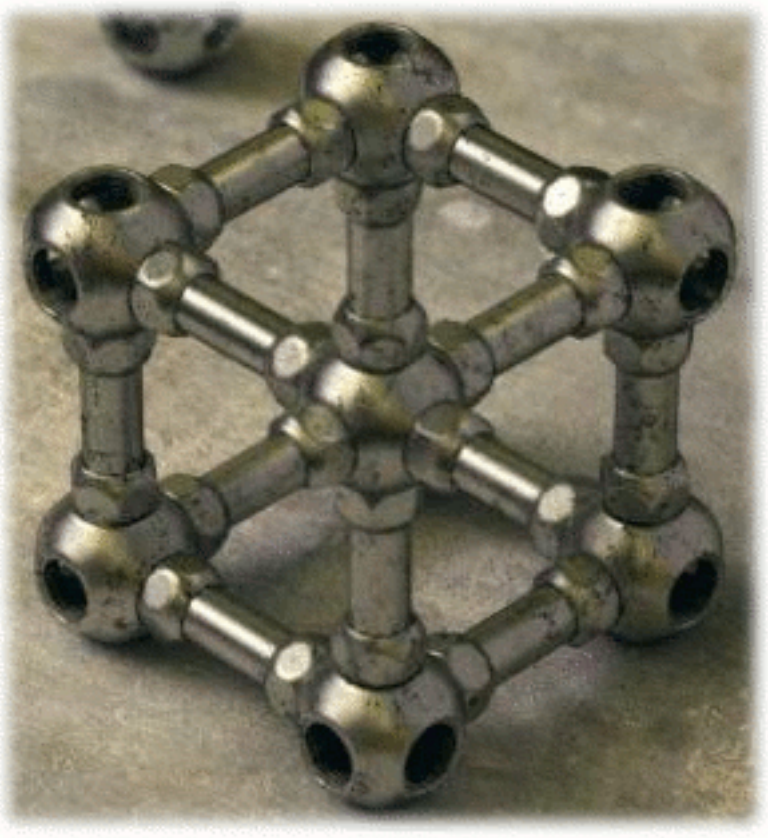

Symbolic Bounded Real-time Dynamic Programming [1]

Author(s):

Karina Valdivia Delgado
Cheng Fang
Scott Sanner
Leliane Nunes de Barros

[1]This work was supported by Fapesp Project LogProb, grant 2008/03995-5, São Paulo, Brazil.

Symbolic Bounded Real-time Dynamic Programming

Karina Valdivia Delgado[1], Cheng Fang[2], Scott Sanner[3], and Leliane Nunes de Barros[1]

[1] University of São Paulo. São Paulo, Brazil
[2] University of Sydney. Sydney, Australia
[3] National ICT Australia. Canberra, Australia

Abstract. Real-time dynamic programming (RTDP) solves Markov decision processes (MDPs) when the initial state is restricted. By visiting (and updating) only a fraction of the state space, this approach can be used to solve problems with intractably large state space. In order to improve the performance of RTDP, a variant based on symbolic representation was proposed, named sRTDP. Traditional RTDP approaches work best on problems with sparse transition matrices where they can often efficiently achieve ϵ-convergence without visiting all states; however, on problems with dense transition matrices where most states are reachable in one step, the sRTDP approach shows an advantage over traditional RTDP by up to three orders of magnitude, as we demonstrate in this paper. We also specify a new variant of sRTDP based on BRTDP, named sBRTDP, which converges quickly when compared to RTDP variants, since it does less updating by making a better choice of the next state to be visited.

1 Introduction

Markov Decision Processes (MDPs) [1] provide a convenient framework for modeling fully-observable stochastic planning problems. In an MDP, the agent computes a policy — a mapping from states to actions — in order to maximize a stream of rewards. A popular approach to policy computation is through a value function — a function that assigns a value to each world state. The computation of the value function can be either synchronous, where all states are updated during each iteration, or asynchronous, where the agent updates some states more than others.

Recent years have seen a resurgence of interest in asynchronous dynamic programming solutions to MDPs [2]. Of particular interest has been the trial-based real-time dynamic programming (RTDP) approach [3] as evidenced by a variety of recent work [4–6]. RTDP algorithms have a number of distinct advantages for practical MDP solutions, specifically: (a) *Anytime performance:* RTDP algorithms can be interrupted at any time, generally yielding a better solution the longer they are allowed to run; (b) *Optimality without exhaustive exploration:* By focusing trial-based search on states reachable from the set of initial states, RTDP algorithms may obtain an optimal policy while visiting only a fraction of the state space.

However, the advantages of RTDP may break down when the transition matrix is not sparse. Yet many MDPs that model interesting real-world problems do not have sparse transition matrices, e.g., elevator scheduling with random arrivals, traffic control with random car movements, and logistical problems with random failures. All of these

problems have the property of *exogenous events* — events beyond the agent's control that can cause arbitrary state changes between time steps that lead to *dense and large transition matrices*. Although dense transition matrices lack structure in terms of sparsity, they often contain factored (symbolic) structure. Based on this last idea, Feng et al (2003) has proposed a factored variant of RTDP named Symbolic RTDP (sRTDP). However they have not shown how this sRTDP behaves when leading with dense transition matrices in large state spaces, as we demonstrate in this paper. Besides, we propose a new variant of sRTDP, a Symbolic Bounded RTDP, which maintains both upper and lower bounds on the optimal value function. We show that with this approach we make less value updates when compared to sRTDP [7].

2 Background

A **Markov decision process** (MDP) is a tuple $\langle S, A, T, R, \gamma \rangle$ [1]. $S = \{s_1, \ldots, s_n\}$ is a finite set of fully observable states. $A = \{a_1, \ldots, a_m\}$ is a finite set of actions. $T : S \times A \times S \rightarrow [0, 1]$ is a known stationary, Markovian transition function. $R : S \times A \rightarrow \mathbb{R}$ is a fixed known reward function associated with every state and action. γ is a discount factor s.t. $0 \leq \gamma \leq 1$ where rewards t time steps in the future are discounted by γ^t. There is a set of initial states $\mathcal{I} \subseteq S$, and a possibly empty set of absorbing goal states $\mathcal{G} \subset S$ where all actions lead to a zero-reward self-transition with probability 1.

A policy $\pi : S \rightarrow A$ specifies the action $a = \pi(s)$ to take in state s. Our goal is to find a policy that maximizes the value function, defined as the sum of expected discounted rewards $V_\pi(s) = E_\pi \left[\sum_{t=0}^{\infty} \gamma^t \cdot r_t \middle| s_0 = s \right]$, where r_t is the reward obtained at time t.

Value iteration (VI) is a *synchronous* dynamic programming (DP) solution to an MDP. Starting with an arbitrary $V^0(s)$, VI performs value updates for *all* states s, computing the next value function $V^{t+1}(s) := \textsc{Update}(V^t, s)$:

$$Q^{t+1}(s, a) := R(s, a) + \gamma \cdot \sum_{s' \in S} T(s, a, s') \cdot V^t(s') \tag{1}$$

$$V^{t+1}(s) := \max_{a \in A} \left\{ Q^{t+1}(s, a) \right\}. \tag{2}$$

This update is known as a *Bellman update*. For discounted problems and certain restrictions of undiscounted problems such as *stochastic shortest path* (SSP) problems [2], $V^t(s)$ converges to the unique optimal value function $V^*(s)$ in the infinite limit of updates, i.e., defining $\epsilon_t = \max_s |V^t(s) - V^*(s)|$ then $\lim_{t \rightarrow \infty} \epsilon_t = 0$. The greedy policy $\pi(s) = \textsc{GreedyAction}(V^t, s)$ w.r.t. V^t and state s is defined as follows:

$$\pi(s) := \arg\max_{a \in A} \left\{ R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') \cdot V^t(s') \right\} \tag{3}$$

For discounted problems, the greedy policy π derived from V^t loses no more than $2\gamma\epsilon_t$ in value over the infinite horizon compared to the optimal policy [1].

Algorithm 1: RTDP

begin
  // Initialize $\hat{V}_u$ with admissible value function
  $\hat{V}_u := V_u$
  while *convergence not detected and not out of time* **do**
    depth $:= 0$
    visited.CLEAR() *// Clear visited states stack*
    Draw s from $\mathcal{I}$ at random *// Pick initial state*
    while $(s \notin \mathcal{G}) \wedge (s \neq null) \wedge$ (*depth* $<$ *max-depth*) **do**
      $depth := depth + 1$
      $visited$.PUSH(s)
      $\hat{V}_u(s) :=$ UPDATE($\hat{V}_u, s$) *// See (1) & (2)*
      $a :=$ GREEDYACTION($\hat{V}_u, s$) *// See (3)*
      $s :=$ CHOOSENEXTSTATERTDP(s, a) *// See (4)*

    // Optimization: end-of-trial update
    // not appearing in the original RTDP
    while $\neg$*visited*.EMPTY*()* **do**
      $s :=$ *visited*.POP()
      $\hat{V}_u(s) :=$ UPDATE($\hat{V}_u, s$)
  return $\hat{V}_u$
end

Asynchronous DP & Real-time DP *Asynchronous* DP methods [2] are a variant of dynamic programming that apply the Bellman update to states in an arbitrary order while still retaining convergence properties under certain conditions. The real-time dynamic programming (RTDP) [3] algorithm (Algorithm 1) is a popular asynchronous DP approach that updates states encountered during trial-based simulations of an MDP. This variant of DP explores the state space in depth-limited trials and performs Bellman updates at each state visited. RTDP selects states to visit by drawing next state samples s' from the transition distribution for the current greedy action a and current state s, i.e.,

$$\text{CHOOSENEXTSTATE}(s, a) := s' \sim T(s, a, \cdot). \tag{4}$$

We say that V_u is an *admissible* upper bound on the optimal value $V^*(s)$ if $V_u(s) \geq V^*(s); \forall s$. Given an admissible upper bound, RTDP converges to the optimal value function in the infinite limit of trials. The primary advantage of RTDP is that its solution is *targeted* to the initial state set. One weakness of RTDP is that unlikely paths tend to be ignored and as a consequence the convergence of RTDP is slow [4]. Thus, some extensions of RTDP were proposed in order to improve the convergence: Labeled RTDP (LRTDP) [4], Bounded RTDP (BRTDP) [5] and Bayesian RTDP [6].

Bounded RTDP (BRTDP) [5] maintains upper and lower bounds on the optimal value function, $V_u(s)$ and $V_l(s)$ respectively, and focus search in areas with high value uncertainty. It was proved that the subsequent updates of upper and lower bounds, de-

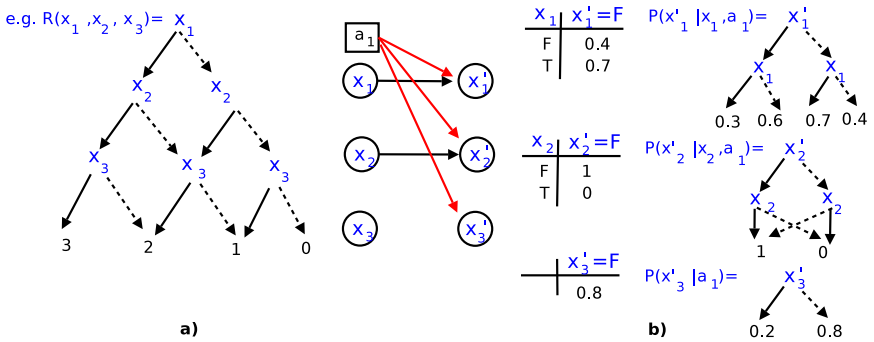

Fig. 1. a)Factored MDP reward and b) transition function example. In all ADDs, true branches are solid, false branches are dashed.

fined as $V_u(s) \geq V^*(s)\ \forall s \in S$ and $V_l(s) \leq V^*(s)\ \forall s \in S$, preserve admissibility and monotonically converge to $V^*(s)$ [5]:

$$\lim_{t\to\infty} V_l^t(s) = V^*(s) = V_u^t(s).$$

The gap between upper and lower bounds provides a measure of the value uncertainty for state s. BRTDP (Algorithm 2) [5] first initializes both $\widehat{V}_u$ and $\widehat{V}_l$ with an admissible upper and lower bounds and then perform many trials. Each trial starts choosing an initial state and for each visited state, upper and lower values are updated and a greedy action is chosen. BRTDP prioritizes the choice of next state according to their *bound gap weighted distribution* $\frac{b(\cdot)}{B}$ (Algorithm 3), where $b(s')$ is given by :

$$b(s') = P(s'|s,a)(\widehat{V}_u(s') - \widehat{V}_l(s')). \tag{5}$$

BRTDP finishes a trial if encounters a goal state, a limited depth is achieved or there is low value uncertainty. BRTDP converges for the simple reason that it still updates all relevant states that RTDP would update, but with a different distribution which biases the updates for more uncertain states (large gap between upper and lower bound), thus reducing the value uncertainty more quickly and leading to faster convergence.

3 Symbolic MDPs and RTDP

Many MDPs often have a natural factored (symbolic) structure that can be exploited in the form of a *factored MDP* [8]. Here, states are represented by vectors $\boldsymbol{x} = (x_1, \ldots, x_n)$ of length n, where each $x_i \in X_i$ and for simplicity, we assume $X_i = \{0, 1\}$; hence the total number of states is $N = 2^n$. Two ways to exploit factored structure in an MDP first jointly introduced in the SPUDD value iteration algorithm [8] are the use of *dynamic Bayesian networks* (DBNs) to represent the transition function and *algebraic decision diagrams* (ADDs) [9] to represent the reward, value function, and *conditional probability tables* (CPTs) that comprise the DBN.

ADDs provide an efficient means for representing and performing arithmetic operations on functions from a factored boolean domain to a real-valued range (i.e., $\{0, 1\}^n \rightarrow$

Algorithm 2: BRTDP

begin
  $\widehat{V}_u = V_u$;
  $\widehat{V}_l = V_l$;
  while *convergence not detected and not out of time* **do**
    depth=0;
    visited.Clear();
    Draw s from I at random;
    $s_0 = s$;
    while *($s \notin G$) and ($s \neq$ null) and (depth $< max\_depth$)* **do**
      depth=depth+1;
      visited.Push(s);
      $\widehat{V}_u(s)$ =`Update` ($\widehat{V}_u, s$) *// See (1) & (2)*
      $\widehat{V}_l(s)$ =`Update` ($\widehat{V}_l, s$) *// See (1) & (2)*
      a =`GreedyAction` ($\widehat{V}_u, s$) *// See (3)*
      s =`ChooseNextStateBRTDP` (s_0, s, a, τ) *// See Algorithm 3*
    while $\neg$ *visited.Empty()* **do**
      s=visited.Pop();
      $\widehat{V}_u(s)$ =`Update` ($\widehat{V}_u, s$);
      $\widehat{V}_l(s)$ =`Update` ($\widehat{V}_l, s$);
  return $\widehat{V}_u$;
end

Algorithm 3: `ChooseNextStateBRTDP`(s_0,s,a, τ)

begin
  $\forall s'$, $b(s') = P(s'|s,a)(\widehat{V}_u(s') - \widehat{V}_l(s'))$;
  $B = \sum_{s'} b(s')$;
  if $B < \frac{\widehat{V}_u(s_0) - \widehat{V}_l(s_0)}{\tau}$ **then**
    return *null*;
  return $s' \sim \frac{b(\cdot)}{B}$;
end

$\mathbb{R}$). They rely on two main principles to do this: (a) ADDs represent a function $\mathbb{B}^n \rightarrow \mathbb{R}$ as a directed acyclic graph (DAG) – essentially a decision tree with reconvergent branches and real-valued terminal nodes and (b) ADDs enforce a strict variable ordering on the decisions from the root to the terminal node, enabling a minimal, canonical diagram to be produced for a given function.

ADDs often provide an efficient representation of functions with context-specific independence [10] and redundant structure. For example, the function $\sum_{i=1}^{3} x_i$ ($x_i \in \{0, 1\}$) represented in Figure 1.a as an ADD exploits the redundant structure of subdiagrams in a DAG to avoid an exponentially-sized tree representation. Unary operations such as scalar multiplication ($\cdot$) and marginalization ($\sum_{x_i \in X_i}$) and binary opera-

Algorithm 4: CHOOSENEXTSTATE_SRTDP($\boldsymbol{x}, a$) $\longrightarrow \boldsymbol{x}'$

begin
  // Sample each next state variable x'_i from DBN
  for *all* X'_i **do**
    $x'_i \sim CPT_a^{x'_i}(\boldsymbol{x})$
  return $\boldsymbol{x}' := (x'_1, \ldots, x'_n)$
end

tions such as addition ($\oplus$), multiplication ($\oplus$), and $\max$ can be performed efficiently on ADDs [9], hence all operations required for DP can be performed directly with ADDs.

A DBN representation of a transition matrix factors the transition probabilities into CPTs $P(x'_i|\boldsymbol{x}_i, a)$ where each next state variable x'_i is only dependent upon the action a and its direct parents $\boldsymbol{x}_i$ in the DBN. Then the transition model can be *compactly* specified as $P(\boldsymbol{x}'|\boldsymbol{x}, a) = \prod_{i=1}^{n} P(x'_i|\boldsymbol{x}_i, a)$ *even when most of the probabilities are non-zero*. Rather than represent each CPT $P(x'_i|\boldsymbol{x}_i, a)$ in a tabular format, ADDs are often more compact as shown in Figure 1.b and facilitate efficient computation.

3.1 sRTDP

Feng et al (2003) proposed the symbolic RTDP (sRTDP) based on the ideas of Hoey et al (1999), i.e., representing the MDP using DBN and ADDs. In this section we explain in details the sRTDP algorithm. We begin by presenting factored forms of (1) and (2):

$$Q^{t+1}(\boldsymbol{x}, a) := R(\boldsymbol{x}, a) + \gamma \sum_{\boldsymbol{x}'} \prod_{i=1}^{n} P(x'_i|\boldsymbol{x}_i, a) V^t(\boldsymbol{x}') \tag{6}$$

$$V^{t+1}(\boldsymbol{x}) := \max_{a \in A} Q^{t+1}(\boldsymbol{x}, a). \tag{7}$$

From these observations, we can now proceed to define the Symbolic RTDP. To do this, we still use the basic RTDP procedure in Algorithm 1, but specify an initialization of $\hat{V}_u$ as an ADD (just a constant if an uninformed upper bound is used), as well as factored versions of the following:

- UPDATE computes the factored RTDP value function update for state $\boldsymbol{x}$. All operations in (6) and (7) can be computed using ADD operations when the reward and value functions and transition CPTs (denoted $CPT_{\boldsymbol{x}}^{a}(X'_i) = P(X'_i|\boldsymbol{x}, a)$ for each X'_i) are ADDs. In RTDP, computations of $V^{t+1}(\boldsymbol{x})$ yield a constant v for a known $\boldsymbol{x}$. Then sRTDP needs only insert this new constant v into the current value function V_{DD}^{t} for state $\boldsymbol{x}$ to obtain V_{DD}^{t+1}. This can also be done efficiently in ADDs.
- GREEDYACTION At the same time that UPDATE is performed, the greedy action $\pi(\boldsymbol{x})$ can be easily computed by keeping track of the $\arg\max_a$ when calculating v.
- CHOOSENEXTSTATE (Algorithm 4) samples a next state $\boldsymbol{x}'$ given a current state $\boldsymbol{x}$ and action a by using DBN structure to sample each next state variable x'_i.

3.2 sBRTDP

We can now define a new symbolic variant of BRTDP (sBRTDP). We modify Algorithm 2 first by initializing $\widehat{V}_u$ and $\widehat{V}_l$ as ADDs. Then we use the same Update and Greedy-Action procedures from sRTDP. The major difference between sRTDP and sBRTDP is

the way they choose the next state, which we explain here in details. We can factor the bound gap weighted distribution $\frac{b(\cdot)}{B}$, that is computed in the BRTDP algorithm, as:

$$P(x'_1, ..., x'_n|\boldsymbol{x}) = P(x'_n|x'_1, ..., x'_{n-1}, \boldsymbol{x})...P(x'_2|x'_1, \boldsymbol{x})P(x'_1|\boldsymbol{x}). \quad (8)$$

Note that we can sample x_1 independently, then condition on it, we can sample x_2, until we reach x_n, we will use this sequential sampling method that is exact. We begin by presenting a symbolic form of $b(s')$ (Equation (5)):

$$b(\boldsymbol{x}') = \prod_{i=1}^{n} P(x'_i|\boldsymbol{x}, a)V_{GAP}(\boldsymbol{x}')$$

where $V_{GAP} = \widehat{V}_u - \widehat{V}_l$. Then starting from x'_1 we do the following:

$$p(x'_1) \propto b(x'_1) = \sum_{x'_i, i\neq 1} \prod_{i=1}^{n} P(x'_i|\boldsymbol{x}, a)V_{GAP}(\boldsymbol{x}').$$

Exploiting the absence of synchronic arcs to decompose, we obtain:

$$b(x'_1) = P(x'_1|\boldsymbol{x}, a) \sum_{x'_2} P(x'_2|\boldsymbol{x}, a) \sum_{x'_3} P(x'_3|\boldsymbol{x}, a) \cdots \sum_{x'_n} P(x'_n|\boldsymbol{x}, a)V_{GAP}(\boldsymbol{x}').$$

Besides the symbolic calculation of $b(s)$, sBRTDP does also the sample of the state variables using an extended *cached ADD*, i.e., an ADD with labeled arcs. In order to do so, sBRTDP constructs a cached V_{GAP} ADD structure: first the transition probabilities of the current state are cached in the arcs of the V_{GAP} ADD and, in a bottom-up fashion, the sums from Equation (9) are also cached in the same structure, starting from inside to outside □s:

$$b(x'_1) = P(x'_1|\boldsymbol{x}, a) \sum_{x'_2} \boxed{P(x'_2|\boldsymbol{x}, a) \textstyle\sum_{x'_3} \boxed{P(x'_3|\boldsymbol{x}, a) \cdots \textstyle\sum_{x'_n} \boxed{P(x'_n|\boldsymbol{x}, a)V_{GAP}(\boldsymbol{x}')}}} \quad (9)$$

Finally, with the resulting V_{GAP} ADD, sBRTDP samples each state variable in the ADD from top to down. This can be done since each box in Equation (9) is precisely one of $P(x'_j|x'_1, ..., x'_{j-1}, \boldsymbol{x})$ in Equation (8), once we have conditioned on the already sampled variables $(x'_1, \cdot, x'_{j-1})$. Then if the factorization in Equation (8) is done in the same order as the ADD variable order, the computation of $P(x'_j|x'_1, ..., x'_{j-1}, \boldsymbol{x})$ always refers to the subdiagram below $(x'_1, ..., x'_{j-1})$ in the ADD and indeed the recursive marginal computations required to compute the true/false probabilities for x_j are already stored locally on the branches of x_j in the subdiagram.

Figure 2 shows an example. Suppose we are in the state $x_1 = T$, $x_2 = T$ and $x_3 = T$; the greedy action is a_1 and we want to choose the next state. First we cache the probabilities from the CPTs (Figure 2.a) in V_{GAP} (Figure 2.b), and then we compute and cache all □s from Equation (9) in V_{GAP} from bottom to up (Figure 2.c). Based on these values we can now sample each state variable (top-down) in turn, conditional on the current values of the above variables that has been sampled. For example using the cached values in V_{GAP} (Figure 2.c), suppose that we have sampled $x'_1 = T$ and

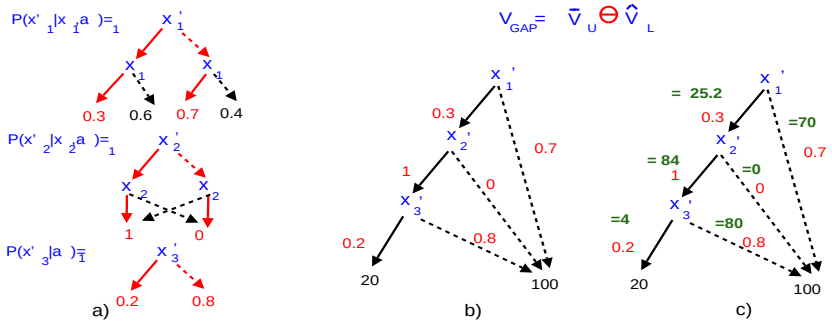

Fig. 2. Computing $b(\boldsymbol{x}') = \prod_{i=1}^{n} P(x_i'|\boldsymbol{x}, a) V_{GAP}(\boldsymbol{x}')$ caching the values in V_{GAP}.

Algorithm 5: `ChooseNextState_sBRTDP`($\boldsymbol{x_0}$,$\boldsymbol{x}$,a, τ,$V_{U_{DD}}$,$V_{L_{DD}}$) $\longrightarrow \boldsymbol{x}'$

begin
  $V_{GAP} = V_{U_{DD}} - V_{L_{DD}}$;
  for *all* X_i' **do**
    put $CPT_a^{x_i'}(\boldsymbol{x})$ in V_{GAP}
  bottom-up in V_{GAP} compute $b(x_1')$ (Equation 9) and cache □s in V_{GAP};
  for *all* X_i' **do**
    // top-down in V_{GAP}
    $B = \sum_{x_i'} b(x_i')$;
    if $B < \frac{\hat{V}_u(\boldsymbol{x_0}) - \hat{V}_l(\boldsymbol{x_0})}{\tau}$ **then**
      return *null*;
    //sampling x_i'
    $x_i' \sim \frac{b(x_i')}{B}$;
    dismiss the other parts of V_{GAP};
  return $\boldsymbol{x}' := (x_1', \ldots, x_n')$
end

$x_2' = F$. The state variable x_3', which is not represented in the ADD in the case of $x_1' = T$ and $x_2' = F$, is sampled with 0.5 probabilities. Algorithm 5 has as input the initial and current state, action a, a constant τ, $V_{U_{DD}}$ and $V_{L_{DD}}$ and returns $\boldsymbol{x}'$ sampled according with the previous described procedure.

4 Empirical Results

First we focus on problems with dense transition matrices where traditional RTDP may be inappropriate (recall that dense matrices can occur in problems with a large exogenous event space). We evaluate Enumerated RTDP and sRTDP on two problems: the logistical problem SysAdmin in the uni-ring configuration with random exogenous equipment failures as defined in [11] and a Traffic MDP for signal control at a single intersection with random exogenous traffic movements [12].

We first analyse properties of the two algorithms on Traffic problem with 12 variables (2^{12} states) in Figure 3 where we show the averaged results of 10 training and

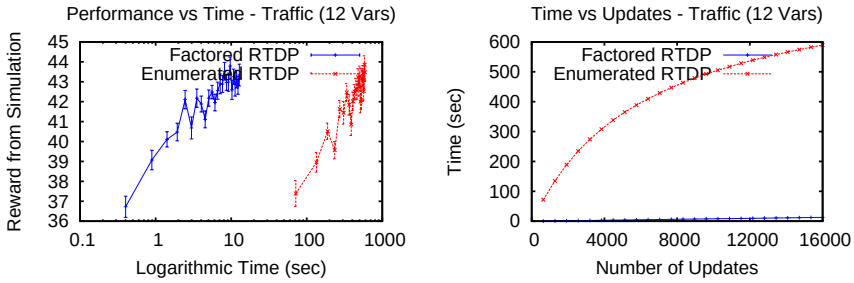

Fig. 3. Various analyses of Factored vs. Enumerated RTDP on the TRAFFIC problem.

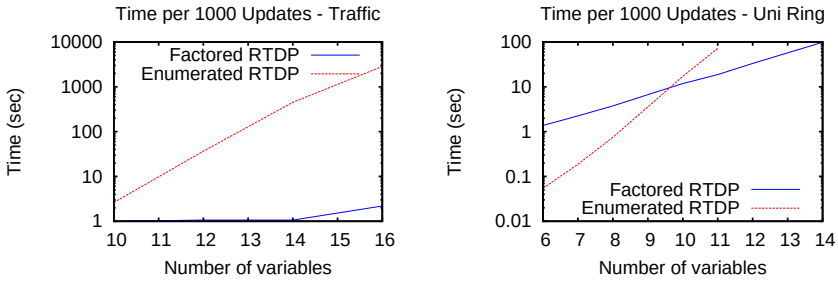

Fig. 4. Analysis of the time per RTDP update for Factored vs. Enumerated RTDP.

policy simulation runs. In the left plot showing performance vs. a log-scale of RTDP training time, we note that sRTDP begins to reach high performance levels *long before* Enumerated RTDP has finished its initial training trials. To explain why, in the plot on the right, we see that the time per RTDP UPDATE for Enumerated RTDP is much larger than sRTDP; Enumerated RTDP slightly speeds up on its later updates as its state cache begins to saturate and yield frequent hits, reducing overhead.

Since time-per-update is the most crucial detail indicating the relative speed of Enumerated and sRTDP, in Figure 4, we provide a time-per-update analysis over two domains varying problems with different number of state variables. Here we see that the overhead of sRTDP may make it slower than Enumerated RTDP on small problems, but as the problem size grows, sRTDP requires substantially less time. For the largest problems (with a large exogenous event space), Enumerated RTDP did not finish in the time limit given by the upper bound of the plots *and* given the log-scale time axis, we note an improvement up to *three orders of magnitude* on large TRAFFIC problems.

Finally we analyse the performance vs number of updates for both symbolic algorithms sRTDP and sBRTDP, for the grid problem 16x16. Figure 5 shows that the sBRTDP converges making $\sim$ 140000 less updates.

5 Concluding Remarks

While RTDP approaches have proved very popular in recent years for their ability to exploit initial state knowledge and sparse transition matrices, the advantages of traditional Enumerated RTDP are often lost on MDPs with dense transition matrices. As we

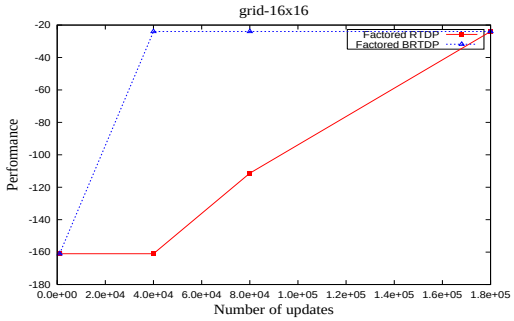

Fig. 5. Symbolic RTDP vs. Symbolic BRTDP.

showed in the experiments sRTDP can speedup learning over its enumerated state counterpart by up to *three orders of magnitude*. Motivated by the results of sRTDP [7], we have proposed a new variant of BRTDP [5] named sBRTDP which is able to converge to the optimal value function with much less updates than sRTDP. This is due to the additional information used for sampling next states. The original idea proposed in this paper is how to sample state variables in the symbolic representation using the ADD structure.

References

1. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley, New York (1994)
2. Bertsekas, D.P.: Distributed dynamic programming. IEEE Transactions on Automatic Control **27** (1982) 610–617
3. Barto, A.G., Bradtke, S.J., Singh, S.P.: Learning to act using real-time dynamic programming. Technical Report UM-CS-1993-002, U. Mass. Amherst (, 1993)
4. Bonet, B., Geffner, H.: Labeled RTDP: Improving the convergence of real-time dynamic programming. In: ICAPS-03, Trento, Italy (2003) 12–21
5. McMahan, H.B., Likhachev, M., Gordon, G.J.: Bounded real-time dynamic programming: RTDP with monotone upper bounds. In: ICML-05. (2005) 569–576
6. Sanner, S., Goetschalckx, R., Driessens, K., Shani, G.: Bayesian real-time dynamic programming. In: 21st IJCAI, San Francisco, CA, USA (2009) 1784–1789
7. Feng, Z., Hansen, E.A., Zilberstein, S.: Symbolic generalization for on-line planning. In: 19th UAI. (2003) 209–216
8. Hoey, J., St-Aubin, R., Hu, A., Boutilier, C.: SPUDD: Stochastic planning using decision diagrams. In: UAI-99, Stockholm (1999) 279–288
9. Bahar, R.I., Frohm, E., Gaona, C., Hachtel, G., Macii, E., Pardo, A., Somenzi, F.: Algebraic Decision Diagrams and their applications. In: IEEE /ACM International Conference on CAD. (1993) 428–432
10. Boutilier, C., Friedman, N., Goldszmidt, M., Koller, D.: Context-specific independence in Bayesian networks. In: UAI-96, Portland, OR (1996) 115–123
11. Guestrin, C., Koller, D., Parr, R., Venktaraman, S.: Efficient solution methods for factored MDPs. JAIR **19** (2002) 399–468
12. Delgado, K.V., Sanner, S., de Barros, L.N., Cozman, F.G.: Efficient Solutions to Factored MDPs with Imprecise Transition Probabilities. In: 19th ICAPS, Thessaloniki, Greece (2009)