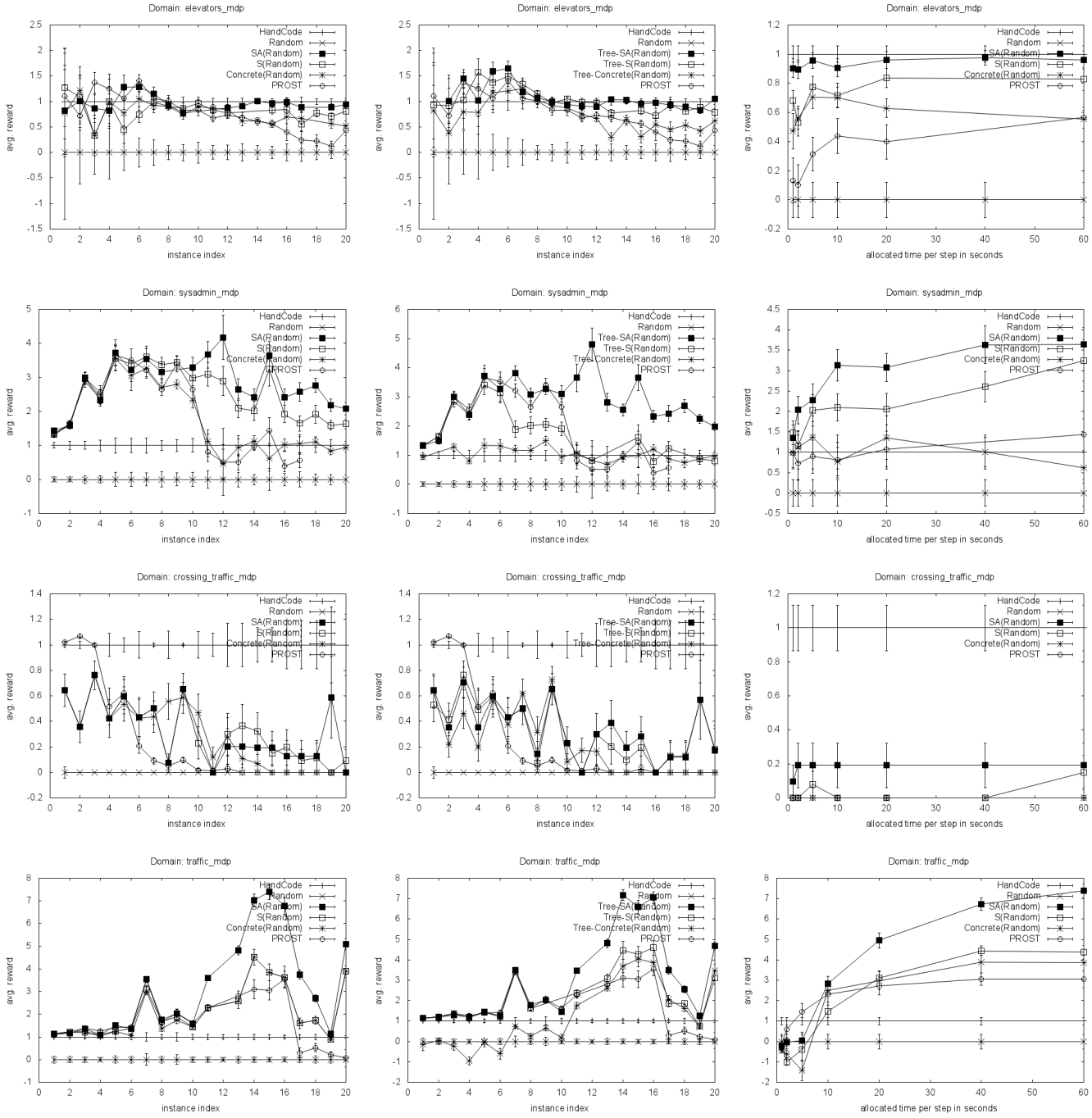

Figure 1: Data from experimental evaluation. Rows correspond to domains ordered as one-dir-elevators, sysadmin, crossing traffic and traffic from top to bottom. Left: comparing rollout algorithms. Middle: comparing MCTS variants of the rollout algorithms. Right: comparing rollout algorithms as a function of time per step on one concrete problem (problem 15) in each domain.

The number of state and action variables in the 20 instances of the four problems is listed in Table 1 where the new instances are listed as problems 11-20. In the plots given in Figure 1 each point represents the average and standard deviation in 20 runs on the same problem.

For the evaluation, we manually wrote a policy for each domain. The idea was not to spend too much time to optimize the policy but simply to get a baseline for comparison. A second baseline is given by the random policy. Any reasonable planning algorithm should be better than the random policy and, depending on the amount of human effort in developing the specialized policy, get close to or even be better than the specialized policy. In addition, we normalize all results using a linear transformation so that the random policy stands for average reward of 0 and the hand-coded policy stands for reward of 1. This puts all problems on the same scale, and thus facilitates comparisons across problems. A third baseline is given by PROST (Keller and Eyerich 2012), which implements a variant of UCT on concrete states.

We evaluate both rollout and MCTS algorithms. The first group includes algorithms based on rollout: Concrete(Random) is the standard algorithm, S(Random) is ARollout with state aggregation only, and SA(Random) is ARollout with state and action aggregation. The second group includes the AMCTS extension where as above we vary the rollout at the leaves. Tree-Concrete(Random) samples a concrete state from the product distribution at the leaf and uses standard rollout, Tree-S(Random) uses state aggregation only, and Tree-SA(Random) uses state and action aggregation. In all the algorithms the rollout policy is the random policy which respects action constraints in the domain. Note that this violates the assumptions in the analysis due to the action constraints.

The settings of the algorithms are as follows. For PROST we use the IPPC2011 setting except that the allocated time per step is explicitly set. For the most difficult sysadmin problems, PROST failed to complete preprocessing after 5 hours and no results are reported. For our algorithms, as mentioned above, the number of samples per action a for the estimate of $Q^\pi(s, a)$ is determined dynamically. The parameter n that controls the number of action samples when aggregating over actions is set to $\min\{10, 0.6 * |\mathbb{A}|\}$. This reflects the fact that when the action space is small we do not need a large number of samples for the estimates. The simulation depth in ARollout and the depth of the MCTS tree are set to half of the horizon used in evaluation time. We use $\epsilon = 0.5$ for the ϵ-greedy action choice in all algorithms. In our main comparison, all the algorithms including PROST are allowed 60 seconds per step. Our second comparison shows how the results change as a function of the time per step for one specific problem in each domain.

The left column in Figure 1 shows the comparison among the variants of the simple rollout algorithms and PROST. For all the domains except crossing traffic SA(Random) is better than S(Random) and neither algorithm dominates in crossing traffic. A possible explanation is the fact that we did not increase the action space in this domain (all problems have 4 action variables). For all the four domains, S(Random) is better than Concrete(Random). This illustrates that, even though our assumptions are not valid, in practice aggregation does improve the performance over simple rollouts. We can also see that the algorithm without aggregation, Concrete(Random), performs significantly worse on larger problems whereas SA(Random) appears to maintain its performance across problem sizes. Finally, the performance of PROST degrades significantly for the large problems and SA always dominates its performance.

The middle column in Figure 1 shows a similar comparison over the MCTS algorithms where the relations between the algorithm remains qualitatively similar. Comparing the levels of performance between the two columns we see that the tree search does not improve performance significantly over simple rollouts, except for some smaller problems. This agrees with intuition because for problems with large action spaces the tree algorithm will spend most of its time sampling unique actions at the top level and may not have much chance to go beyond simple rollouts.

The right column in Figure 1 shows how the performance of the algorithms changes as a function of the time per step for problem 15 from each domain. As the plots demonstrate, although there is some variation, the relations observed between algorithms are stable across the time range.

Despite these successes, our algorithms do not perform well on the IPPC2011 elevators domain, which is the only domain to date where we have found them to fail. In this domain the reward function includes a large penalty for people in the elevator who move away from their top or bottom floor destination, expressed as two terms "person-going-up-in-elevator and elevator-going-down" and "person-going-down-in-elevator and elevator-going-up". Now, the fact that elevator-going-down and elevator-going-up are mutually exclusive is lost in the product distribution of the aggregate states, with the result that the penalty is applied whenever there is a person in the elevator. In turn this yields a bad approximation of values in this domain. This is clearly a case where the bias from our approximation is too strong. We note that the same happens for simple rollout of the random policy because the penalty from random actions violating these conditions dominates value estimates. Therefore in this case deep trees are required even with simple rollouts.

Discussion

The paper presents a new approach for approximate aggregate simulation through algebraic representation of the transition function. Empirical evaluation demonstrated that this works well across a number of challenging benchmarks even when our assumptions do not hold. This yields new baselines for large scale stochastic planning problems.

On the other hand, our algorithms did not perform well in the elevators domain where the interaction between the reward and the correlated state variables was not approximated well through the product distributions. Analyzing when exactly the algebraic method will succeed and potential improvements is an important direction for future work.

A second limitation of our algorithms is that, just like basic rollouts, they must evaluate $Q^\pi(s, a)$ for each a explicitly. Efficient estimation of $Q^\pi(s, a)$ without enumeration of actions is necessary for scaling to very large action spaces.

Acknowledgments

This work was partly supported by NSF under grants IIS-0964457 and IIS-0964705. We are grateful to Scott Sanner for use of the RDDL software, to Thomas Keller for use of the PROST system, and to both for answering numerous questions about the setup and use of these systems.

References

Bertsekas, D. P., and Tsitsiklis, J. N. 1996. *Neuro-Dynamic Programming*. Athena Scientific.

Boutilier, C.; Dean, T.; and Hanks, S. 1995. Planning under uncertainty: Structural assumptions and computational leverage. In *Proceedings of the Second European Workshop on Planning*, 157–171.

Boutilier, C.; Dearden, R.; and Goldszmidt, M. 1995. Exploiting structure in policy construction. In *International Joint Conference on Artificial Intelligence*.

Browne, C. B.; Powley, E.; Whitehouse, D.; Lucas, S. M.; Cowling, P. I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2012. A survey of Monte Carlo tree search methods. *Computational Intelligence and AI in Games, IEEE Transactions on* 4(1):1–43.

Darwiche, A., and Marquis, P. 2002. A knowledge compilation map. *Journal of Artificial Intelligence Research* 17:229–264.

Feng, Z.; Hansen, E. A.; and Zilberstein, S. 2002. Symbolic generalization for on-line planning. In *Proceedings of the Nineteenth conference on Uncertainty in Artificial Intelligence*.

Gelly, S., and Silver, D. 2008. Achieving master level play in 9 x 9 computer go. In *AAAI Conference on Artificial Intelligence*, volume 8, 1537–1540.

Hoey, J.; St-Aubin, R.; Hu, A.; and Boutilier, C. 1999. SPUDD: Stochastic planning using decision diagrams. In *Proceedings of the Fifteenth conference on Uncertainty in Artificial Intelligence*.

Keller, T., and Eyerich, P. 2012. PROST: probabilistic planning based on UCT. In *International Conference on Automated Planning and Scheduling*.

Keller, T., and Helmert, M. 2013. Trial-based heuristic tree search for finite horizon MDPs. In *International Conference on Automated Planning and Scheduling*.

Kocsis, L., and Szepesvári, C. 2006. Bandit based Monte-Carlo planning. In *Proceedings of the European Conference on Machine Learning*. Springer. 282–293.

Kolobov, A.; Dai, P.; Mausam, M.; and Weld, D. S. 2012. Reverse iterative deepening for finite-horizon MDPs with large branching factors. In *Twenty-Second International Conference on Automated Planning and Scheduling*.

Murphy, K., and Weiss, Y. 2001. The factored frontier algorithm for approximate inference in DBNs. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, 378–385. Morgan Kaufmann Publishers Inc.

Ontañón, S. O. 2013. The combinatorial multi-armed bandit problem and its application to real-time strategy games. In *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*.

Puterman, M. L. 1994. *Markov decision processes: Discrete stochastic dynamic programming*. Wiley.

Raghavan, A.; Joshi, S.; Fern, A.; Tadepalli, P.; and Khardon, R. 2012. Planning in factored action spaces with symbolic dynamic programming. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*.

Raghavan, A.; Khardon, R.; Fern, A.; and Tadepalli, P. 2013. Symbolic opportunistic policy iteration for factored-action MDPs. In *Advances in Neural Information Processing Systems*.

Sanner, S. 2010. Relational dynamic influence diagram language (rddl): Language description. *Unpublished Manuscript. Australian National University*.

Tesauro, G., and Galperin, G. 1996. On-line policy improvement using monte-carlo search. In *International Conference on Neural Information Processing Systems*.

Tolpin, D., and Shimony, S. E. 2012. MCTS based on simple regret. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*, 570–576.